UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

Novos Algoritmos para Roteamento de Circuitos VLSI

por

MARCELO DE OLIVEIRA JOHANN

Tese submetida à avaliação,
como requisito parcial para a obtenção do grau de
Doutor em Ciência da Computação

Prof. Ricardo Augusto da Luz Reis
Orientador

Porto Alegre, abril de 2001.

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Johann, Marcelo de Oliveira
Novos Algoritmos para Roteamento de Circuitos VLSI/por Marcelo de Oliveira Johann. – Porto Alegre: PPGC da UFRGS, 2001.
159 p. il.

Tese (doutorado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR-RS, 2001. Orientador: Reis, Ricardo Augusto da Luz.
1. Algoritmos. 2. Roteamento. 3. Síntese de Leiaute. 4. Projeto Físico. 5. CAD. 6. VLSI. I. Reis, Ricardo Augusto da Luz. II. Título.

Em memória de minha mãe
Norma Maria de Oliveira Johann

Agradecimentos

Agradeço em primeiro lugar a todos que constituíram o ambiente onde pude desenvolver minha formação. À minha família, pela cultura herdada e pelos valores que me infundiram o caráter, e às instituições de ensino, que me transmitiram desde cedo o saber e o gosto pela ciência. Agradeço especialmente à Universidade Federal do Rio Grande do Sul, onde cursei a graduação, mestrado e o curso de doutorado que concluo, e particularmente ao Instituto de Informática, cujo ambiente cientificamente profícuo e ao mesmo tempo agradavelmente humano me permitiu ter liberdade para experimentar a inovação e a pesquisa na área de microeletrônica.

Ao meu orientador e amigo Ricardo Reis, pela sua visão otimista, incentivo e apoio nas mais diversas situações, exemplo este indispensável para driblar os obstáculos que se nos interpõe à caminhada. A todos os colegas com quem trabalhei durante estes anos de pesquisa, os quais compartilharam os horários mais absurdos para cumprir as metas a que nos propusemos, e que não ouso mencionar para não ser extenso. Mesmo no silêncio fomos colaboradores. Agradeço a Luigi Carro (IEE-UFGS), Saulo Finco (IM-CTI) e a toda equipe do projeto ÁGATA, pela oportunidade, compreensão e espírito de equipe que marcaram o sucesso desse projeto e contribuíram para a minha experiência prática.

Aos auxiliares de pesquisa do grupo de microeletrônica, que também não mediram esforços na sua dedicação, ajudando na preparação dos diversos sistemas, experimentos, artigos, demonstrações e cursos que realizamos neste grupo. Agradecimento especial a Renato Hentschke, que contribuiu significativamente para o desenvolvimento deste trabalho em particular, com a implementação do algoritmo de duplo assinalamento OTPA do sistema GAROTA, de diferentes testes do algoritmo LCS*, e de um protótipo de roteador com o mesmo algoritmo.

Agradeço ao Prof. Andrew Kahng, que me recebeu na UCLA e custeou minha habitação durante 6 meses, permitindo o desenvolvimento do algoritmo LCS* e o conhecimento de diversos trabalhos e problemas atuais, e aos colegas com quem tive o prazer de trabalhar naquela universidade, especialmente Andrew Caldwell e Dvendra Vidhani.

Agradeço a Marcus Kindel e Leandro Indrusiak, pela oportunidade, companheirismo e acolhida na PUCRS em Uruguaiana. A Regina Ribeiro, pela companhia e conforto em momentos muito delicados de mudança de cidade e ambiente de trabalho, e também pelo exemplo de perseverança e desenvoltura frente aos desafios e novas perspectivas da vida. Agradeço ao meu pai, Antônio, pela sempre interessante companhia, também às minhas irmãs, Ângela e Jane, por todo o suporte diário, e aos inúmeros amigos que conquistei, muitos deles pela Internet. Ao meu primo Geraldo Johann, pelas fotos do circuito realizado na matriz GAAL.

Enfim, agradeço a Deus, cuja existência é a causa necessária da criação, e cuja presença é única esperança e motivo de alegria. Se não fosse por sua vontade nossos projetos não teriam nem êxito, nem sentido algum!

Sumário

Lista de Abreviaturas

ASIC	*Application Specific Integrated Circuit*
BFS	*Breadth First Search*
BTM	*Boundary Terminal Model*
CAD	*Computer Aided Design*
Cap.	Capítulo
CI	Circuito Integrado
CMP	*Chemical-Mechanical Polishing*
CPA	*Cross Point Assignment*
CPU	*Central Processing Unit*
DFS	*Depth First Search*
DIP	*Dual Inline Package*
DRC	*Design Rule Checking*
EDA	*Electronic Design Automation*
Fig.	Figura
FOTC	*Full Over-the-Cell*
GHz	Giga Hertz
GRC	*Global Routing Cell*
HCG	*Horizontal Constraint Graph*
HVH	modelo de canal Horizontal-Vertical-Horizontal
IC	*Integrated Circuit*
*IDA**	*Itertative Deepening A**
ILP	*Integer Linear Programming*
*LCS**	*Lowerbound Cooperative Search*
LEA	*Left-Edge Algorithm*
LEGAL	*Left-Edge Greedy ALgorithm*
MCM	*Multi-Chip Module*
OTC	*Over-The-Cell*
Pág.	página
PCB	*Printed Circuit Board*
PWB	*Printed Wiring Board*
SMD	*Surface Mounting Device*
SRRP	*Single Row Routing Problem*

Tab.	Tabela
TTL	*Transistor-Transistor Logic*
UCLA	*University of California at Los Angeles*
VCG	*Vertical Constraint Graph*
VHV	modelo de canal Vertical-Horizontal-Vertical
VLSI	*Very Large Scale Integration*

Lista de Figuras

Lista de Tabelas

Resumo

Este trabalho apresenta novos algoritmos para o roteamento de circuitos integrados, e discute sua aplicação em sistemas de síntese de leiaute. As interconexões têm grande impacto no desempenho de circuitos em tecnologias recentes, e os algoritmos propostos visam conferir maior controle sobre sua qualidade, e maior convergência na tarefa de encontrar uma solução aceitável. De todos os problemas de roteamento, dois são de especial importância: roteamento de redes uma a uma com algoritmos de pesquisa de caminhos, e o chamado roteamento de área. Para o primeiro, procura-se desenvolver um algoritmo de pesquisa de caminhos bidirecional e heurístico mais eficiente, LCS*, cuja aplicação em roteamento explora situações específicas que ocorrem neste domínio. Demonstra-se que o modelo de custo influencia fortemente o esforço de pesquisa, além de controlar a qualidade das rotas encontradas, e por esta razão um modelo mais preciso é proposto. Para roteamento de área, se estuda o desenvolvimento de uma nova classe de algoritmos sugerida em [JOH 94], denominados LEGAL. A viabilidade e a eficiência de tais algoritmos são demonstradas com três diferentes implementações. Devem ser também estudados mecanismos alternativos para gerenciar espaços e tratar modelos de grade não uniforme, avaliando-se suas vantagens e sua aplicabilidade em outros diferentes contextos.

Palavras-chave: algoritmos, roteamento, síntese de leiaute, projeto físico, *CAD*, *VLSI*

TITLE: “NEW ALGORITHMS FOR VLSI ROUTING”

Abstract

This work presents new algorithms for routing of VLSI integrated circuits and discusses their aplicability in layout synthesis tools. Interconnects have a major impact on circuit performance in recent technologies, and the methods proposed here aim to provide more control over routing quality and convergence in the task of finding an acceptable solution. From a universe of routing problems, two of them are specially important: net-at-a-time routing using shortest path search, and the so called area routing. For the former, we seek the development of a more efficient bi-directional heuristic path search method, called LCS*. Its application on routing exploits specific situations that happens in this domain. The cost model strongly influences the search effort, and also the performance of the routes being discovered, and a more acurate cost model is proposed to optimize this relationship. Yet for area routing, we propose the development of new algorithm class that was suggested in [JOH 94], called LEGAL. The viability and efficiency of these algorithms is demonstrated by using three different implementations. A set of alternative mechanisms for space management and for dealing with non-uniform grid models should also be studied to evaluate the advantages and applicability on other specific contexts.

Keywords: algorithms, routing, layout synthesis, physical design, CAD, VLSI

1 Introdução

A crescente demanda no uso de sistemas computacionais cada vez mais complexos e as diversas vantagens em embutir componentes eletrônicos nos produtos de consumo em massa fizeram da microeletrônica uma área de espantoso crescimento, cujo domínio é crítico para a economia de uma nação. São estes circuitos que tornam possível a evolução da sociedade de informação e a nova economia da qual participamos. O Brasil relegou por longo tempo e desenvolvimento científico e industrial nesta área a um segundo plano. Hoje, a importação de semicondutores está perto de superar a importação do petróleo, ainda o primeiro item da pauta de importações. Assim, é indispensável para o país atualizar-se e dominar o enorme conhecimento envolvido na fabricação e no projeto de tais componentes. Este trabalho é especialmente motivado neste momento pela contribuição científica e tecnológica que pode proporcionar não somente para os problemas que a comunidade internacional encontra, mas para a atualização tecnológica do país nesta tão importante área.

1.1 Evolução dos Circuitos Integrados

Um circuito integrado (*IC, Integrated Circuit*) consiste em um número de componentes eletrônicos construídos pela sobreposição bem definida de diferentes materiais sobre um substrato de silício. Microprocessadores de uso genérico, contendo vários milhões de transistores em um único substrato são utilizados hoje em dia em computadores domésticos. O sucesso da indústria se deve principalmente ao processo de miniaturização. A densidade dos circuitos aumentou cerca de 10 mil vezes nos últimos 20 anos (IBM, [RYA 95]). Esta evolução é modelada pela Lei de Moore [MOO 65], que diz que a capacidade dos circuitos dobra a cada 18 meses. A Fig. 1.1 mostra como exemplo a evolução de microprocessadores da Intel.

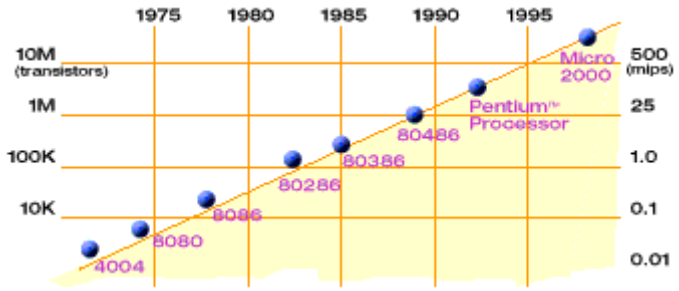


FIGURA 1.1 – Lei de Moore em processadores Intel [INT 2000].

A diversidade de aplicações também é um fator que contribui para esta evolução. Em diferentes produtos de consumo como eletrodomésticos e automóveis, e também em equipamentos profissionais e industriais, existe a necessidade de circuitos integrados para aplicações específicas, os chamados *ASICs*, do inglês, *Application Specific Integrated Circuits*. A presença dos *chips* nos mais variados produtos é conhecida como *ubiquitous computing*, ou computação em toda a parte, fato que vem se acentuando cada vez mais. O projeto de todos estes componentes precisa ser realizado sob fortes compromissos de desempenho, considerando velocidade, dissipação de potência, etc., e garantindo custo e tempo de projeto compatíveis com o mercado e com a obsolescência do produto.

1.2 Ferramentas de Automação de Projeto

Ferramentas de *CAD* (*Computer Aided Design*) t|êm sido utilizadas desde há muito para o projeto automatizado de sistemas eletrônicos, sendo aplicadas em diferentes etapas de sua concepção. O uso destes programas permite que os projetistas trabalhem com informações mais significativas, evitando a necessidade de conhecimento e intervenção nas etapas de projeto mais próximas da tecnologia de fabricação. Essas etapas tratam com um número elevado de elementos, e são mais susceptíveis a erros. Nem todos os problemas podem ser exatamente resolvidos, por limitações de tempo e memória. Quando é possível modelar e solucionar os problemas computacionalmente, realiza-se a tarefa de projeto automaticamente em tempo reduzido.

O crescimento na complexidade desses problemas provocou o surgimento de uma considerável indústria de *software* para automação de projetos eletrônicos, ou *EDA* (*Electronic Design Automation*). A indústria de *EDA* faturou US$ 3 bilhões em 1998 e teve faturamento estimado em US$ $3.5 bilhões para 2000 [BAR 2000], sendo que a licença para rodar um pacote de ferramentas em um ponto de trabalho pode chegar ao preço de US$ 1 milhão. Apesar da sofisticação dessa indústria, o crescimento na produtividade dos projetistas é de apenas 21% ao ano, enquanto o crescimento potencial na complexidade dos sistemas é de 58% ao ano, devido não somente às necessidades de mercado mas também à capacidade de fabricação [BUS 97]. Isto significa que a indústria de semicondutores pode fabricar muito mais circuitos do que se consegue projetar com sucesso[1]. O problema de produtividade se deve a dois principais fatores: **desempenho** e **complexidade**.

O problema de desempenho advém do fato de que é cada vez mais difícil modelar os problemas físicos e elétricos associados ao funcionamento de circuitos com dimensões sub-micrônicas e operando em freqüências da ordem de GHz. Assim, as ferramentas usadas até então são incapazes de proporcionar soluções que considerem tais efeitos e atendam corretamente os requisitos necessários ao funcionamento do circuito. Deve-se reavaliar uma série de simplificações feitas inicialmente para que se tenham ferramentas que permitam observar e controlar os efeitos desejados.

O problema de complexidade resulta do número crescente de componentes em um único circuito, hoje da ordem de dezenas de milhões. As ferramentas de *EDA* já sofrem com esta combinação (por isto o problema de produtividade), cuja solução requer uma metodologia especial. O problema tende a aumentar no futuro próximo, e segundo as previsões feitas pelo *Roadmap* da Indústria de semicondutores norte-americana [SIA 97] (hoje com cooperação internacional), teremos no ano de 2009 tecnologia de fabricação com transistores de 70 nanômetros e circuitos com mais de 500 milhões de transistores operando em freqüências de 2.5 GHz.

Existe um grande conjunto de desafios científicos que precisam ser vencidos para cumprir tais previsões. Certamente um dos maiores está no desenvolvimento de

[1] Soluções práticas para este problema dependem do tipo de circuito. Para projetos de circuitos *high-end* como microprocessadores, as empresas têm aumentado as equipes de projeto, mas isto tende a encarecer mais o produto e a criar problemas de gerenciamento. Já em circuitos de aplicação, a indústria está se adaptando a realidade dos sistemas integrados (*Systems On Chip - SOCs*), utilizando *cores*. Sendo assim, as ferramentas de *EDA* ainda são as grandes responsáveis pelo avanço da tecnologia de projeto.

produtos de *EDA* que sejam capazes de projetar componentes desta complexidade com sucesso [CON 97]. Estes produtos devem prover três novos tipos de capacidades:

- modelar precisamente efeitos antes desprezíveis;
- tratar com eficiência problemas com bilhões de elementos;
- prover uma metodologia de projeto convergente;

Além destes, também merecem destaque especial os problemas de representação e verificação formais, e teste, pois não mais será possível que projetistas humanos conheçam e garantam o funcionamento de algo tão complexo mesmo em níveis altos de abstração. Esta tese se concentra objetivamente na segunda capacidade, isto é, em desenvolver algoritmos que permitam maior controle sobre a solução sendo encontrada e maior eficiência ou rapidez, contribuindo então para os dois problemas fundamentais de desempenho e complexidade das ferramentas de *software*. A convergência de uma metodologia também é considerada através de uma análise teórica de um sistema real.

1.3 Roteamento de Circuitos Integrados

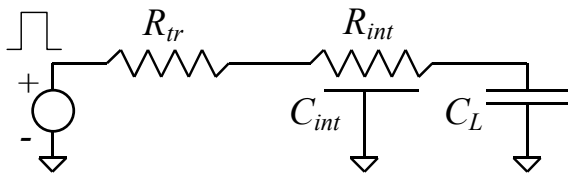
A área de *EDA* engloba uma grande variedade de ferramentas, para solucionar problemas os mais diversos. O objetivo final é a obtenção das máscaras fotolitográficas que são enviadas à fundição. A construção deste leiaute depende da forma como o sistema será implementado, considerando as tecnologias disponíveis, e a síntese física é a parte do processo de projeto que trata de todos os aspectos envolvidos nessa construção. A síntese das interconexões do circuito é a tarefa mais complexa na geração do leiaute do mesmo. Eis a razão: é nela em que está a maior quantidade de detalhes e o maior número de elementos. Um microprocessador tem hoje em dia mais de 1Km de fios de metal acomodados em uma área de cerca de $3.4cm^2$.

O **roteamento** é responsável pela definição das rotas e dos materiais para conexão de pinos de elementos que devem ter o mesmo potencial elétrico. Os elementos podem ser desde transistores isolados, pequenas células lógicas, macro células funcionais, grandes blocos já projetados, até componentes inteiros já empacotados. Os materiais disponíveis para roteamento são um conjunto de camadas metálicas, e furos nas camadas de isolamento para pôr em contato trechos de camadas metálicas verticalmente adjacentes. As rotas devem ser estabelecidas com estas camadas nos espaços permitidos. As conexões não podem se tocar quando não pertencem ao mesmo conjunto eqüipotencial de terminais, e, portanto, são obstáculos naturais entre si. Obstáculos externos são impostos também pela forma de implementação ou estilo de leiaute, tanto como limitação na área, como com a presença de objetos dentro dela.

O roteamento possui diversas aplicações em diferentes instâncias de um projeto. Fatores como a forma das regiões para roteamento e a posição dos terminais a conectar condicionam fortemente a definição de cada problema. Desde a década de 60 se usam programas de computador para roteamento automático de *PCB*s, e eles evoluíram de modo que dispomos de uma boa variedade de algoritmos para diferentes problemas, como será apresentado no Capítulo 2. Apesar disto, as conexões voltaram a ser objeto de grande atenção pois passaram a ter maior impacto no desempenho dos circuitos em tecnologias de fabricação recentes, devido aos atrasos que impõem aos sinais.

Um modelo de atraso bastante conhecido é mostrado na Fig. 1.2, uma

aproximação apresentada em [BAK 90] (cap. 5) e que está de acordo com os modelos de Sakurai. O atraso medido entre tempos de subida e descida em 50% de Vdd é usado pois pode-se somá-lo em estágios subseqüentes para obter o atraso final. O **atraso de uma porta** lógica depende da resistência de saída e da carga em sua saída, carga esta que inclui as capacitâncias do estágio de saída da porta, dos estágios de entrada das portas que ela ataca, e das rotas que as conectam. É possível projetar as portas para que sejam adequadas às cargas que lhes serão impostas, reduzindo o atraso do circuito[2]. Já o **atraso das conexões** depende da resistência das interconexões e das capacitâncias de interconexão e dos estágios de entrada a ela conectados. Para alterar este atraso, não basta reprojetar as células, mas é necessário reprojetar as próprias interconexões. É esse o atraso que tem crescido relativamente nas tecnologias recentes, de modo que passou a dominar o atraso do circuito nas gerações atuais, que têm menor tamanho do canal do transistor, conforme mostra a conhecida Fig. 1.3.



$$T_{50\%} = T_{gate} + T_{int}$$
$$T_{gate} = R_{tr}\,(0.7\,C_{int} + 0.7\,C_L)$$
$$T_{int} = R_{int}\,(0.4\,C_{int} + 0.7\,C_L)$$

FIGURA 1.2 – Atraso de porta e atraso de interconexão.

Novos materiais como tungstênio (para os contatos e conexões locais), cobre (em lugar de alumínio), isolantes com menor permeabilidade elétrica, e novas técnicas como o preenchimento de valas com metal em vez do tradicional processo de corrosão permitem melhorar a transmissão dos sinais nas conexões, reduzindo a sua resistência. O problema é atenuado, mas sua solução continua dependendo de novas metodologias, onde o projeto deve ser centrado nas interconexões, e não apenas nas células.

Além do aumento do atraso, outro efeito indesejado ocorre pelo acoplamento capacitivo entre conexões vizinhas, que é a interferência, ou *crosstalk*. Quando um sinal em uma conexão varia, uma conexão vizinha fortemente acoplada sofre interferência em forma de ruído. A amplitude dessa interferência não pode passar de uma certa percentagem da alimentação, sob pena de causar transições espúrias no circuito. Satisfazer esta margem para *crosstalk* impõe limites à proximidade das conexões, e ao comprimento máximo que podem compartilhar de vizinhança.

Outros efeitos físicos que tomam importância são a eletro-migração, pela qual as moléculas de metal se deslocam ao longo do fio quando a capacidade de corrente deste não é suficiente, causando sua ruptura, e a contribuição da indutância para o atraso das conexões. Em alguns trabalhos, as conexões são modeladas como linhas de transmissão, pois o caminho de retorno da corrente passa a ser significativo. Apesar de estes efeitos serem bem estudados na literatura, considerá-los nas ferramentas de síntese pode não ser trivial. Uma questão simples, importante, e difícil de responder é saber qual o tamanho da lógica que pode ser sintetizada sem considerar tais efeitos.

[2] O reprojeto da célula é essencialmente reduzir a resistência de saída pelo uso de transistores maiores. Normalmente as bibliotecas de células comerciais usam transistores de saída com larguras de 10 a 100 vezes maiores do que a largura mínima permitida na tecnologia.

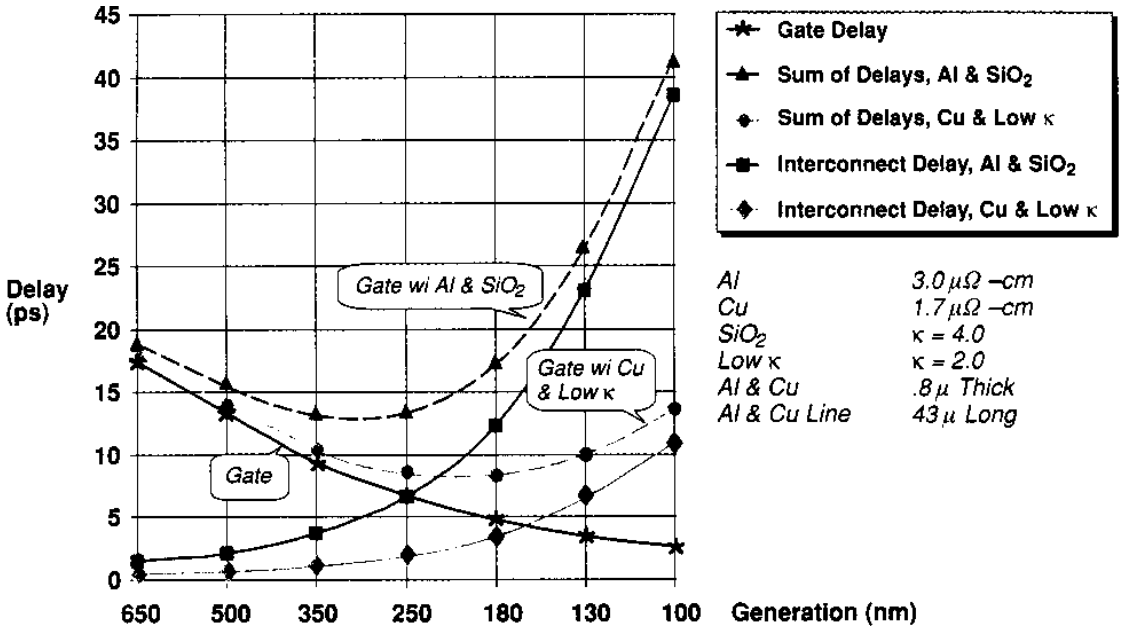


FIGURA 1.3 – Atraso de conexões passa a ser dominante [SIA 97].

1.4 Tecnologia de fabricação de interconexões

A tecnologia de fabricação de conexões evoluiu enormemente. Inicialmente, apenas uma camada metálica era disponível para realizar o roteamento. As tecnologias hoje empregadas têm normalmente 5 ou 6 camadas metálicas sobre os transistores, e este número tende a aumentar para 8 ou 9 em 2009. Processos avançados de planarização com polimento mecânico-químico (*CMP*) são usados para fabricar esta estrutura de interconexões sem distorções.

Em geral, o desenvolvimento da indústria de semicondutores tem sido baseado na redução das dimensões dos elementos fabricados, ou *scaling*. Com isto, cabem mais transistores no mesmo espaço, e os transistores se tornam mais rápidos. A quantidade de conexões, entretanto, aumenta mais do que a de componentes ativos, já que é preciso sempre conectar todas as partes do circuito entre si. Assim, para que se possa realizá-las sem gastar muita área adicional, ao longo do tempo foram sendo acrescidas as demais camadas metálicas, a uma taxa aproximada de 0.75 por geração [THE 2000]. Este fato é representado na Fig. 1.4. A quantidade de conexões e seu comprimento podem ser avaliados com base na Lei de Rent [BAK 90] (Cap. 9) ou utilizando modelos recentes mais precisos, como [DAV 98].

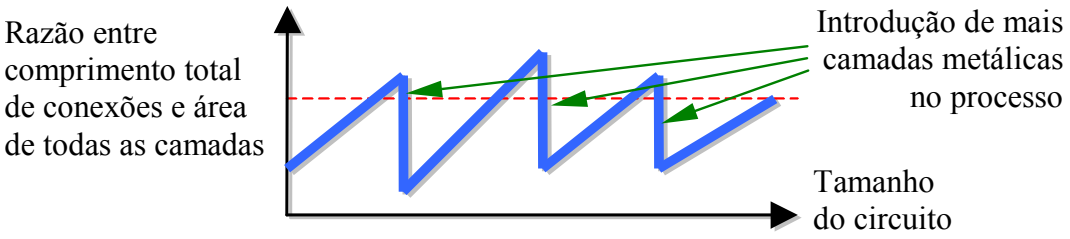


FIGURA 1.4 – Necessidade de conexão e inclusão de novas camadas.

Inicialmente as regras de tecnologia especificavam maior largura mínima e espaçamento para as camadas metálicas superiores por motivos de relevo. Estes

problemas já foram amenizados com tecnologias como a planarização por polimento e o preenchimento de valas com metal. Atualmente, há tecnologias que permitem a fabricação de circuitos com até 6 camadas metálicas com as mesmas dimensões mínimas (Fig. 1.3a [EDE 97]). Entretanto, tendo o tamanho do circuito permanecido mais ou menos constante, os sinais passaram a percorrer caminhos muito mais longos relativamente ao tamanho dos componentes e à freqüência de operação, impondo maior atraso, como já mencionado. O principal meio de reduzir esse atraso está no uso do que se chama *reverse scaling*, ou miniaturização inversa. As últimas camadas metálicas são propositadamente mais espessas e afastadas, como em circuitos de tecnologias mais velhas, para reduzir a resistência de interconexão, como na Fig. 1.5b de [BRA 98]. Outras imagens e dados sobre *RC* em miniaturização normal e inversa podem ser encontrados em [STA 98].

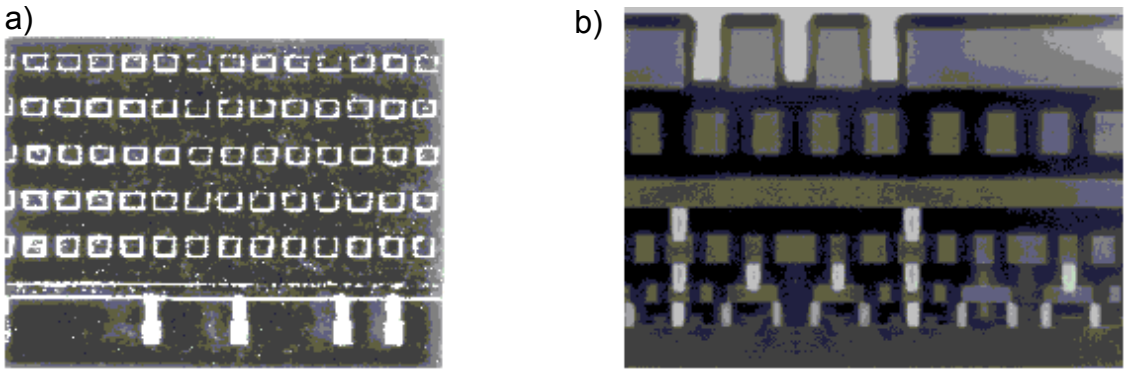


FIGURA 1.5 – Tecnologias com passo mínimo [EDE 97] e otimizada RC [BRA 98].

Uma forma inicial de *reverse scaling* foi o aumento da razão de aspecto (altura/largura) nas interconexões. A razão de aspecto pode ser aumentada para diminuir a resistência (Fig. 1.6b), mas a capacitância lateral também aumenta linearmente, trazendo problemas maiores de acoplamento entre sinais adjacentes. Não é vantajoso se ter razão de aspecto maior do que 2. Assim, para que os sinais globais do circuito possam vencer a enorme distância relativa entre seus componentes, as camadas metálicas superiores deverão ser otimizadas tendo ao mesmo tempo maior altura, largura e espaçamento (Figs. 1.5b e 1.6c), diminuindo a sua densidade de conexões e portanto também o impacto de sua inclusão em novos processos [THE 2000].

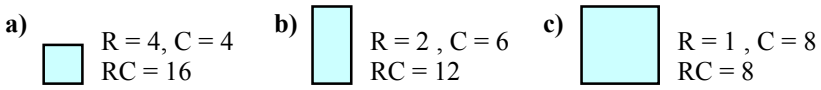


FIGURA 1.6 – Redução de RC com aumento da secção.

Encontrar as dimensões corretas de dielétrico, altura, largura e espaçamento de cada camada metálica para uma determinada tecnologia e classe de circuitos é um problema conhecido como *interconnect tuning* [KAH 99]. A Fig. 1.7 mostra, em passos (largura + espaçamento) horizontal e vertical, os pontos de tempo de chaveamento adequado a uma conexão de tamanho igual ao lado do circuito, para limites de *crosstalk* de 10%, 20% e 40% de Vdd em diferentes tecnologias [RAH 95], com materiais convencionais (alumínio e óxido de silício) e com novos materiais (cobre e dielétrico de baixa constante *k*) respectivamente. Observa-se o fato desagradável de que as conexões globais precisam aumentar de tamanho para as tecnologias onde os transistores

diminuem (o valor nominal da tecnologia é o comprimento desenhado do canal).

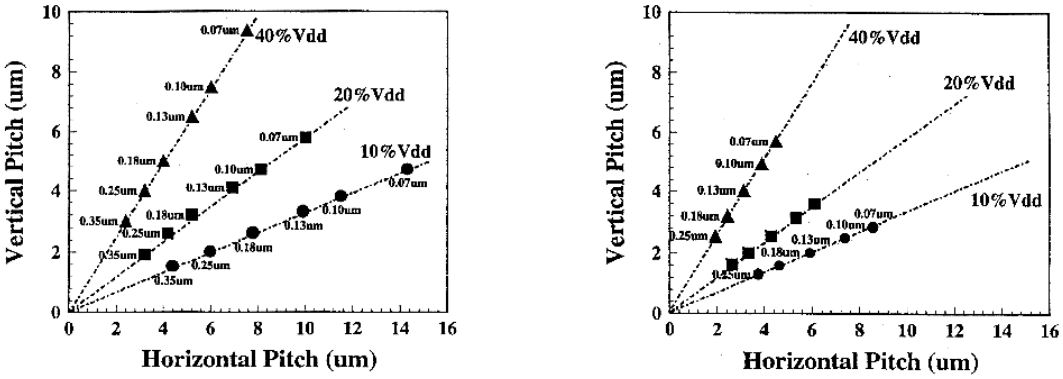


FIGURA 1.7 – Tamanhos de conexão adequados a sinais globais [RAH 95].

Para este trabalho, que não busca considerar os detalhes de efeitos físicos e elétricos, mas apenas seu relacionamento com os algoritmos, esse esquema de miniaturização tem duas implicações. Primeiro, observar que as camadas metálicas em circuitos *VLSI* possuem características diferentes, e estas devem ser consideradas apropriadamente. Em segundo lugar, mostrar que um circuito *VLSI* ainda é um domínio essencialmente bidimensional. A pequena altura em relação à área já o torna um domínio de **duas dimensões e meia**, onde apenas se representa os efeitos tridimensionais (como capacitâncias parasitas ou interferências eletromagnéticas) enquanto ainda se dá mais importância à sua característica planar. Considerando também as características e diferentes aplicações das camadas metálicas, conclui-se que continua sendo de interesse algoritmos eficientes de roteamento em dois níveis, pois normalmente se estará tratando problemas desta natureza, mesmo nas próximas gerações de circuitos.

Existe também modelos para roteamento com larguras de conexão variáves, para quando é apropriado ter conexões com características diferentes na mesma camada. Este grau de liberdade certamente permite sintetizar conexões com características ótimas, mas ao mesmo tempo dificulta sua modelagem, estimativa, e também o aproveitamento de espaço e alinhamento de contatos com outras camadas.

1.5 Objetivos e Divisão do Trabalho

O Capítulo 2 dá uma visão abrangente sobre problemas de roteamento, sua definição, e em especial algoritmos usados na sua solução, sendo essencialmente didático. O Capítulo 3 enriquece este universo apresentando as situações práticas que ocorrem na implementação de sistemas de roteamento, cujo efeito vai desde o aumento no tamanho do código até a dificuldade de encontrar uma solução possível. É dado destaque à metodologia e ao problema de convergência deste processo. Dois sistemas implementados no âmbito deste trabalho são apresentados e servem como exemplo para os assuntos abordados, demonstrando particularmente a construção de um fluxo eficiente de roteamento e geração de leiaute.

O Capítulo 4 aprofunda o estudo de algoritmos de pesquisa de caminhos já que estes são intensivamente utilizados para roteamento. Os algoritmos são apresentados em

ordem histórica e as propriedades da pesquisa heurística são resumidas para que então se demonstre o novo algoritmo LCS*, uma das contribuições importantes desta tese. O Capítulo inclui ainda vários testes com este algoritmo em diferentes domínios.

A realização de roteamento usando algoritmos de pesquisa é tema do Capítulo 5, que utiliza dados do sistema MARTE, de uma implementação de roteador com algoritmos A* e LCS*, entre outros, e considera roteamento global e detalhado. A principal contribuição está, contudo, na proposta de um modelo de custos para roteamento global que permite maior controle sobre o objetivo de cada pesquisa, ao mesmo tempo em que evita que a função heurística do algoritmo tenha pouco poder de poda. Este modelo, portanto, contribui para os dois principais problemas mencionados anteriormente, de precisão de resultados e velocidade com que se os obtém.

O Capítulo 6, enfim, dedica-se ao problema de roteamento detalhado de área, o qual é difícil por não apresentar divisões naturais. É proposto o uso do algoritmo LEGAL, que faz o roteamento de todas as conexões simultaneamente em tempo aproximadamente linear em relação à área do circuito. A eficiência deste algoritmo é avaliada pelos resultados de três diferentes implementações: no sistema MARTE, no sistema GAROTA e com uma versão simplificada denominada ***ilegal***. Os resultados indicam que se pode obter com este algoritmo um roteamento de boa qualidade em um tempo de *CPU* ordens de grandeza menor do que com algoritmos de pesquisa ou otimização combinatória.

Os anexos incluem, além de algumas figuras grandes de leiautes de circuitos, quatro artigos publicados sobre os sistemas e algoritmos em questão, e o código fonte de uma implementação do algoritmo LEGAL. A leitura do artigo sobre o sistema GAROTA é recomendada para melhor compreensão do Capítulo 3, e algumas figuras são referenciadas no texto. O artigo sobre o sistema MARTE (*FOTC*) serve como ilustração e referência, mas sua leitura não é indispensável. Os artigos sobre o algoritmo LCS* também são apenas complementares, e incluem outros resultados e principalmente as provas formais de que LCS* é completo e admissível.

2 Algoritmos de Roteamento

Considerando sistemas atuais que envolvem dezenas de milhões de transistores, há claramente uma grande hierarquia tanto em nível lógico como em nível físico, com diferentes tecnologias de fabricação das conexões. Para cada parte desta hierarquia há um conjunto de problemas característicos de roteamento, alguns equivalentes, outros diferenciados. A definição destes problemas provoca a necessidade de diferentes algoritmos, mas há também um conjunto de técnicas genéricas que podem ser aplicadas a diversos casos. Este Capítulo define os problemas mais importantes e apresenta os algoritmos conhecidos para sua solução.

2.1 Classificação e Terminologia

Os problemas e os algoritmos de roteamento podem ser classificados por diversos critérios. São apresentadas algumas classificações importantes, bem como termos notáveis empregados historicamente.

2.1.1 Classificação de roteamento por objetivos

Em primeiro lugar, visando distinguir diferentes objetivos, usa-se a classificação de [PRE 88], onde o roteamento pode ser: detalhado, global, ou especializado (Fig. 2.1).

O **roteamento detalhado** especifica completa e detalhadamente as rotas de cada conexão, seus materiais, furos, posições e dimensões exatas. Sendo um problema muito complexo para o circuito inteiro, é normalmente tratado por partes. Assim, o roteamento detalhado completo de um circuito é formado pela simples união das soluções de um grande conjunto de roteamentos detalhados restritos a pequenas porções do circuito.

O **roteamento global** é a etapa responsável por dividir o problema de roteamento de todo o circuito para um conjunto completamente especificado de problemas de roteamento detalhado, suficientemente pequenos para que sejam tratáveis. As conexões mais longas são decompostas em pequenas conexões locais. São tarefas também características do roteamento global a definição de espaços para que as conexões sejam acomodadas, e o gerenciamento da forma de cada rede de múltiplos terminais.

O **roteamento especializado** é aquele necessário para conexões com características especiais, como alimentação, relógio, barramento, ou sinais de entrada e saída, ou também em etapas específicas de fabricação, como o empacotamento.

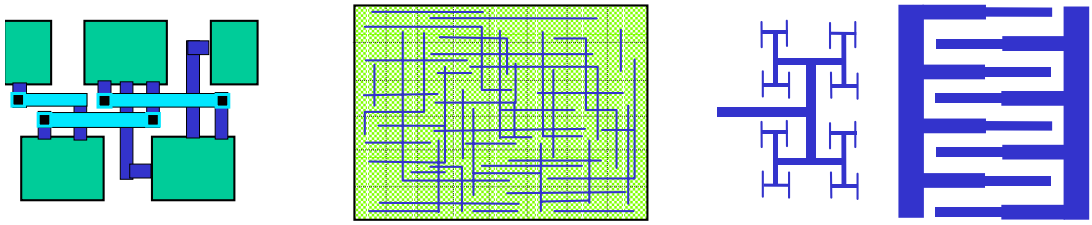


FIGURA 2.1 – Diferentes objetivos de roteamento.

2.1.2 Classificações de roteamento quanto ao espaço existente

O número de conexões em um circuito cresce aproximadamente de acordo com o número de componentes a conectar. Entretanto, com o maior número de componentes, se tem maior distância entre eles, e como conseqüência, maior tamanho das conexões. Assim, a área necessária para o roteamento cresce mais se comparada à área necessária para as células. Com o tempo, as tecnologias de fabricação passam a oferecer mais camadas de interconexão para que o roteamento possa ser realizado sobre a área dos elementos primitivos do circuito.

Quando há espaços dedicados somente para as conexões entre as células, estes espaços são denominados de **canais de roteamento** ou **caixas de conexão** *(switch boxes*) (Fig. 2.2a ou 2.2b), como caracterizado adiante. O termo roteamento sobre células, ou *OTC* (*over-the-cell*), aparece pela primeira vez em [DEU 80], e designa, neste mesmo modelo, a realização de algumas conexões por sobre a área das células, quando há espaço e camadas disponíveis para isto [CON 90]. Modelos como o roteamento de canal ou roteamento planar são de grande importância mesmo em abordagens onde todo o roteamento é realizado sobre as células, e onde há várias camadas disponíveis.

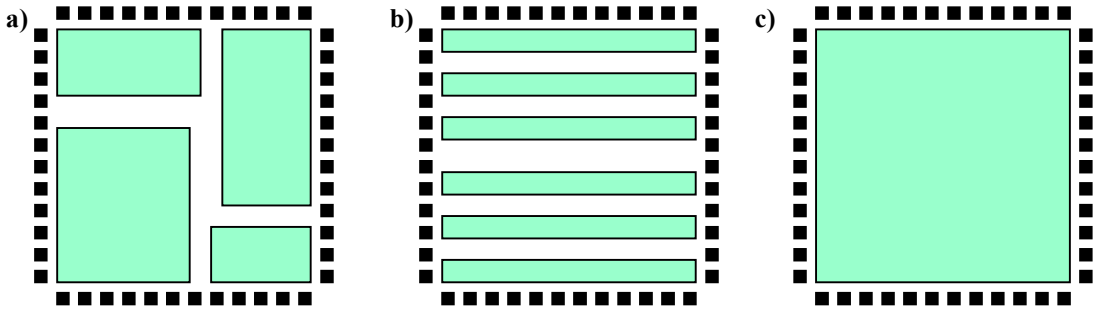


FIGURA 2.2 – Espaços disponíveis para roteamento.

Quando o roteamento é realizado sem canais, por sobre células ou blocos do circuito, diz-se que é um **roteamento de área** (*area routing,* Fig. 2.2c). Dois fatores o caracterizam: a inexistência de limites parciais nesta área, como é o caso em outros modelos; e a presença de obstáculos, ou restrições, nas camadas que precisam ser usadas para o roteamento. Este tipo de roteamento se tornou importante na atualidade devido a presença de maior número de camadas metálicas. Em circuitos maiores é comum a área acomodar blocos arbitrários (Fig. 2.2a) e células de altura padrão (Fig. 2.2b). Este modelo misto é denominado por alguns autores de *MBC*, *Mixed Block and Cell*.

2.1.3 Classificações de roteamento quanto à modelagem do espaço

Na maioria dos casos, para os sinais ordinários do circuito, todas as conexões têm a mesma largura e os terminais estão alinhados. Neste caso o roteamento pode ser baseado em uma **grade**, onde se abstrai o tamanho físico exato, e consideram-se apenas as unidades de passo da grade. O roteamento será também **simbólico**, sobre uma grade virtual, se o restante do leiaute também estiver sendo projetado simbolicamente, em diagramas de barras. Roteamento **topológico** ocorre sempre que se pode abstrair as posições físicas dos elementos (em coordenadas) e tratar o relacionamento relativo entre

eles. Por exemplo, avalia-se o número de cruzamentos necessários entre várias conexões em uma área restrita, ou como minimizá-los.

Em circuitos de alto desempenho, ou para redes críticas em particular, é necessário usar conexões com largura ou espaçamento variáveis, reduzindo a constante *RC* associada e também o acoplamento capacitivo entre conexões vizinhas. Também se pode usar conexões em ângulos diferentes de 90^o. No roteamento de um canal, por exemplo, o uso de conexões inclinadas em 45 graus reduz consideravelmente a altura deste. De acordo com o espaço existente e com o modelo que se aplica sobre este, define-se um **problema de roteamento**. Tipicamente, entre os parâmetros que condicionam a definição de um problema, se tem:

- A forma das regiões de roteamento em cada camada;
- O número de camadas de metal para conexões;
- O modelo de células e posição dos terminais;
- Presença de terminais eqüipotenciais, ou estruturas de passagem;
- Restrições para a colocação de furos de contato (vias);

Para roteamento global, pode-se modelar o espaço como um grafo ou como uma grade regular. No caso de modelagem por grafo, os vértices podem representar tanto os canais e *switchboxes*, quanto as interseções entre estes. Em ambos os modelos, os objetivos principais são:

- encontrar o caminho mais curto para cada rede;
- reduzir o tamanho do circuito como um todo;
- balancear o congestionamento das regiões.

2.1.4 Classificação dos algoritmos de roteamento quanto ao processamento

Podemos classificar os algoritmos de roteamento pelo modo com que tratam a solução no que diz respeito ao relacionamento entre as diversas redes que precisam ser conectadas no mesmo espaço. Assim, os algoritmos podem ser:

- **incrementais** ou **seqüenciais** - aqueles que realizam as conexões uma a uma até completar todas ou não ser mais possível efetuá-las;
- **integrais** ou **paralelos** - são os que consideram ao mesmo tempo todas as conexões necessárias e buscam uma solução global ao problema;
- **refinadores** ou **iterativos** - partem de uma solução inicial e a modificam através de operações de "desfaz e refaz" até ser obtida a melhor solução;

A classe de algoritmos seqüenciais apresenta entre outros o problema da ordenação. Tomando cada conexão sem considerar as demais, sua realização pode impedir outras se bloquear trechos do único caminho possível para conexões posteriores. Isto significa que mesmo que haja uma combinação de caminhos que permita realizar todas as conexões de um circuito, num algoritmo seqüencial esta solução pode não ser encontrada em conseqüência da ordem em que elas foram testadas.

2.1.5 Classificação dos algoritmos quanto à sua aplicação

Os algoritmos de roteamento se dividem em dois grandes grupos quanto à sua aplicação em problemas diferentes. Os **algoritmos genéricos** podem ser em princípio aplicados a qualquer problema de roteamento. Por sua vez, os **algoritmos restritos** se aplicam somente a um determinado problema ou conjunto pequeno de problemas. Sua vantagem reside no fato de que, explorando características específicas da situação a que se aplicam, são capazes de encontrar soluções com maior qualidade e eficiência. As desvantagens são que o alto grau de especialização dificulta mínimas adaptações e em muitos casos permite a ocorrência de instâncias consideradas ruins, para as quais o algoritmo não apresenta boa solução.

2.2 Algoritmos de roteamento genéricos

Há dois tipos principais de algoritmos de roteamento genéricos, os baseados em pesquisa de caminhos e os baseados em geometria. Os primeiros são muito mais populares, e baseiam-se na pesquisa de caminhos em grafos, os quais podem modelar situações as mais diversas. Em roteamento, costuma-se chamá-los de *maze-routers* ou *maze-runners*. Para roteamento detalhado, a área é representada por um grafo regular de granularidade fina, chamado também de grade.

Já os algoritmos baseados em geometria não requerem um espaço discreto, e utilizam as posições reais dos elementos. Entretanto, este grau de liberdade a mais geralmente impõe severas penas à qualidade do seu resultado. Já que o modelo do espaço é uma das principais diferenças, pode-se pensar em uma generalização de pesquisa por caminhos, a qual pode usar qualquer um dos modelos. De fato, algumas ferramentas comerciais usam algoritmos geométricos em conjunto com algoritmos de pesquisa para superar obstáculos, mas não é comum se encontrar referência a este tipo de processo na literatura.

Pode-se definir uma terceira classe de algoritmos genéricos, representada pelo algoritmo hierárquico. Na verdade, este é um processo de particionamento de conexões, e todo o processo de particionamento quando levado ao extremo termina por definir a posição de todos os elementos, que no caso são as conexões. Estas três classes de algoritmos genéricos são revisadas a seguir.

2.2.1 Algoritmos genéricos baseados em pesquisa de caminhos

Os *maze routers* são algoritmos seqüenciais que fazem a pesquisa pelo caminho mais curto entre dois pontos em uma grade, a qual poderia ser um grafo qualquer, pesquisa feita em largura. Cada célula desta grade pode estar livre ou bloqueada. O algoritmo básico é conhecido na área de automação de projeto eletrônico como algoritmo de Lee (1961), e consiste em marcar as células não bloqueadas a partir da origem até o destino, com números de uma seqüência, de tal forma que seja possível identificar o menor caminho a partir do destino. Esta técnica deriva dos algoritmos de inteligência artificial e pesquisa operacional, mais especificamente de *Breadth-First Search* (*BFS*). As principais vantagens deste algoritmo são que se existe um caminho entre a origem e o destino, ele é encontrado, e o caminho encontrado é o mais curto. Em contrapartida, estes algoritmos tem as seguintes desvantagens:

- tempo de processamento, quadrático em relação ao comprimento das conexões;
- muita memória requerida para armazenar o espaço em grade;
- ordem das conexões, a qual pode inviabilizar o roteamento, perdendo a garantia de encontrar a solução para um conjunto de redes, mesmo quando existe;

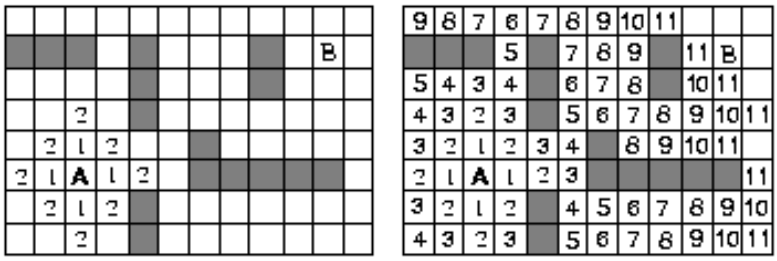


FIGURA 2.3 – Marcação com seqüência de números pelo algoritmo de Lee.

Mesmo com estes inconvenientes, esta é sem dúvida a técnica mais amplamente utilizada até os dias de hoje para roteamento em geral, só perdendo para os algoritmos de roteamento de canal, que estiveram em evidência com as metodologias *Standard Cell*. Alguns dos fatores que contribuem para sua importância são que se pode obter o caminho mais curto ou o único possível para uma determinada conexão que é crítica, pode-se utilizá-la em qualquer aplicação e em conjunto com qualquer outro método, e permite que conexões individuais sejam desfeitas ou refeitas independentemente. Existem diversas técnicas para aumentar sua eficiência e minimizar suas desvantagens. As técnicas de pesquisa em geral, e de roteamento com algoritmos de pesquisa são vistas detalhadamente em Capítulos subseqüentes.

2.2.2 Algoritmos genéricos baseados em geometria

Dentre os baseados em geometria, destacam-se os algoritmos *line-probe* e *line-expansion* (ver em [PRE 88] cap. 5). Ambos traçam linhas da origem para o destino. Se estas linhas intersecionam algum obstáculo, são escolhidos pontos de escape de onde novas linhas são traçadas. Diversas outras abordagens geométricas são também propostas. Em geral, elas apresentam menor consumo de memória e menor tempo de execução, mas pecam igualmente por serem seqüenciais. O uso de algoritmos seqüenciais encontra sua utilidade em avaliação de roteabilidade, geração de uma solução inicial de roteamento global, e também no roteamento das ultimas redes que não puderam ser roteadas por um algoritmo paralelo. Para problemas grandes de roteamento detalhado de todas as conexões, entretanto, deve-se evitar tanto quanto possível o seu uso. Em especial, para áreas congestionadas, o armazenamento e identificação dos obstáculos em algoritmos geométricos é mais difícil.

2.2.3 O algoritmo hierárquico

Embora seja classificado como um algoritmo de roteamento de canal em [PRE 88] e [SHE 93], na verdade este é um algoritmo de roteamento genérico, tendo sido desenvolvido para o roteamento de *gate arrays* e utilizado com eficiência também em roteamento de canais e *switch boxes* [BUR 83a] [BUR83b]. Sua abordagem hierárquica na forma de divisão e conquista aliado ao modelo da área com capacidades de roteamento horizontal, vertical e possibilidades de mudança de camada o tornam

atraente também para roteamento de área. O algoritmo faz o roteamento sobre uma grade de duas camadas com orientação restrita. Sua técnica se baseia na redução do problema de roteamento de uma grade (*m x n*) para o roteamento em grade (*2 x n*). Isto é feito dividindo-se a altura *m* em dois subgrupos de tamanho *m/2* sucessivamente, solucionando o problema entre cada par de subgrupos a cada vez. Para o problema de roteamento em grade *2 x n* são apresentadas duas abordagens. A primeira é baseada na solução exata do problema de roteamento em uma grade *2 x 2* usando programação linear e estendendo hierarquicamente estas soluções. A segunda baseia-se no roteamento mais tradicional sobre grade *2 x n* rede a rede de forma seqüencial. Isto é realizado porque o problema de árvores de Steiner com custos arbitrários pode ser solucionado em tempo linear quando a altura da grade é somente 2.

2.3 Roteamento de Canal

Um dos modelos mais empregados no roteamento de circuitos *VLSI* é o modelo de canal. Um canal é uma região retangular, tendo uma das dimensões geralmente bem maior do que a outra, sendo a maior chamada de comprimento, e a menor de altura. Neste problema temos um conjunto de terminais dispostos nas duas margens, ao longo do comprimento do canal, os quais devem ser conectados na sua área interna, tipicamente por duas camadas. A Fig. 2.3 mostra este modelo e sua nomenclatura.

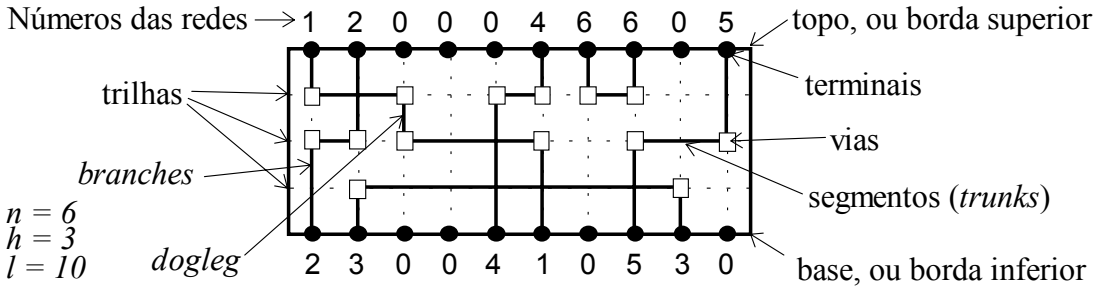


FIGURA 2.3 – Modelo e terminologia de roteamento de canal.

2.3.1 Custo do Roteamento de Canal

Já que um canal tem altura típica de algumas unidades até algumas poucas dezenas de trilhas, e comprimento de algumas dezenas até milhares de trilhas, ele representa um problema de roteamento horizontal. Seja dado um conjunto de n redes. Se, em média, estas redes abrangem a metade do comprimento do canal, temos que a necessidade de roteamento horizontal é de $n * l / 2$. Assumindo que a metade das redes tem terminais em apenas um lado, a necessidade de roteamento vertical é de $n * h / 2$. Observe que se h é muito menor que l, então a necessidade de roteamento vertical é muito menor do que a horizontal, embora a área disponível para conexões verticais, $h * l$, seja exatamente igual àquela disponível para conexões horizontais quando associamos uma camada à direção horizontal e outra à vertical, como normalmente é feito. Observe que esta análise não é perfeita, pois avaliando a necessidade horizontal e vertical da rede da Fig. 2.4, percebe-se que um roteamento mínimo exige mais do que a mínima necessidade horizontal somada à mínima vertical.

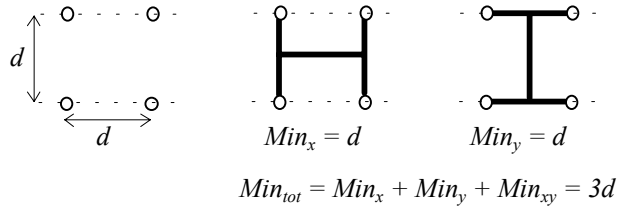


FIGURA 2.4 – Necessidades de roteamento em x, y, e mínimo global.

Esta diferença é justamente o espaço de trabalho que existe para aliviar o congestionamento em um sentido, através do uso de maiores conexões no outro. De fato, os algoritmos de roteamento de canal procuram analisar e controlar da melhor forma possível as conexões laterais, apenas modelando as restrições possíveis nas conexões verticais.

2.3.2 Definição de um Canal de Roteamento

A definição formal de um problema de canal pode ser dada em uma de duas formas: simbólica, ou espacial. Na forma simbólica, temos dois conjuntos de terminais, T e B, respectivamente do topo e da base do canal, e funções que têm estes conjuntos como domínio e suas posições x e seus números de rede como imagens. Na forma espacial, temos dois vetores V_t e V_b de comprimento l, onde cada posição tem um zero se nenhum terminal está presente, ou o número da rede que tem um terminal nesta posição.

O modelo de roteamento no canal também pode ser classificado conforme o número de camadas e o uso que fazemos delas. Existem algoritmos para roteamento de canais em duas e três camadas. A atribuição de uma direção obrigatória para cada camada acontece no modelo de camadas **reservadas**. Neste caso, conforme a direção que atribuímos para as camadas, contando a partir das inferiores, temos os modelos VH ou HV para canais com duas camadas, ou VHV e HVH para canais com três camadas. Os modelos com camadas não reservadas são superiores em termos de altura mínima e também em menor utilização de vias, mas requerem algoritmos mais complexos.

É freqüente a necessidade de se fazer conexões nas laterais do canal, mas poucas vezes isto é considerado nos algoritmos. Algumas outras extensões incluem canais com quebras, ou canais circulares (com 4 quebras), também chamados de anéis, canais com topo e base não retilíneas, entre outras. A altura do canal é fixa quando se trabalha com formas de implementação programáveis por algumas máscaras, onde o espaço existente para o roteamento já foi definido. Para projetos *full-custom*, no entanto, esta altura é variável, e então assume-se que sempre será possível resolver o problema, sendo desejada a solução que apresente a menor altura em trilhas.

2.3.3 Grafos associados ao roteamento de canal

Em função da dificuldade de roteamento horizontal, o roteamento do canal é basicamente o problema de assinalar segmentos horizontais de redes a determinadas trilhas do canal. Segmentos verticais são usados para conectar segmentos horizontais da mesma rede em diferentes trilhas, ou com os terminais. Há dois tipos de restrições que devem ser satisfeitas, as restrições horizontais e as verticais.

Chama-se HCG o grafo de restrições horizontais, que é na verdade um grafo de intervalos dos segmentos de rede. Este grafo é importantíssimo para a análise e roteamento do canal. Em um modelo de duas camadas, segmentos diferentes não podem ocupar a mesma trilha, e então, o máximo clique em HCG define o limite inferior em número de trilhas para o canal.

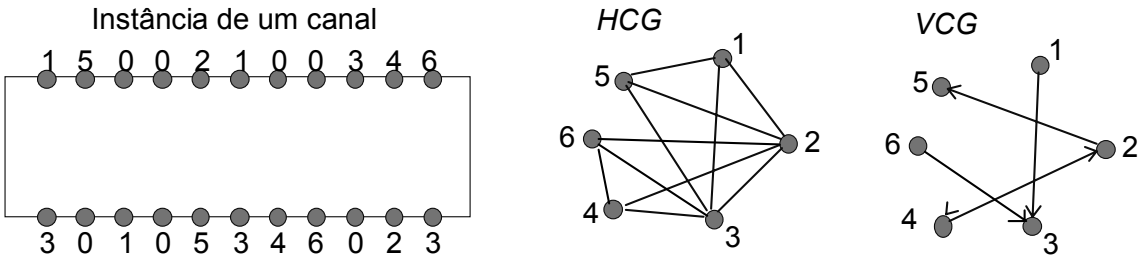


FIGURA 2.5 – Grafos de restrições associados a um canal.

O grafo de restrições verticais VCG possui uma aresta entre dois segmentos se eles possuem conexão com terminal na mesma coluna, ou seja, se começam ou terminam no mesmo ponto. VCG é um grafo dirigido, onde o sentido das arestas representa a ordem em que as conexões devem estar para que os segmentos verticais não se interseccionem. Se houver ciclos em VCG então alguns segmentos devem ser divididos através de *doglegs*, para que uma ordenação seja possível. As restrições verticais também impõem limitação na altura do canal. Se não forem usados *doglegs*, então o maior caminho em VCG limita a altura mínima do canal.

2.3.4 Algoritmos para roteamento de canal

Sendo um dos problemas mais freqüentes em leiaute *VLSI*, existem dezenas de algoritmos diferentes empregados na sua solução. Os algoritmos básicos mais importantes são: *Left-Edge*, *Dogleg*, YK, *Greedy*, *YACR2*, Hierárquico.

O algoritmo *Left-Edge* (*LEA*) [HAS 71] roda em tempo linear em relação ao tamanho do canal, e gera sempre a solução ótima em alocação de trilhas, em ausência de restrições verticais. O algoritmo consiste em ordenar os segmentos horizontais necessários para completar as conexões pela sua coordenada *x* esquerda (Fig. 2.6a). Para cada linha, são selecionados os segmentos nesta ordem, a partir da esquerda, tomando-se sempre o próximo que começa depois do final do anterior (Fig. 2.6b). O processo se repete linha a linha até que todos os segmentos tenham sido encaixados.

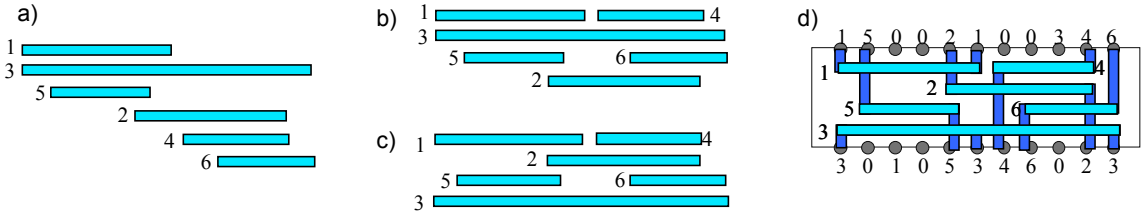


FIGURA 2.6 – Roteamento de canal pelo algoritmo *Left-Edge*.

Caso hajam restrições verticais, pode-se combinar os nodos do *VCG* para as conexões assinaladas às mesmas trilhas. Se o grafo resultante não contiver nenhum ciclo, então é possível uma ordenação das trilhas para produzir um roteamento válido

(Fig. 2.6c). Se, no entanto, *VCG* combinado tiver um ou mais ciclos, não é possível realizar o roteamento sem a inserção de *doglegs*, que são pequenos desvios, ou quebras em uma conexão. Os algoritmos conhecidos como *Dogleg* [DEU 76] consistem em considerar a introdução de *doglegs* nas conexões, removendo a restrição de que cada segmento de rede esteja em apenas uma trilha por toda a sua extensão. Isto quebra o *VCG*, de forma que os nodos sobre os quais se aplicam as arestas (restrições) são replicados, e os ciclos podem ser removidos. Foi demonstrado que encontrar o melhor número e posição dos *doglegs* é NP-completo ([PRE 88] pág. 174).

O algoritmo *Greedy* [RIV 82] é um algoritmo heurístico que também roda em tempo linear, apresentando um bom desempenho. Ele processa o roteamento da esquerda para a direita, conectando os terminais às trilhas, e estas entre si, em cinco passos, de acordo com a prioridade natural destas conexões (Fig. 2.7). Ele busca conectar redes presentes em mais de uma trilha o tanto quanto possível, já que as conexões horizontais são limitantes em um canal. Como outras heurísticas, ele é sensível a alguns parâmetros de que necessita em sua implementação, e não garante completar o roteamento antes de chegar ao outro extremo do canal.

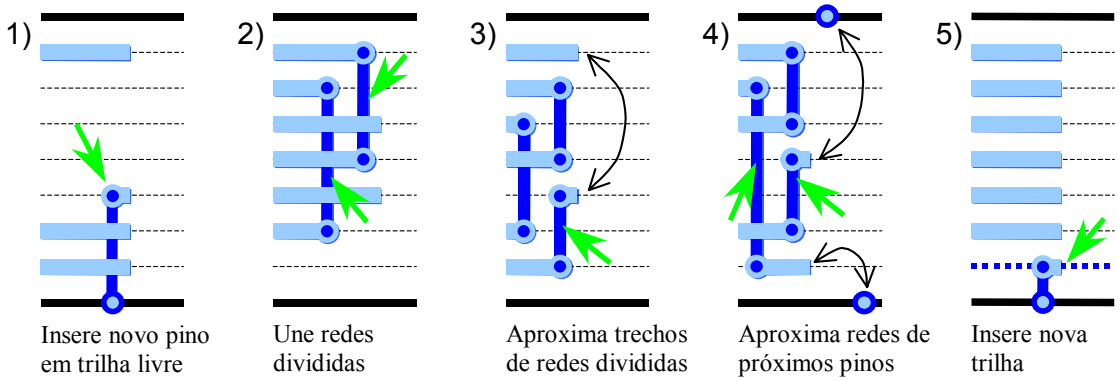


FIGURA 2.7 – Funcionamento do algoritmo *Greedy*.

O algoritmo *YACR2* e Y-K (Yoshimura e Kuh, 1982, [SHE 93] pág. 286) também se baseiam no *LEA*, e fazem o roteamento por este método usando também avaliações dos grafos de restrições e inserção de *doglegs* e desvios. Um quadro comparativo destes algoritmos básicos pode ser observado em [PRE 88] pág. 180, e [SHE 93] pág. 301.

Há uma série de algoritmos para o roteamento de canais com três camadas metálicas, em sua maioria, adaptações dos algoritmos anteriores. O canal pode ser construído nos modelos VHV ou HVH. No primeiro, temos duas camadas para roteamento vertical, o que elimina os conflitos verticais existentes. Assim, o roteamento sempre será possível com altura mínima correspondente ao *clique* de *HCG*, com um algoritmo *Left-Edge*. Entretanto, como já foi visto o canal é um problema de roteamento horizontal, e no modelo HVH pode-se encontrar um roteamento com altura mínima dada pela metade do *clique* de *HCG*, ou o máximo caminho em *VCG*, mas nem sempre com a garantia de encontrar esta solução.

O algoritmo Glitter de Chen e Kuh (1986) ([SHE 93] pág. 291) permite o roteamento de canais não baseados em grade, com conexões de larguras variáveis. Este algoritmo foi estendido para o roteamento em três camadas por Chen em 1986 ([SHE 95] pág. 88). Existem outros algoritmos de roteamento de canal específicos para canais com bordas não retilíneas, ou para conexões em 45^{o}, para duas e três camadas de

roteamento.

2.3.5 Caixa de conexões, ou *switch box*

Uma caixa de conexões, ou *switch box*, é um retângulo com largura e altura aproximadamente iguais, com terminais nos quatro lados. Este tipo de problema ocorre para conectar canais de roteamento perpendiculares, ou uma região entre quatro módulos, entre outras situações. Embora se possa pensar em um *switch box* como uma generalização de um canal, ele não apresenta a mesma característica horizontal, e também não possui altura variável. Enquanto se procura minimizar a altura de um canal, para um *switch box* procuramos avaliar sua roteabilidade, e encontrar apenas uma solução possível. Assim, ele é melhor caracterizado por um grafo circular.

Os problemas de *switch box*, apesar de mais complexos do que o roteamento de um canal convencional, são normalmente solucionados por algoritmos semelhantes, adaptados. Encontra-se na literatura o uso do algoritmos hierárquico [BUR 83b], uma variação do algoritmo Greedy [HAM 84] para o sistema se síntese de leiaute *Magic*, sistemas baseados em inteligência artificial, como o algoritmo WEAVER [JOO 86], dentre outros.

2.4 Roteamento Planar

Roteamento planar é aquele possível de ser realizado em apenas uma camada, sem cruzamento de conexões. Uma série de modelos e algoritmos surgiram para este tipo de problema, principalmente quando se dispõe de uma única camada de roteamento. Mas mesmo nas situações onde várias camadas estão disponíveis pode-se dividir o problema em um conjunto de problemas em apenas uma ou duas camadas. Uma característica importante de todo roteamento planar é que ele não possui vias, pois não precisa de troca de camadas, exceto nos seus terminais. Um grande esforço é empregado na redução do número de vias no leiaute de um circuito pois possuem várias desvantagens:

- menor confiabilidade no processo de fabricação;
- maior atraso dos sinais pelo aumento das restências;
- menor imunidade a ruído pela introdução de resistências em série;
- maior área devido à margem dos metais sobre os furos;

Assim, uma possível abordagem para roteamento em múltiplas camadas é selecionar os subconjuntos planares de conexões para serem realizados cada um em uma única camada, reduzindo o número de furos. De fato, esta abordagem tem sido utilizada para tratar roteamentos com várias camadas em *PCB*s [DOR 81] [TSU 81a] [TSU 81b], leiautes *VLSI*, e *MCM*s [SHE 95] págs. 323-329. Alguns problemas particulares de roteamento planar são notáveis por sua importância. A Fig. 2.8 apresenta, para referência, exemplos de roteamento planar de uma caixa, roteamento em rio e roteamento de pontos em linha.

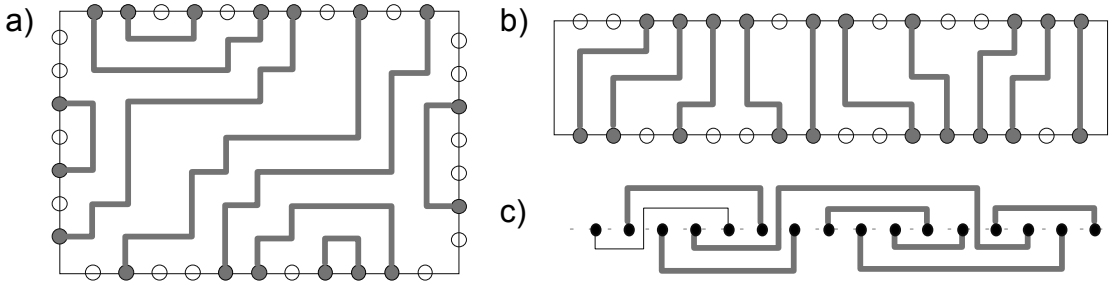


FIGURA 2.8 – Exemplos de roteamento planar: a) caixa; b) rio; c) *SRRP*.

2.5 Roteamento global

O roteamento global pode ser visto claramente como um processo de otimização combinatória, e as mais diversas técnicas já foram aplicadas. Alguns trabalhos antigos importantes são apresentados em [TIN 83] e [NAI 87], os quais podem ser tomados como ponto de partida para análise. A etapa de roteamento global tem três objetivos específicos:

- Distribuir as conexões no espaço, gerenciando o congestionamento;
- Definir a forma das redes maiores de múltiplos pinos;
- Definir exatamente os problemas de roteamento detalhado;

2.5.1 Roteamento Global com *General Cells*

Em uma metodologia *general cell*, a etapa de planejamento topológico deverá definir o tamanho, forma e posicionamento dos blocos, além de quais são as regiões usadas para roteamento. O tamanho destas regiões pode ou não já ter sido definido, mas de qualquer forma uma ordenação dos canais e *switch boxes* deve ser providenciada de forma a minimizar a perda de área após o roteamento detalhado. Com uma ordenação sem dependência cíclica, o roteamento detalhado de cada região pode ser feito a partir dos que já estão feitos, e definir as alturas mínimas dos canais.

O roteamento global deve modelar as áreas para roteamento, suas interseções, e encontrar os caminhos de todas as redes nestas áreas. A modelagem é feita com um grafo, em uma de duas formas possíveis. Pode-se modelar os canais como vértices e suas interseções como arestas, como na Fig. 2.9, ou as interseções como vértices e os canais como arestas. Em ambos os casos se tem um grafo regular ou irregular sobre o qual deve-se encontrar os caminhos para as redes.

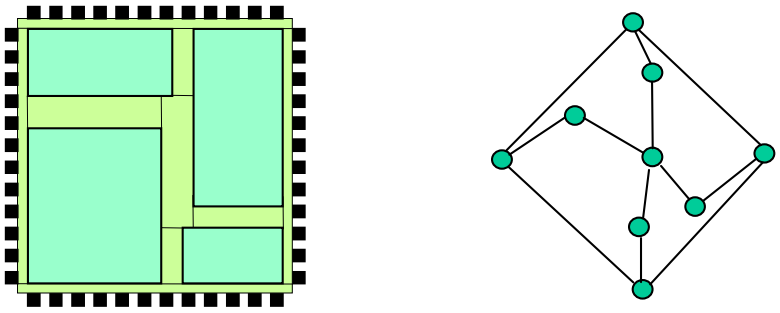


FIGURA 2.9 – Roteamento global de uma rede em circuitos *General Cell.*

2.5.2 Roteamento Global com *Standard Cells*

Já em circuitos baseados em bandas, normalmente não há canais verticais, e portanto não é possível construir um grafo desta forma. As conexões verticais devem ser feitas em uma de três formas: usando pontos específicos de passagem; inserindo espaços adicionais para conexões verticais; ou então livremente sobre as células, se houver espaço para tal. Neste caso, o roteamento global pode assemelhar-se a roteamento de área. Nas duas primeiras formas de implementar conexões verticais, mais freqüentes, a modelagem mais natural é formar grafos específicos de cada rede, a partir das posições de seus terminais. A Fig. 2.10 mostra os terminais de uma rede em um circuito *standard cell*, o grafo correspondente, e duas árvores deste mesmo grafo que podem ser selecionadas para conectar esta rede.

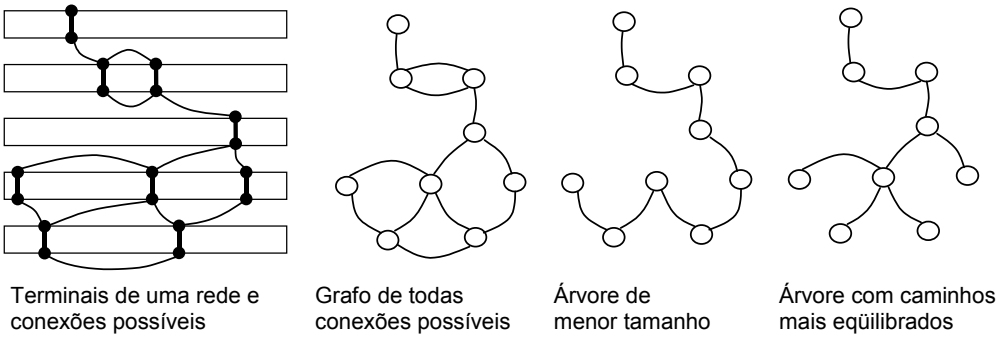


FIGURA 2.10 – Roteamento global de uma rede em circuitos *Standard Cell.*

A seleção de árvores melhores para cada rede deve ser provida para melhorar o desempenho do circuito. Isto pode ser feito com um algoritmo que remove as arestas de maior custo que não desconectam o grafo. Ao mesmo tempo, o congestionamento dos canais deve ser modelado e evitadas as conexões nas regiões de maior densidade.

2.5.3 Roteamento Global de Área

Quando o circuito não apresenta estrutura e espaços dedicados para roteamento, deve-se efetuar o roteamento global através de uma divisão arbitrária da área em regiões globais. Como mostra a Fig. 2.11, se obtém um grafo regular no qual deve ser executado o processo. Este será uma grade semelhante às usadas em roteamento detalhado. A principal diferença está no fato de que várias conexões podem passar pela mesma posição, e isto se reflete no modo de especificar o custo, que não mais representa somente livre ou bloqueado.

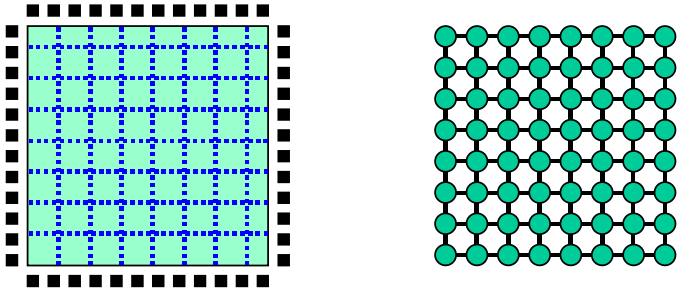


FIGURA 2.11 – Grafo (grade) para roteamento global de área

2.5.4 Formação de árvores para as redes

A Fig. 2.10 já apresentou um exemplo de árvores diferentes que podem ser formadas com o mesmo conjunto de pinos disponíveis para uma rede. Outros modelos que não o de *standard cells* também precisam considerar a forma destas árvores, que tem relação direta com o comprimento de conexões e atraso da rede.

Para as redes que necessitam a seleção ou inclusão de estruturas de passagem, o grafo precisa modelar estas estruturas em adição aos terminais da própria rede. Mesmo tendo grafos ainda particulares para cada rede, se tem um problema diferenciado de encontrar a árvore mínima. Este problema consiste em encontrar a árvore mínima que cubra determinados nodos (aqueles que representam os terminais), mas não todos os nodos do grafo (Fig. 2.12). Estas árvores são conhecidas como árvores de Steiner. Há diversos algoritmos usados para encontrar árvores de diversas formas e segundo diferentes critérios (ver em [KAH 95]).

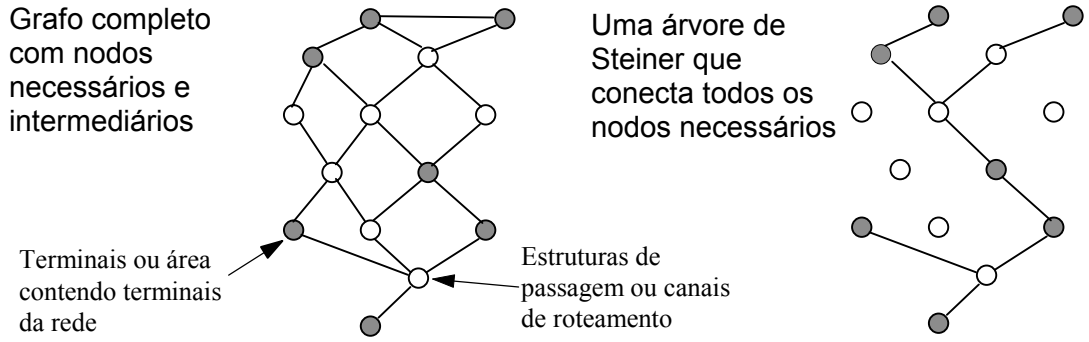


FIGURA 2.12 – O problema de árvores de Steiner.

3 Sistemas de Roteamento

É evidente que um simples algoritmo não é suficiente para implementar um sistema de roteamento de circuitos integrados complexos. Um sistema real envolve um grande conjunto de problemas, algoritmos para solucionar cada um desses, e define o modo pelo qual se relacionam. A seqüência de etapas (operações sobre dados) que são realizadas até se atingir o objetivo final é normalmente conhecida como metodologia, ou seja, conjunto de métodos e sua ordem de execução, ou também como fluxo de projeto.

Os dados que se submetem a estas operações também não são simplistas como em algoritmos didáticos em geral. Os problemas reais nem sempre são adequadamente modelados, não somente por deficiências em detalhes sutis, mas freqüentemente em seus objetivos e características estruturais. Além disto, o universo é rico em situações particulares e exceções, o que complica a implementação. Ou as linhas de código se multiplicam para contemplar tais casos, ou um nível maior de abstração deve ser elaborado para tratar homogeneamente todas as situações distintas. Assim, as informações de roteamento demandam uma especificação criteriosa, implementação flexível, e por vezes uma infra-estrutura de *software* adequada, como um Banco de Dados, de suporte.

Finalmente, já que a etapa de roteamento é uma das mais próximas à implementação física do circuito, a quantidade de detalhes é grande, e a etapa de geração de leiaute deve preocupar-se com minúcias até então abstraídas pelo processo de síntese.

Este Capítulo apresenta sucintamente as características de sistemas reais de roteamento quanto aos aspectos supra mencionados. Estes aspectos são ilustrados com o exemplo de dois sistemas de roteamento implementados pelo autor, capazes de produzir o leiaute final para fabricação a partir da especificação estrutural do circuito e de informações de posicionamento relativo ou absoluto.

3.1 Hierarquia de Problemas

Há duas razões principais para que um problema a ser solucionado tenha que ser dividido em vários subproblemas. A primeira é ser um problema complexo, de difícil modelagem. A segunda é, mesmo para um problema mais simples, ter um número de entradas muito grande. Problemas genéricos de roteamento são em geral complexos, e pelas suas características, pode-se identificar claramente algumas partes que são independentes. Deste modo se tem pontos específicos para a divisão necessária. Para o roteamento de um CI, esta divisão envolve tanto subproblemas puramente de roteamento como outros de posicionamento, lógica, etc... Em outras palavras, a decomposição dos problemas e a composição das soluções são necessárias e implícitas nas metodologias ou estilos de leiaute usados. Por exemplo, o roteamento de um circuito com standard cells é decomposto como mostra a Fig. 3.1. Embora se possa ter problemas completamente independentes, isto nem sempre ocorre. Assim, nesta hierarquia de problemas, existem duas possibilidades de dependência, que são a dependência cíclica, e a criação de problemas solúveis.

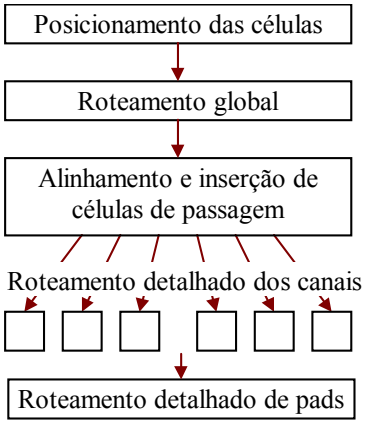


FIGURA 3.1 - Exemplo de decomposição de problemas.

3.1.1 Dependência Cíclica

A dependência cíclica ocorre quando a definição de um problema A depende da solução de um problema B, cuja definição depende da própria solução de A. Esta dependência é normalmente apenas parcial, sendo que ambos os problemas possuem dependência de outros dados que podem ser fixos ou oriundos de outros problemas, como mostra a Fig. 3.2. Quando eles são fixos em todos os casos, talvez eles possam ser explorados incorporando seu conhecimento nos algoritmos usados para resolver os problemas.

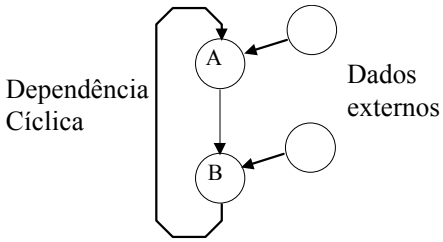


FIGURA 3.2 – Dependência cíclica entre dois problemas A e B.

É fácil de encontrar exemplos deste tipo de dependência. O algoritmo de troca de terminais eqüivalentes por banda apresentado em [JOH 97b], págs. 39-43, depende completamente da informação de extremidade destes terminais em relação às conexões nos canais adjacentes. Ora, esta informação somente pode ser exata se o roteamento global já tiver definido quais são os segmentos de conexão usados para cada rede. Para canais com capacidades fixas, como *gate arrays*, este mesmo roteamento global necessita avaliar exatamente a capacidade dos canais, e, portanto, precisa saber em que posição os terminais de cada rede estão. Esta informação é obviamente alterada pelo algoritmo de troca de terminais.

Existe duas considerações para a redução deste tipo de conflito sem o uso de iterações, e elas devem ser usadas em conjunto. A primeira é comparar a quantidade de informações dependentes e independentes de cada um dos problemas, e fazer em primeiro lugar aquele problema que apresenta menor sensibilidade à dependência cíclica. A segunda é usar algum método de aproximação das informações que deveriam ser providenciadas pelo problema que será feito em segundo lugar. Nem sempre é

possível que os erros desta aproximação mantenham a primeira solução válida. Se os erros inviabilizam a solução, a iteração é necessária. Quando, no entanto, os erros são aceitáveis, apenas nos afastamos da solução ótima, mas permanecemos com um processo que termina em um passo. A aproximação será geralmente uma estimativa de alguma solução para o segundo problema, e podemos acrescentar um refinamento do primeiro problema ainda em uma passagem, como mostra a Fig. 3.3.

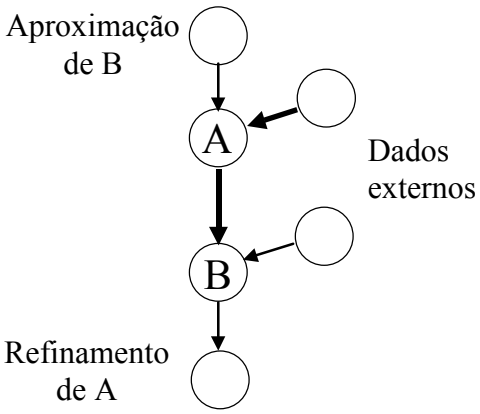


FIGURA 3.3 – Quebra de dependências cíclicas em uma passagem.

3.1.2 Criação de Problemas Solúveis

O segundo tipo de dependências é aquele que afeta a garantia de se encontrar uma solução para um dos problemas, podendo ocorrer mesmo em dependências não cíclicas. Seja dois problemas A e B, B dependendo de A, sendo que B tem solução para um conjunto do universo de instâncias possíveis, mas não para todos elas, como mostra a Fig. 3.4. O problema é: como fazer com que a solução de A sempre gere dados que se encontrem dentro do conjunto solúvel de B? Esta relação ocorre com freqüência entre problemas de leiaute de circuitos. As metodologias que mais tiveram sucesso foram justamente aquelas que venceram este tipo de dificuldade, algumas pela sua inexistência.

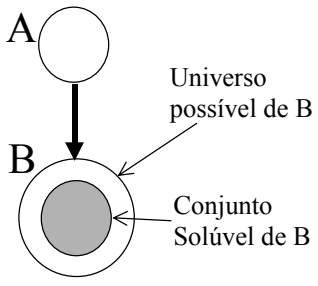


FIGURA 3.4 – Universo de problemas com sub-conjunto solúvel.

Para um dado problema que não apresenta solução para todas as entradas, somente o fato de analisar quando uma instância sua tem ou não solução é geralmente mais complexo do que realizar esta solução. Isto mostra a dificuldade de identificar cada elemento do conjunto solúvel de B. O problema de gerar soluções possíveis é o problema de definir completamente o conjunto solúvel de B.

Para solucionar estas situações pode-se trabalhar tanto em B quanto em A. Se B é

um problema muito complexo, cuja solução não é exata, pode-se procurar melhores algoritmos para solucioná-lo, de forma que o conjunto solúvel de B seja ampliado. Esta é a principal ocupação da pesquisa por algoritmos de otimização combinatória em geral, e mesmo de algoritmos heurísticos dedicados a problemas particulares.

Pode-se trabalhar no problema A, ou nos dados que dele provêm, de modo a limitar o conjunto de suas soluções dentro do universo possível. Limitar este conjunto pode ser bastante fácil, mas nem sempre é fácil fazer com que ele case com o conjunto solúvel de B. Mesmo quando se consegue caracterizar bem este conjunto, como se mostrou ser difícil, esta caracterização pode não ser fácil de usar dentro do problema A, porque isto praticamente significa solucionar A e B juntos. Ora, foi visto que a divisão em subproblemas separados é necessária, e, portanto, solucioná-los juntos pode ser impraticável. Apesar destas dificuldades, que devem ser avaliadas, há muitas situações onde a criação de problemas sempre solúveis pode ser encontrada. A Fig. 3.5 ilustra o aumento do conjunto solúvel, o uso da caracterização exata, a limitação a um subconjunto menor, e a caracterização de um conjunto inexato, mas cuja probabilidade de gerar problemas insolúveis é aceitável. Neste último caso, pode-se gerar alguma perturbação em A para gerar outra solução, ou usar algum método dedicado para remover os conflitos que tornam B insolúvel, se estes são casos restritos locais.

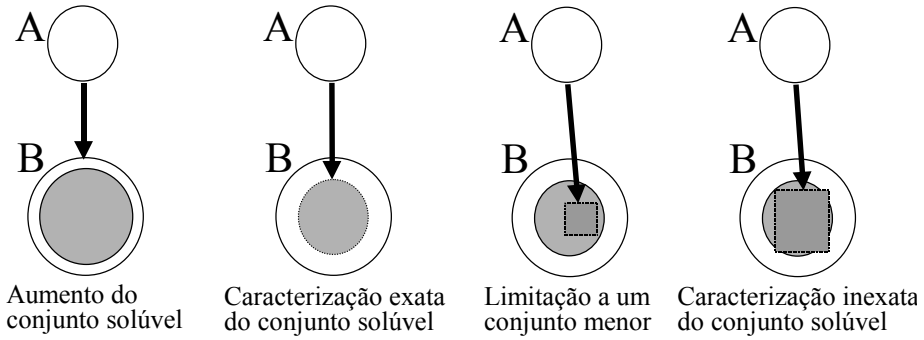


FIGURA 3.5 – Abordagens para criação de problemas solúveis.

3.2 Fluxo de Síntese do Sistema GAROTA

O sistema GAROTA faz parte do ambiente de projeto ÁGATA [CAR 96] [CAR 97] e consiste em uma ferramenta para o roteamento de circuitos digitais utilizando um *gate array* cuja arquitetura foi proposta no Brasil. O sistema foi inteiramente desenvolvido pelo autor para esta arquitetura específica [JOH 97b], e, portanto fornece um bom exemplo de como a especificação e solução de um conjunto de problemas concorrem para a realização de um processo de síntese claro e eficiente. No anexo 1 encontra-se artigo publicado contendo uma síntese de como o projeto funcional (não são considerados aspectos de implementação do *software*) foi elaborado, e resultados recentes podem ser observados em [JOH 98] e adiante. A leitura do anexo é recomendada para melhor compreensão do texto.

3.2.1 Arquitetura e Abstração da Matriz

A característica que mais se distingue na arquitetura de matriz proposta é a personalização do circuito usando apenas a última camada metálica, mas utilizando trechos de conexões (denominados *underpasses*) já fisicamente presentes nas matrizes.

Há alguns trabalhos na literatura contemplando roteamento em modelos semelhantes. Os elementos fixos podem ser considerados como exceções em algoritmos de roteamento genéricos, mas uma análise mais objetiva revela que o número de restrições impostas é tal que permite serem exploradas com vantagem para especificação de problemas mais fáceis e com soluções eficientes.

As Figs. A5.1 e A5.2 no Anexo 5 mostram a matriz GA2500 inteira, e uma parte do canto inferior esquerdo, onde pode-se observar a estrutura de bandas do circuito, intercaladas com os canais de roteamento já preenchidos com *underpasses* verticais, e como esta estrutura do centro do circuito faz interface com os canais dedicados às conexões com *pads*. A análise dos espaços e elementos a conectar no centro do circuito sugere a definição dos seguintes problemas, localizados na Fig. 3.7:

- roteamento de canal - conectar *underpasses* entre si;
- roteamento sobre alimentação (OTPA) - ligar *underpasses* às células;
- roteamento sobre as células (OTCA) - levar os pinos das células até suas bordas;

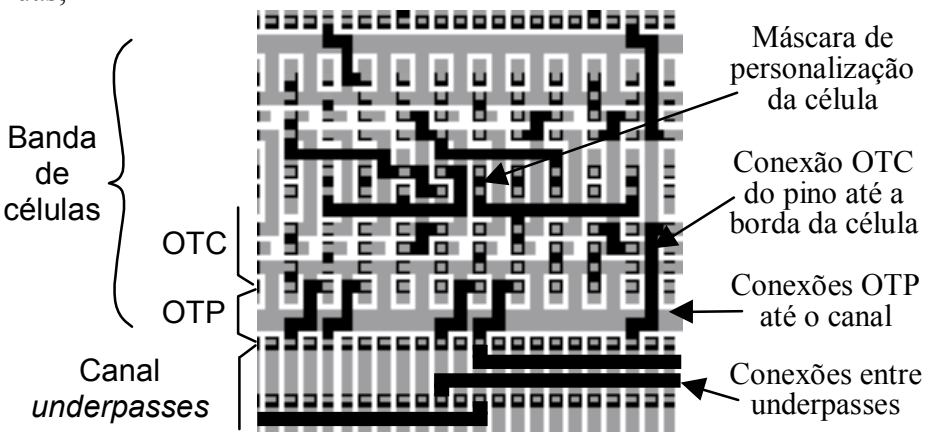


FIGURA 3.7 – Caracterização dos problemas de roteamento detalhado.

De fato, estes são problemas bem definidos, e alguns de fácil solução. Do ponto de vista do circuito completo, foi elaborado um fluxo geral de funcionamento que inclui estes e outros problemas adicionais. Este fluxo pode ser visto no anexo 1, Fig. 5, e corresponde à estrutura física apresentada na Fig. 3.8. Sua execução toma como entrada o circuito posicionado e gera como resultado o conjunto de conexões necessárias para o completo roteamento do circuito. Este exemplo é particularmente interessante pela quantidade e variedade de tarefas especificadas, capaz de ilustrar com riqueza os problemas encontrados em qualquer metodologia.

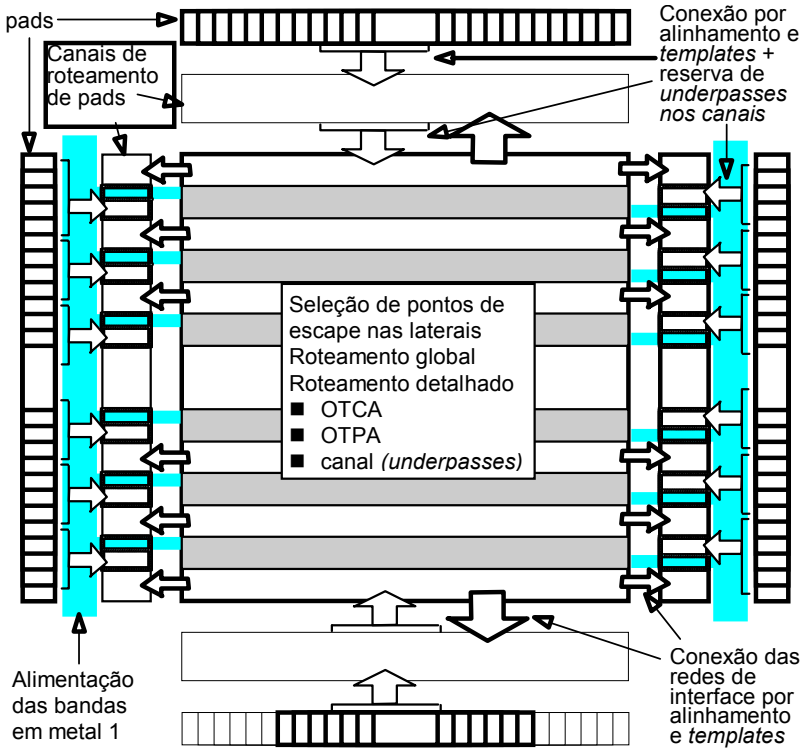


FIGURA 3.8 – Modelo abstrato de roteamento no sistema GAROTA.

3.2.2 Algoritmos para Criação e Solução dos Problemas

Pode-se facilmente identificar os tipos de dependências que ocorrem e como elas foram resolvidas. O nível mais baixo apresenta pouca complexidade. O roteamento ótimo de canal sempre pode ser obtido em tempo linear pelo algoritmo *Left-Edge*, desde que os *underpasses* já tenham sido assinalados. Já o problema de OTPA é um problema difícil de se resolver. Nem toda a instância criada deste problema tem solução, o teste de existência de solução é de grande complexidade, e encontrar uma solução quando há também o é.

Inicialmente foi utilizado um algoritmo simples de varredura para o assinalamento OTPA [JOH 97b], fazendo com que o conjunto de problemas solúveis fosse menor do que os possíveis de terem solução por um algoritmo ótimo. O algoritmo consistia basicamente em assinalar cada pino ao *underpass* anterior mais próximo com a mesma rede, ou o penúltimo livre, ou o próximo livre em último caso. Para evitar que o algoritmo recebesse problemas que não pudesse resolver, o problema anterior (OTCA) foi ajustado de forma que não produzisse seqüências de muitos terminais próximos. O algoritmo de OTCA usado também é bastante simplista, e consiste em testar o assinalamento para a primeira posição possível de uma lista cadastrada na descrição de cada célula primitiva. O ajuste foi propriamente feito nestas listas de posições possíveis para os pinos de cada célula da biblioteca, e corresponde à limitação das instâncias de OTPA a um conjunto menor do que o conjunto solúvel pelo algoritmo de OTPA.

Para as últimas versões do sistema GAROTA, foi desenvolvido um trabalho específico de pesquisa para modelagem e solução do problema de OTPA. Um resumo deste trabalho aparece em [HEN 99], onde se vêem três principais objetivos: 1- implementar um algoritmo que sempre resolva o problema quando tem solução, e pare

rapidamente quando não tem; 2- fazer tal algoritmo encontrar as melhores soluções, minimizando o roteamento posterior entre *underpasses*; 3- definir formalmente quando uma instância de OTPA tem solução ou não, para evitar que sejam gerados problemas insolúveis.

O trabalho produziu um algoritmo bastante robusto, que testa uma árvore de assinalamentos possíveis, voltando atrás (retroativo) quando encontra um assinalamento impossível. Isto resolve os conflitos locais, maioria absoluta neste problema em conseqüência da localidade física dos terminais. Para evitar que o algoritmo se empenhasse em testar todas as soluções quando não existe nenhuma, foi limitado a desfazer no máximo 7 níveis de assinalamento, número este definido por experimentação. As melhores soluções são encontradas primeiro, utilizando um conjunto de prioridades nos assinalamentos (Fig. 3.9).

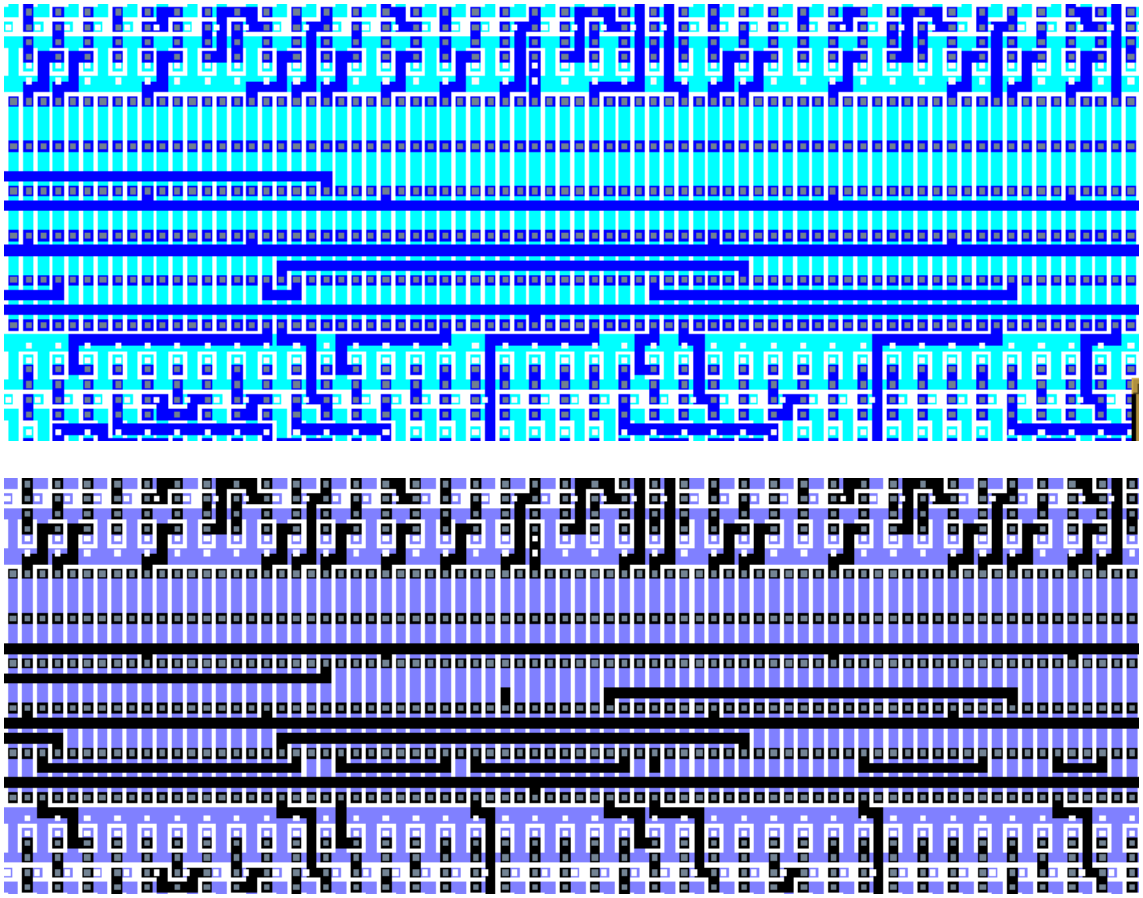


FIGURA 3.9 – OTPA inicial e otimizado no GAROTA.

Foi desenvolvido um autômato baseado em uma linguagem formal que descreve as instâncias solúveis, que em certas condições (para algumas matrizes) é capaz de dizer, à cada terminal que se vai inserir no circuito, se o problema de OTPA criado permanece solúvel. A ordem dos terminais, entretanto, ainda impede que ele seja usado na prática, pois os terminais de bandas diferentes do circuito são inseridos em momentos diferentes, e o canal que está entre elas precisaria ser composto da direita para a esquerda exclusivamente, no modelo desenvolvido. Mais detalhes podem ser vistos em [HEN 99].

3.2.3 Algoritmos para Roteamento Global

O roteamento completo do centro do circuito é composto pelo roteamento global e a seqüência descrita acima para cada canal. O roteamento global é quem deve especificar quais pinos de quais células devem ser conectados em cada canal. Conectar todos os pinos que pertencem a uma mesma rede em um canal é uma idéia primitiva, e de fato, foi o primeiro algoritmo implementado. Ressalva-se que somente devem ser conectados os pinos até então não considerados, já que cada pino participa dos dois canais adjacentes à banda, e conexões redundantes são inúteis. Isto não conduz a uma boa solução de roteamento global e nem garante encontrar uma viável. Uma alternativa melhor é apresentada no Capítulo 6, usando o algoritmo LEGAL. Por ora, apenas consideraremos este novo algoritmo como uma caixa preta. Ele elege algumas conexões que devem ser feitas dentre todas as possíveis para a mesma rede. Esta seleção leva em conta a capacidade dos canais, e, portanto, a informação exata de onde cada terminal está. É o que conduz à situação de dependência cíclica previamente citada na Subseção 3.1.1.

3.2.4 Reserva de *Underpasses*

Conectar o circuito central aos seus *pads* cria uma série de outros detalhes e problemas. Não há espaço entre o centro do circuito e os canais dedicados a conexões com *pads*, tampouco entre estes canais e os próprios *pads* (vide Fig. A5.2) Assim, as conexões de entrada e saída não podem sair do centro do circuito em uma posição destinada a um outro pino de *pad* que não deva ser conectado com ela. Por esta razão, os *underpasses* dos canais de *pads* e dos canais na borda do centro que já estão alinhados com pinos de *pads* devem ser reservados somente para estas conexões, como se já fossem pré-assinalados. Este tipo de problema de reserva é um meio termo entre problema metodológico de encontrar sempre uma solução e um tratamento de exceções. Ocorre sempre que uma determinada demanda necessita um recurso, mas com a exceção de que há somente um único recurso que ela pode usar (ou um conjunto). Este recurso então deve ser reservado, dentre os demais para seu uso exclusivo. Caso semelhante ocorre no sistema MARTE [JOH 95], para pinos de células, conforme explicado adiante.

3.2.5 Roteamento de Pads

A reserva de *underpasses* é um dos mecanismos necessários para que o roteamento de *pads* seja feito com sucesso. Há canais dedicados às conexões de entrada e saída do circuito ao redor de seu centro. O roteamento detalhado destes canais é semelhante aos do centro, exceto pela sua interface. Em vez de conectarem-se com células em uma banda, eles fazem interface de um lado com o centro do circuito e de outro com os *pads*. A interface com os *pads* foi inicialmente feita com uma biblioteca de trechos de conexão pré-elaborados, os quais podem ser observados na Fig. A5.3, de forma que todos os *underpasses* nos quais as conexões podem sair do centro do circuito permaneçam livres. Posteriormente esses padrões de conexão foram substituídos por um algoritmo de roteamento em rio sobre a alimentação vertical. A interface com o centro do circuito se dá por herança simples, mapeando posições diretamente, já que as reservas e padrões garantem que não haverá colisões.

Os canais horizontais de roteamento de *pads* têm 20 trilhas úteis cada, e há 28

pads que devem ser acessados através de cada um deles. Desta forma, apesar de se poder gerar conexões em posições tais que não haja solução, é muito improvável que isso aconteça. Já os canais horizontais de roteamento de *pads* têm somente 10 trilhas úteis cada. Nas primeiras versões, sem usar roteamento em rio sobre a alimentação vertical, a probabilidade de se ter uma instância de problema insolúvel já era considerável. Por esta razão, não somente a posição dos *pads* é considerada no posicionamento do circuito, como as posições por onde as redes de entrada e saída estarão disponíveis na interface do circuito são bem avaliadas. Esta etapa foi chamada de seleção de pontos de escape, e busca minimizar a distância entre estes pontos e as posições reais dos terminais de *pads*.

3.2.6 Desempenho do Roteamento

O fluxo direto do sistema GAROTA se baseia em algumas simplificações. Se o centro do circuito estiver muito congestionado, não é possível usar os canais de *pads* para oferecer mais espaço para as conexões. E se o canal de roteamento de *pads* não puder ser roteado, não há nenhuma forma pela qual o sistema automaticamente possa refazer todo o roteamento do centro de modo a atender a esta restrição.

A maior vantagem desta abordagem direta é a eficiência obtida pela modelagem de problemas restritos exatos e eliminação de longas iterações. O GAROTA pode rotear circuitos completos em menos de 10 segundos em uma máquina Intel® Pentium® de 133MHz, incluindo as etapas de posicionamento absoluto, geração de máscaras, e tempos de entrada e saída. A Tab. 3.1 mostra alguns circuitos roteados em diferentes versões do sistema GAROTA, na matriz de 20 bandas e 626 trilhas de largura. Observa-se que nas versões mais recentes os circuitos maiores podem ser roteados sem *overflows*, com exceção do ex40n, que é um circuito gerado aleatoriamente.

TABELA 3.1 – Alguns circuitos roteados pelo sistema GAROTA.

Nome do circuito	N° de células	N° de pads	N° de redes	tamanho	over-flow	opções	CPU time
Garota Versão Beta 2.1							
timer14	154	19	186	8 x 182	0	-F5	5s
m8255b	210	29	268	10 x 298	0	-F5	5s
powxor11	220	108	334	10 x 558	0	-F30 -i12	6s
copel	562	37	662	17x434	30	-F5	8s
copel	562	37	662	17x434	15	-F26 -i10	8s
ex40n	882	58	907	17x410	45	-F5	9s
Garota Versão Beta 2.3							
copel	562	37	662	17x434	25	-F5	8s
copel	562	37	662	17x434	6	-F-1	8s
copel	562	37	662	17x434	4	-F-1 -i15	8s
ex40n	882	58	907	17x410	21	-F-1	9s
Garota Versão Beta 2.5							
copel	562	37	662	17x434	0		8s
ex40n	882	58	907	17x410	10	-i16	9s

Para ter esse desempenho, paga-se o custo de desenvolvimento de um sistema

dedicado à arquitetura alvo. Mas mesmo considerando esse custo, é um desempenho notável comparado aos de outros tantos sistemas ou métodos que podem gastar até horas de CPU buscando encontrar uma solução viável para o mesmo circuito, conforme relatado por Simões, 1992, citado em [JOH 97b] e [JOH 97c].

A principal desvantagem do fluxo direto é a inexistência de operações retroativas quando ocorrem problemas não solúveis. Assim, é preciso manter os casos práticos distantes dessa possibilidade, o que restringe as soluções encontradas a soluções subótimas. Como dito, quando se busca melhorar o desempenho para acomodar circuitos maiores, algumas instâncias de projeto começam a falhar em vez de terem sucesso. No fluxo direto, estas falhas não são facilmente corrigidas. Um processo retroativo poderia ser usado eficientemente se o tempo adicional de processamento fosse proporcional ao número de dificuldades ou falhas encontradas. Mas na prática, processos retroativos não são capazes de identificar diretamente a causa dos conflitos, e gastam tempo desfazendo e fazendo decisões que não interferem no problema[3]. O sucesso do algoritmo de OTPA apresentado em [HEN 99] e também do algoritmo LEGAL está na exploração de localidade dos conflitos. Isto ajuda a garantir que operações de correção chegarão a afetar a ocorrência do problema eliminando-o sem longas iterações. Operações de "desfazer" e "refazer" também exigem mais cuidados de implementação, com dados melhor encapsulados, maiores observabilidade e controlabilidade.

3.3 Redes, Terminais e Assemelhados

O sistema MARTE é um ambiente para roteamento simbólico sobre células desenvolvido pelo autor [JOH 94] [JOH 95], e será usado para ilustrar alguns exemplos de modelos de redes e terminais, já que tem uma abordagem consideravelmente genérica destes elementos. O sistema utiliza algoritmos de pesquisa de caminhos (*maze router*) para fazer roteamento detalhado de área em presença de muitas restrições. Observações sobre o desempenho deste processo serão feitas oportunamente (Cap. 5), e por ora apenas considerados os modelos de interconexões. O anexo 3 contém um artigo publicado com um resumo das principais características e resultados obtidos.

A estrutura básica do circuito é representada pela lista de redes, ou *netlist*. Apesar do nome, é comum que esta informação seja descrita através da lista de instâncias de células, onde para cada instância, são ditas quais redes se conectam a cada um de seus terminais, já que estes são ordenados conforme o tipo de célula. Uma lista de redes é um pouco menos econômica, pois nela, cada rede será um conjunto de pares: instância e terminal. Uma mesma rede pode se conectar a dois terminais de uma mesma célula. Uma rede é ao mesmo tempo uma informação estrutural e física (geométrica), sem

[3] O autor implementou um pequeno programa de alocação de horários de disciplinas que implementa praticamente o mesmo algoritmo de assinalamento OTPA, e bem exemplifica esta dificuldade. O programa foi elaborado para testar todas possibilidades de horários, mas usando assinalamentos iniciais aleatórios em horários vagos (nas turmas e professores), até encontrar uma solução válida. A variação no horário das últimas disciplinas assinaladas não é capaz de resolver conflitos que impedem assinalar todas. Assim, quando uma série de tentativas não conduz a uma solução após um certo tempo, é preferível interromper o processo e recomeçá-lo à partir de uma nova semente de número aleatório, pelo que se encontra soluções válidas em tempo hábil. Este comportamento parece semelhante, mas não corresponde diretamente ao famoso conceito de mínimo local versus global, já que não estamos avaliando a qualidade da solução, nem comparando-a com outras.

maiores distinções. Seria ingenuidade pensar que uma rede é apenas um conjunto de pontos, já que há muitos outros detalhes estruturais e físicos.

3.3.1 Redes

A especificação de uma rede será em princípio um conjunto de terminais - também chamados pinos. Este conjunto deve estar completamente conectado ao final do roteamento. Para conectá-lo, deve haver alguma estrutura representando conexões realizadas, conforme será visto a seguir. Em etapas intermediárias de roteamento é necessário saber quais pinos estão ou não conectados, ou mesmo desfazer algumas conexões. Para isto, cria-se uma informação de conectividade, armazenada em uma de duas formas. A primeira consiste em manter um número associado a cada pino de forma que pinos com o mesmo número estejam conectados, e com números diferentes não. Atualizar esta informação exige tempo linear em relação ao número de pinos, mas este não é grande. Essa forma foi empregada no roteamento global do sistema GAROTA.

3.3.2 Grupos

A segunda forma é criar uma entidade grupo distinta, da qual os pinos são componentes. Assim, uma rede se torna um conjunto de grupos, e cada grupo um conjunto de pinos, e possivelmente também um conjunto de conexões (Fig. 3.11c). A atualização pode ser mais rápida, pois cada nova conexão implica em unir dois grupos formando somente um, e na implementação se resume em uma manipulação simples de ponteiros, e eliminação de um dos grupos. Deve-se apenas ter o cuidado de não manter ponteiros para grupos, já que são dinâmicos, mas apenas para pinos que são persistentes. No sistema MARTE, estes grupos denominavam-se **partes**, e se confundiam com a descrição dos terminais, pois representavam conjuntos de elementos que podiam ser pontos ou barras. Um terminal já era um conjunto destes elementos, e por isto era representado por um grupo. Esta implementação dificultava manter ponteiros consistentes para os terminais do circuito, pois deviam ser atualizados quando os grupos que representavam os terminais se uniam formando outros grupos (Fig. 3.11b).

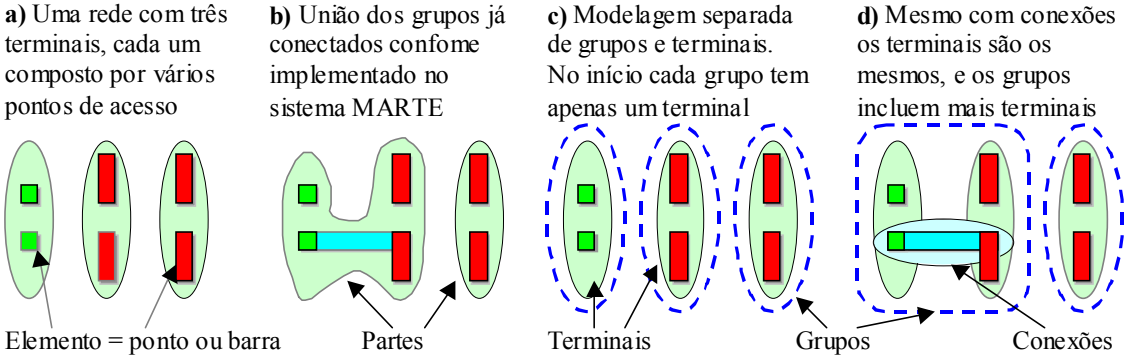


FIGURA 3.11 – Modelos de conectividade pela formação de grupos com e sem a preservação de terminais.

3.3.3 Posições de acesso

No sistema GAROTA, a etapa de assinalamento OTCA escolhe uma das possíveis

posições de acesso para cada terminal, e então ignora as demais. Entretanto, em muitas outras situações, se deseja tratar igualmente todos os pontos de acesso de cada terminal. A entidade que aparece com o nome de elemento (Fig. 3.11a) serve justamente para modelar diferentes posições de acesso de um mesmo terminal homogeneamente. Estes elementos podem ser pontos isolados, ou barras, com a única finalidade de compactar a descrição de posições consecutivas. É necessário observar também que embora disjuntos, os elementos de um mesmo terminal já estão garantidamente conectados pela estrutura interna da célula. Esta conexão, todavia, pode ser fraca ou forte. Será fraca quando a ligação interna usar materiais ou tamanhos que ofereçam pouca capacidade de corrente, e forte em caso contrário. As conexões fortes podem estar presentes em qualquer parte intermediária de uma rede, mas as fracas não. Deve-se controlar o acesso aos terminais com conexões fracas, para que fiquem pendurados na rede física, e não intercalados nela.

A camada de leiaute na qual um ponto de acesso existe também é importante. Genericamente se pode modelá-la como a terceira dimensão, tendo o cuidado de observar que os terminais podem estar acessíveis nas camadas adjacentes ao roteamento (como polissilício e área ativa), e não nas próprias camadas usadas para as conexões (metais). Neste caso, o mesmo problema de reservas descrito anteriormente aparece. O sistema MARTE implementa dois tipos de reservas: absolutas e alternativas. As absolutas são mais simples. Quando um terminal possui um único ponto de acesso em uma camada externa ao roteamento, a posição correspondente na camada de roteamento mais próxima deve estar reservada para acessá-lo, impedindo que outras conexões passem sobre ele e o deixem isolado. As reserva alternativas ocorrem quando há vários pontos de acesso, todos em camadas externas (inferiores no leiaute). Neste caso, um dos pontos deve ser reservado, mas não se sabe qual. A reserva é feita em todos os pontos possíveis, e sempre que uma conexão passa por um deles, a situação do terminal é reavaliada. Se ainda restam dois ou mais pontos de acesso, as reservas alternativas são mantidas. Se, entretanto, restar somente mais um ponto de acesso ela tornar-se-á uma reserva absoluta, implementada pela restrição de que nenhuma conexão mais passe pela posição reservada.

3.3.4 Terminais

Nem todos os terminais estão já conectados internamente. O sistema MARTE foi usado para roteamento de circuitos em uma abordagem onde células básicas como **nand** e **nor** já estavam pré-difundidas, mas sem nenhuma metalização [GUN 95]. Assim, a saída da célula era composta por dois pontos de acesso separados, como mostra a Fig. 3.12a, os quais são modelados como partes que são posteriormente unidas. A modelagem homogênea feita no sistema permitia simplesmente criar a rede através da inclusão da descrição de cada terminal seu, seja de apenas uma parte ou mais. No sistema GAROTA, a limitação de espaço para o projeto das bibliotecas de personalizações também faz com que as células tenham conexões faltantes. Neste caso, a ocorrência dos terminais separados implica em alteração da estrutura das redes. Cada terminal tem apenas um ponto de acesso, e a divisão dos problemas exige que se trate estas entidades com inteligência. Para isso, após o posicionamento absoluto, a informação estrutural do circuito é adaptada incluindo novos terminais, que na verdade são partes disjuntas dos mesmos presentes na estrutura original.

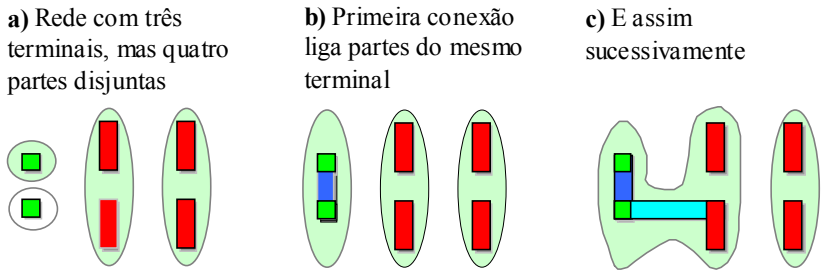


FIGURA 3.12 – Modelo de terminais separados com **partes** no MARTE.

Além dos terminais das células, podem ser necessários terminais intermediários, usados para fazer interface entre uma região e outra de roteamento, ou para direcionar redes globalmente, ou ainda para as redes de entrada e saída. Estes terminais podem ser criados e destruídos e podem ainda ser móveis. No sistema GAROTA, o roteamento de redes que cruzam uma banda é feito usando células de passagem, e, portanto, seus terminais se comportam como os normais. A implementação é mais simples do que se supõe, pois embora saibamos que estas células estarão intercaladas na rede fisicamente, estruturalmente elas são somente células adicionais nela penduradas. As redes de entrada e saída foram apenas marcadas como tal e tratadas como exceções nos roteamentos global e de canal. Redes de apenas um terminal são ignoradas, como saídas adicionais de **flip-flops** não usadas, mas quando são redes de entrada ou saída, devem ser assinaladas a *underpasses* do canal para serem conduzidas posteriormente aos *pads*. Outro tipo de terminal a ser ignorado é aquele que não participa de roteamento em um canal, já que foi conectado em um canal vizinho. No sistema MARTE o roteamento é completamente sobre as células, e somente são necessários terminais adicionais para redes de entrada e saída. Algumas outras abordagens, entretanto, exigem a modelagem explícita destes terminais e de seu estado indefinido de posição. Por exemplo, após o roteamento global encontrar uma determinada árvore, pode ser necessário ao roteamento detalhado identificar os pontos de interseção de trechos desta árvore como destinos em posições específicas, porém com certa flexibilidade.

3.3.5 Conexões

As conexões que serão realizadas durante o roteamento devem ser representadas de alguma forma, com diversos objetivos, entre eles: especificar o que deve ser feito, o caminho encontrado, ou o leiaute a ser gerado. O sistema MARTE trata os terminais e as conexões da rede como entidades idênticas (partes), pois tanto estas, quanto aqueles, são conjuntos já conectados de posições simbólicas. Já no sistema GAROTA, as conexões realizadas são entidades distintas, e armazenam a posição de origem e um caminho que pode fazer várias voltas, além de diversas outras informações, como rede a que pertencem, rótulo, entre outras. Para especificar quais conexões devem ser feitas, uma outra entidade, chamada **segmento**, é necessária, e não deve ser confundida com a informação de grupo. Como dito, nem todos os terminais da mesma rede devem ser conectados em um determinado canal. Um segmento é uma conexão desejada do ponto de vista global, e corresponde a uma rede do ponto de vista local, como ilustrado na Fig. 3.14. Na especificação de um canal, por exemplo, o número presente nos vetores topo e base será o número do segmento e não da rede.

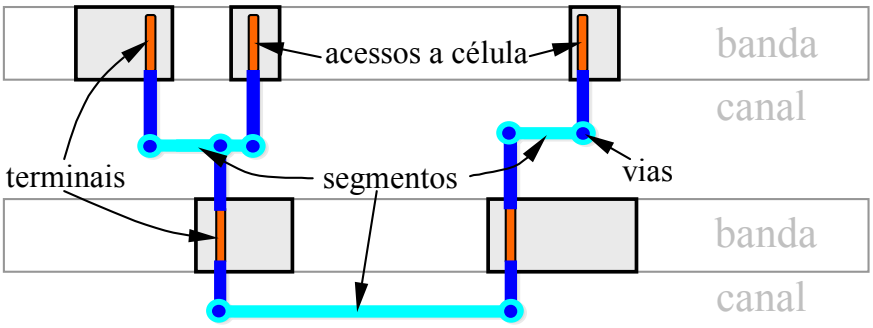


FIGURA 3.14 – Entidade “segmento” especifica conexões parciais de uma rede.

3.4 Otimizações e Exceções

As situações citadas acima já trazem uma série de exceções para a implementação de algoritmos inicialmente descritos de forma sucinta. Por exemplo, no sistema GAROTA, a presença das conexões de entrada e saída laterais nos canais altera o roteamento de *underpasses* pelo algoritmo *Left-Edge*, tanto pela origem das informações quanto pelo seu processamento e destino. Os segmentos de redes de entrada e saída começam ou terminam nos extremos do canal, em vez de limitarem-se aos primeiros e últimos terminais ligados ao segmento. Segmentos normais também nunca começam e terminam na mesma posição, mas os de entrada e saída sim, e a forma como estes dados são armazenados e processados é, portanto, diferente.

Além destas exceções, muitas outras surgem para otimização de situações peculiares ou que somente são observáveis após se ter obtido uma primeira solução. No mesmo problema de roteamento de canal, observa-se que conexões entre *underpasses* vizinhos não precisam ocupar as trilhas de roteamento, mas podem ser feitas diretamente nas linhas reservadas para as cabeças de contato, afetando novamente a implementação do algoritmo LEA. No assinalamento OTCA, terminais vizinhos que se devem conectar também poderiam ser assinalados às mesmas posições, e no assinalamento OTPA, conexões entre terminais vizinhos podem ser feitas diretamente na região OTP, sendo excluídos do canal se não têm outras conexões. Esse tipo de otimização não foi implementado no GAROTA devido às inúmeras alterações que seriam necessárias nos modelos e algoritmos de roteamento detalhado.

No sistema MARTE, após o retraço dos algoritmos de pesquisa, etapas específicas de otimização são executadas para melhorar as rotas feitas e facilitar as próximas. Normalmente, as camadas nas quais as conexões são feitas têm direção obrigatória, isto é, metal 1 apenas é usado em trechos horizontais e metal 2 em verticais. Posteriormente, se reduz o número de vias (para mudança de camada) quando a conexão apenas faz deslocamento entre duas trilhas vizinhas, já que nenhuma outra conexão pode passar entre essas trilhas. Também são executadas etapas que encolhem conexões que fazem voltas desnecessárias após a substituição das vias, ou empurram esses deslocamentos adiante até encostarem em outras conexões, como é possível observar na Fig. 3.15. Mesmo estando estas etapas de melhoramento fora do retraço, implicam no uso de uma estrutura específica para armazenar o trajeto inicial, processá-lo e depois traduzi-lo para a descrição normal de conexões com elementos.

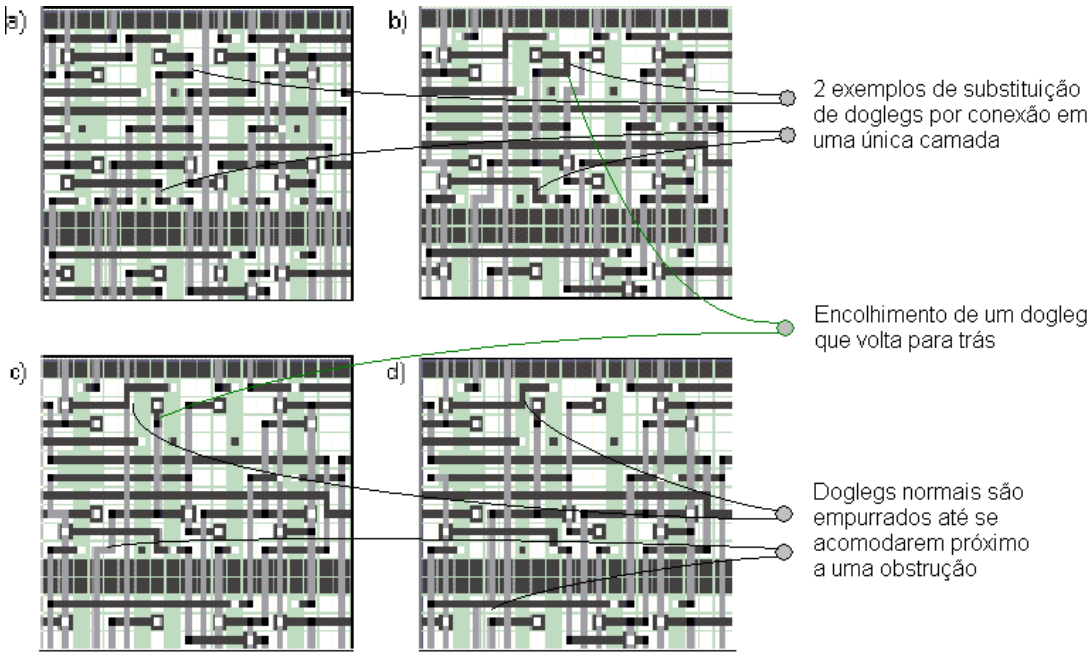


FIGURA 3.15 – Processamento de otimização de *doglegs* no sistema MARTE.

3.5 Dados e Descrições

Uma boa infra-estrutura de *software* é útil para qualquer sistema mais complexo, e um sistema de roteamento não é exceção. Boas práticas de modularidade e verificação de erros foram usadas com êxito em ambos os sistemas aqui apresentados, mas alguns detalhes de processamento e armazenamento de dados merecem destaque.

No sistema MARTE terminais e roteamento são modelados como entidades idênticas. Isto permite reprocessar o circuito em várias etapas sucessivamente, pois o formato de arquivo também é homogêneo. A estrutura do circuito é uma lista de redes, onde cada rede é uma lista de partes a conectar, e cada parte uma lista de elementos já conectados. Essa abordagem apresenta-se bastante flexível e simples de tratar. Algumas características, porém, estão ausentes. O sistema não preserva a informação dos terminais originais, não os separa das conexões, e não oferece nenhuma informação de continuidade ou localidade destas, o que dificulta algumas aplicações reais onde se deseja identificar os terminais do circuito, pré-calcular capacitâncias parasitas, etc...

Já no sistema GAROTA, pelo seu fluxo direto, não há necessidade de reprocessar ou completar qualquer parte do roteamento. Todo o processo deve funcionar sem correções do início ao fim. As conexões são então representadas simbolicamente (também em uma grade) de uma forma compacta e contínua. São entidades que têm um ponto de partida, e um caminho descrito por uma seqüência de caracteres. Os caracteres são "<^v>" para representar os sentidos oeste, norte, sul e leste, números que os antecedem para dizer qual o tamanho do movimento naquele sentido, ou ainda um ponto que faz com que a posição inicial para o próximo movimento seja a inicial do anterior, desconsiderando a atual. Esta é uma forma muito compacta, legível, e eficiente de armazenar conexões, as quais podem ter pequenas ramificações (ver Fig. 3.16).

```
/*---- UFRGS LOGO */

srot(38,0,801, "7v7>7^1<6v5<6^1<");
srot(38,10,801, "7v1>3^3>1^3<2^5>1^6<");
srot(38,20,800, "2v5>2^5<");
srot(38,19,801, "7v1>3^ 2>1v1>1v1>1v2>1^1<1^1<1^1<3> 4^7<");
srot(38,29,801, "7v7>4^3<1v2>2v5<5^6>1^7<");
srot(38,39,801, "4v6>2v6<1v7>4^6<2^6>1^7<");
srot(38,0,792, "46>");
```

FIGURA 3.16 – Um conjunto de conexões desenhando a sigla UFRGS.

Como mecanismo genérico de *software*, no sistema GAROTA um esquema arrojado de verificação de erros em **todas** as chamadas de subrotina foi empregado, evitando a propagação de qualquer situação anômala. Entretanto, o que mais se destaca é o uso de um formato padrão de descrição de dados, chamado de UFDS, cuja semântica é dada por funções que são cadastradas dinamicamente dentro das ferramentas. Os efeitos deste sistema, descrito em [JOH 96] são vários, mas principalmente a padronização dos níveis léxico e sintático, com o uso de apenas um leitor para todas as descrições, e a flexibilidade de se incluir novas funções, novos parâmetros ou usar diferentes versões sem necessidade de recompilação das partes já implementadas. Esse tipo de vantagem é percebido principalmente durante as fases de desenvolvimento, quando se fazem necessárias muitas alterações, mas também em situações diversas de configuração externa do sistema. Como exemplo, arquivos de configuração do GAROTA possuem em média mais de 500 linhas com algumas dezenas de funções de dados diferentes. Além disto, cerca de 10 arquivos com propósitos distintos são usados e lidos pelo mesmo leitor, como, por exemplo, biblioteca de células, configuração de matriz, posicionamento relativo, absoluto, informações de verificação, roteamento simbólico, etc...

3.6 Geração de Leiaute

Entende-se por geração de leiaute: a montagem das partes componentes para obter o leiaute completo do circuito, a tradução de roteamento simbólico para coordenadas reais (expansão) e algumas outras etapas de pós-processamento necessárias.

A montagem parte do posicionamento absoluto final do circuito e reúne as células de base, personalizações ou programações, roteamento, *pads*, cantos ou outros elementos e os coloca em posições relativas corretas em um único arquivo de leiaute (CIF para os sistemas aqui descritos). Estes elementos provêem de bibliotecas ou geradores, e devem ser corretamente identificados antes e após a inclusão com nomes e números coerentes. O sistema GAROTA usa uma hierarquia de blocos simbólicos que podem ser criados e instanciados uns dentro dos outros, com deslocamento espelhamento e rotação simbólicos. Esta estrutura é utilizada durante o roteamento para que os problemas sejam abstraídos de posição e orientação. Ela permite fácil tradução de coordenadas entre blocos e acompanhamento de conexões neles presentes.

A grade usada para o roteamento é quem define a relação entre a abstração simbólica e o leiaute final. No sistema MARTE, a grade acomoda também trilhas com cabeças de contato ou vias alternadas, como mostra a Fig. 3.17. Para usar trilhas

alternadas, justapostas e trilhas mais largas para alimentação, uma especificação compacta destes elementos é expandida para um vetor que contém as coordenadas de início e fim reais de cada uma das trilhas. A expansão é um mapeamento de coordenadas usando este vetor.

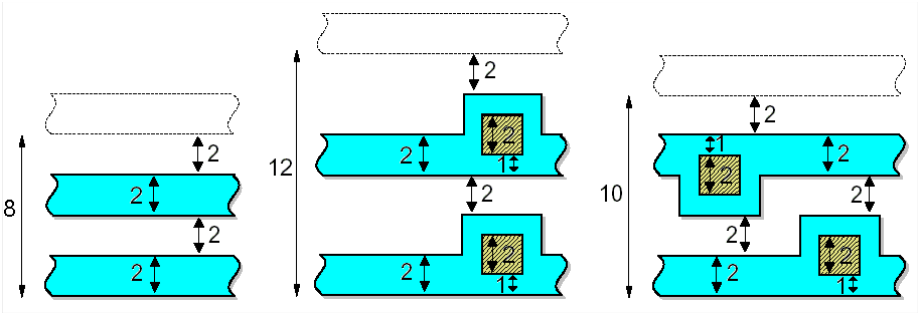


FIGURA 3.17 – Passos da grade: sem contatos, com contatos justapostos, e com contatos alternados, opção do sistema MARTE.

A grade de roteamento serve tanto para o alinhamento de partes componentes na montagem quanto para a geração do leiaute dos caminhos das conexões. Para ambos, a questão mais importante é a das posições de referência. A grade pode começar a partir de um dos cantos das trilhas (inferior direito, normalmente) ou estar centrada nas trilhas. A segunda é preferível, pois facilita as operações de rotação. O sistema GAROTA inclui as duas formas de referência, mas o roteamento é totalmente referenciado pelos cantos, por compatibilidade com a descrição da primeira matriz desenvolvida no CTI. Assim, um estado global de rotação é calculado para cada bloco (pois pode haver várias rotações potencialmente) de modo que o roteamento presente no bloco sofre deslocamentos correspondentes às larguras de trilha.

Algumas matrizes ÁGATA possuem passos horizontal e vertical de grade diferentes. O estado global de rotação também serve para encontrar em qual dos passos os elementos devem se encaixar. Isto significa que um segmento de conexão com x posições de largura apresentará comprimentos diferentes conforme colocado verticalmente ou horizontalmente no circuito. Por outro lado, os componentes de biblioteca têm sempre o mesmo tamanho, e as células de base e personalização de *pads* podem ser colocadas em qualquer um dos lados do circuito. Mesmo tendo sido elas projetadas de modo que suas dimensões sejam múltiplas dos dois passos, isto traz a complicação adicional de que suas medidas simbólicas precisam ser feitas nas duas direções. Ou seja, um determinado terminal de um *pad* está colocado em uma posição simbólica x_1 se o *pad* está na horizontal e x_2 se está na vertical. Ainda, na montagem do leiaute, já que os *pads* estão descritos todos em blocos horizontais, estas larguras aparecem intercambiadas, o que pode parecer incorreto se não for detalhadamente estudado.

Outras tarefas normalmente necessárias incluem opções de geração de leiaute por camadas, minimização das descrições, inclusão de estruturas de terminação, preenchimento, etc... Estes são apenas alguns exemplos dos detalhes que envolvem a preparação das máscaras finais de um circuito. No sistema GAROTA, que possui um total de 28 mil linhas de código, aproximadamente 5 mil são referentes à geração de leiaute final e 8 mil a etapas de preparação de máscaras, exibição, planificação e extração, para as quais o suporte das informações de roteamento é essencial.

O sistema ÁGATA com o roteamento GAROTA é a primeira ferramenta desenvolvida no Brasil capaz de gerar o leiaute completo para fabricação de um circuito sem qualquer intervenção manual. A Fig. 3.18 mostra uma fotografia de um circuito fabricado no CNM em Barcelona e encapsulado no CTI, cujo leiaute foi obtido com a versão Beta 2.4 do sistema.

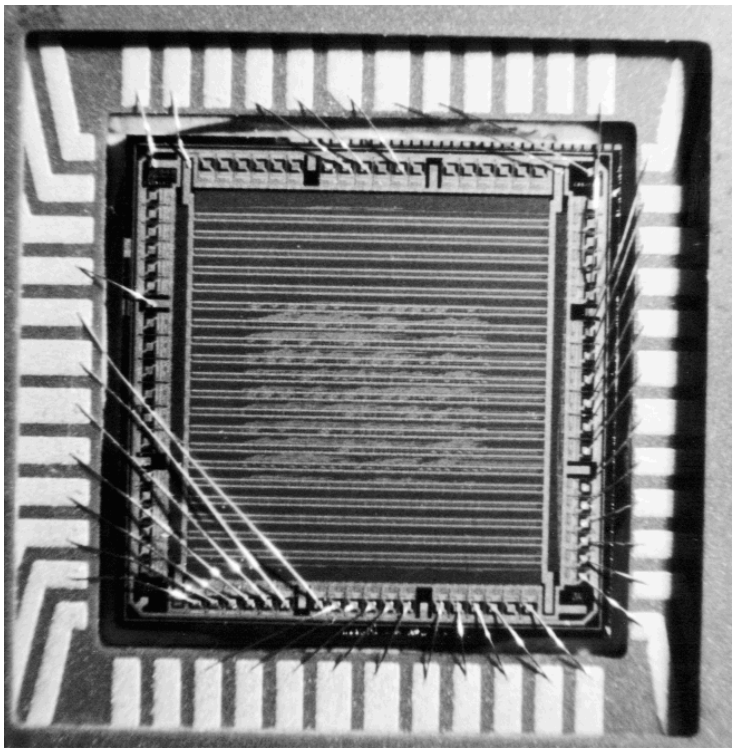


FIGURA 3.18 – Fotografia de um circuito gerado com o GAROTA.

A versão 2.5 opera para um conjunto de seis matrizes, sendo que quatro delas foram sintetizadas pelo próprio roteador (Fig. 3.19). Ao invés do tradicional projeto manual, a arquitetura da matriz foi descrita com os mesmo recursos de posicionamento de blocos e células antes presentes no sistema. Foram implementadas algumas funções adicionais de geração de pequenas células para *underpasses* ou emendas de tamanho variável. Entretanto, a maior parte da automação no projeto destas matrizes se dá unicamente pela definição de variáveis e fórmulas nos arquivos de configuração em linguagem UFDS. Detalhes deste trabalho podem ser vistos em [JOH 2000c]. A Fig. 3.20 mostra um circuito de teste gerado em uma matriz de tamanho personalizado, ilustrando que a automação das matrizes se assemelha a uma forma de geração automática de um leiaute completo para um circuito.

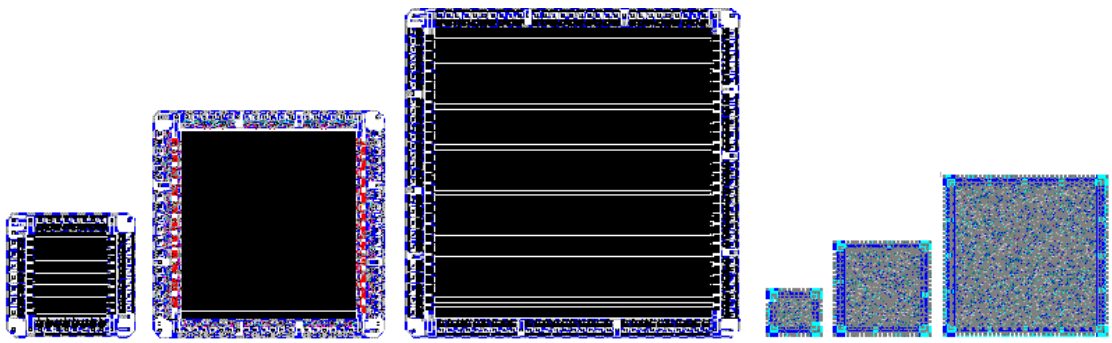


FIGURA 3.19 – Conjunto de matrizes usadas pelo ÁGATA.

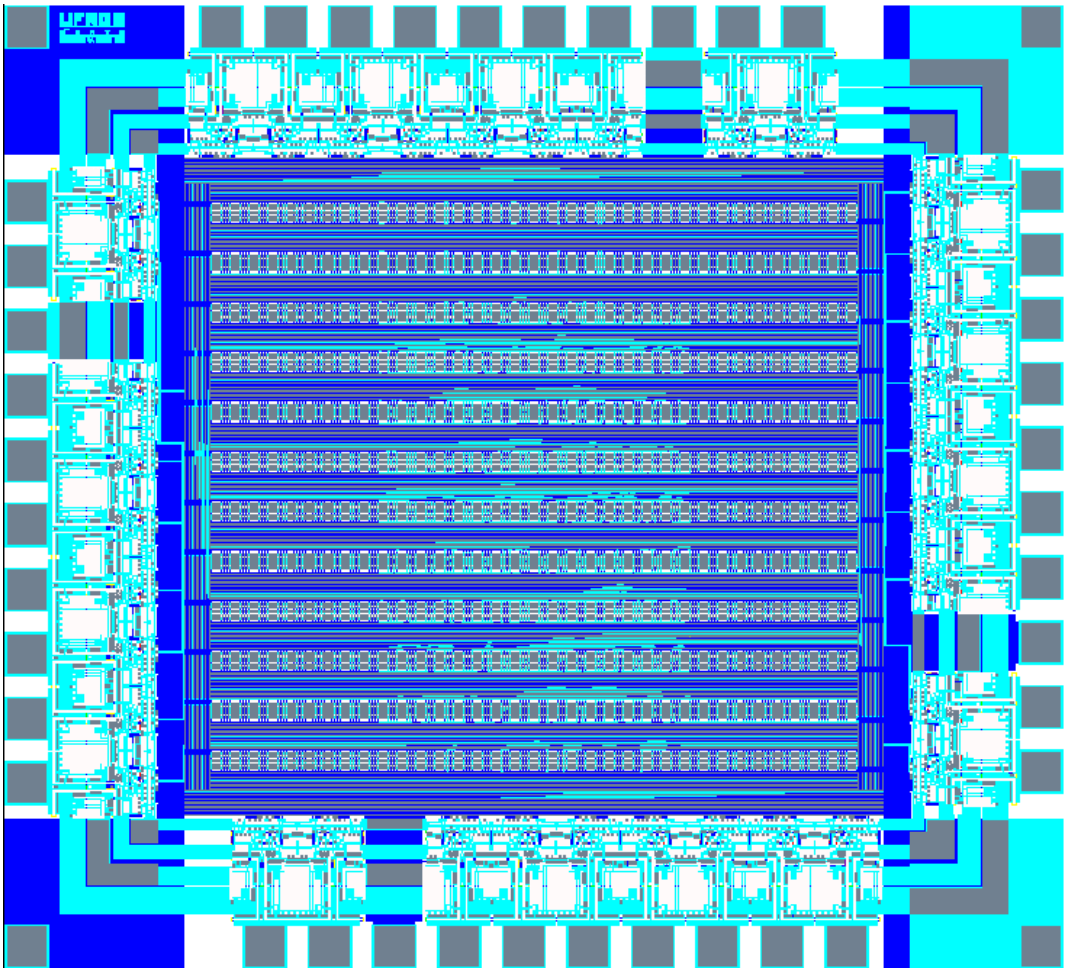


FIGURA 3.20 – Leiaute completo automático do circuito *chip1k*.

Estas são, pois, as contribuições do autor no desenvolvimento de sistemas reais de roteamento e alguns dos conhecimentos teóricos e práticos adquiridos com a sua implementação.

4 Algoritmos de Pesquisa de Caminhos

Algoritmos de pesquisa de caminhos são estudados desde o século XIX, de quando se têm referências a métodos de encontrar a saída de um labirinto. Neste Capítulo serão considerados os algoritmos conhecidos para encontrar o caminho mais curto entre dois nodos de um grafo. Estes algoritmos são objeto de estudo de diversas áreas, como matemática discreta, pesquisa operacional e, dentro da informática, inteligência artificial em especial. Eles possuem inúmeras aplicações devido ao fato de que a solução de problemas computacionais em geral pode ser entendida como um caminho a ser encontrado, o qual representa operações, desde um estado inicial até um estado final desejado. A primeira Seção define o problema, excluindo aqueles domínios e algoritmos que não são de interesse deste trabalho. Segue-se uma apresentação histórica dos algoritmos conhecidos, a qual termina com a apresentação do algoritmo *LCS** desenvolvido como parte desta tese.

4.1 Definição do problema

Seja um **grafo localmente finito** (ou de grau limitado) $G=(V,E)$ de vértices $v \in V$, também chamados de nodos, e arestas $(v_1,v_2) \in E$, também chamadas de arcos. O problema é encontrar, para um par de vértices s e t, origem e destino, um caminho de s a t de tal modo que o caminho encontrado seja o caminho mais curto de todos os caminhos possíveis, ou então provar que não existe nenhum caminho ligando s a t. Um **caminho** de s a t é uma seqüência ordenada de vértices $\{v_1,v_2,...,v_n\}$ de tal forma que $(v_i,v_{i+1}) \in E \; \forall i \mid 1 \leq i \leq n, \; v_1=s, \; v_n=t$, representado por $P_{s\text{-}t}$. Se não há custos diferentes associados às arestas, o **caminho mais curto** é aquele com o menor número de arestas. Quando se tem **custos $c(v_1,v_2)$** associados às arestas, o caminho mais curto é o que apresenta o menor somatório de custos. Este caminho se denomina de **caminho ótimo** e será representado por $P^*_{s\text{-}t}$. Excluem-se, portanto, desta análise aqueles domínios para os quais a função de custo não é puramente aditiva (por exemplo, multiplicativa, aditiva e subtrativa, etc...). As **funções de custo aditivas** garantem a recursividade e preservação de ordem, propriedades pelas quais, se um caminho $P^1_{s\text{-}n}$ é menor que um caminho $P^2_{s\text{-}n}$ então o caminho $P^1_{s\text{-}n} \cup P_{n\text{-}t}$ é menor que o caminho $P^2_{s\text{-}n} \cup P_{n\text{-}t}$ para qualquer outro caminho $P_{n\text{-}t}$. Esta propriedade permite garantir o princípio da excelência[4], o mesmo aplicado à programação dinâmica, pelo qual um caminho é ótimo se e somente se todos os seus trechos são ótimos.

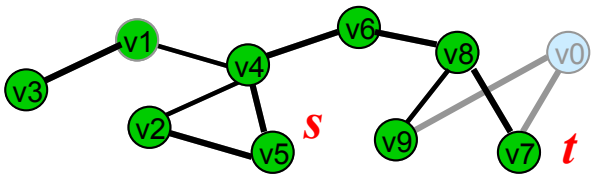


FIGURA 4.1 – Pesquisa do caminho *s*-*t* em um grafo.

[4] Em inglês se usa o termo *optimality*, mas por não haver tradução semelhante, se emprega aqui o vocábulo **excelência** para designar a qualidade daquilo que é ótimo, o melhor possível.

Um algoritmo de pesquisa do caminho mais curto deve encontrar o caminho $P^*_{s\text{-}t}$ $\forall s,t \in V$. Quando se deseja encontrar um caminho entre um conjunto de nodos origem e um conjunto de nodos destino, pode-se transformar esta instância de problema em uma outra com nodos origem e destino únicos pela inserção de dois nodos adicionais *s* e *t*, cada qual com conexões de igual custo para todos os nodos origem e destino originais, respectivamente. Como mencionado em [Pea84], a inserção de nodos *s* e *t* adicionais pode mudar significativamente a estrutura do espaço de estados. Entretanto, algoritmos de pesquisa em geral não dependem desta estrutura.

4.2 Princípios da Pesquisa

Algoritmos de pesquisa funcionam do seguinte modo. Começando pelo nodo *s*, eles pesquisam uma parte do grafo *G* chamada de **árvore de pesquisa** através da aplicação repetida do operador de sucessão. Para cada nodo *n*, um **operador de sucessão** *S(n)* retorna todos os nodos $m \mid (n,m) \in E$. Isto permite tratar **grafos implícitos**, para os quais não se tem armazenados todos os nodos e arcos, mas sim regras para se descobri-los (ou explicá-los), e também grafos infinitos. Por esta razão, a cada vez que se aplica o operador de sucessão a um nodo diz-se que o algoritmo **expandiu** este nodo, e que os novos nodos retornados pelo operador foram **gerados** pelo algoritmo. Um nodo já expandido é também dito **fechado**, e é armazenado na lista de nodos fechados, enquanto os nodos gerados e não expandidos são ditos **nodos abertos**, e são armazenados na lista de nodos abertos, também chamada de frente ou fronteira da pesquisa. Um nodo já descoberto pode estar aberto ou fechado em um dado momento, mas não ambos. Se um algoritmo não pode identificar quando um nodo já foi gerado ou expandido, diz-se que a árvore de pesquisa é parte do **espaço de pesquisa**, que difere do **espaço de estados** porque um mesmo **estado** (nodo) pode aparecer mais de uma vez. Um algoritmo de pesquisa pode ter ou não as seguintes propriedades:

- **Perfeição**[5] – Um algoritmo é dito **completo** se ele termina sempre que houver uma solução. Caso não haja, ou ele expande todos os nodos do espaço, ou permanece eternamente em execução se o grafo for infinito;
- **Admissibilidade** – Um algoritmo é **admissível** quando garante encontrar a solução ótima se uma solução existe;
- **Dominância** – Um algoritmo **domina** outro se todo o nodo expandido por ele necessita também ser pesquisado pelo outro;
- **Excelência** – Um algoritmo é **ótimo** em uma classe de algoritmos se ele domina todos os algoritmos possíveis pertencentes a esta classe;

4.3 Algoritmos de pesquisa

Os algoritmos de pesquisa conhecidos se dividem em diversas classes, conforme será visto a seguir. Alguns algoritmos pertencem a mais de uma classe, mas aqui serão introduzidos por ordem cronológica aproximada. Com o passar do tempo, algoritmos mais específicos foram propostos, e enquanto as primeiras classes representam apenas um algoritmo, as últimas envolvem muitos.

[5] Neste caso opta-se pelo termo perfeição para designar a qualidade daquilo que é completo, terminado, já que não há tradução semelhante para o termo inglês *completeness*.

4.3.1 Pesquisa primeiro em profundidade

Conhecida como *depth-first search*, ou *DFS*, se caracteriza pelo fato de que tão logo um novo nodo é gerado ele é selecionado para ser expandido, e, portanto os caminhos são pesquisados em profundidade. Este tipo de pesquisa foi introduzido por Lucas em 1882 e Tarry em 1895, segundo [EVE 79]. A pesquisa em profundidade pode ser a mais eficiente para encontrar o destino quando o grafo é uma árvore finita e o destino está em uma das folhas (Fig.4.2a).

4.3.2 Pesquisa primeiro em largura

Conhecida como *breadth-first search,* ou *BFS*, é a pesquisa que primeiro expande todos os nodos que se encontram há uma determinada distância da origem *s*, antes que seja expandido qualquer nodo mais distante (Fig. 4.2b). Está técnica também é conhecida como técnica de marcação (*labeling*), tendo sido introduzida por Moore em 1957 ([EVE 79]), e pode ser implementada com um procedimento do tipo *FIFO* (*First In First Out*), ou fila. Caminhos de mesmo tamanho são avaliados ao mesmo tempo. O algoritmo de **Dijkstra** é uma variante de *BFS* onde os nodos abertos são ordenados pela soma dos custos das arestas e não pelo número delas como antes. Ele possui a mesma característica de pesquisa em largura, mas poder-se-ia classificá-lo como um tipo de pesquisa ordenada, conforme definido a seguir. A principal diferença prática entre BFS e o algoritmo de Dijkstra é que este último exige uma ordenação não trivial dos nodos já descobertos, tendo maior complexidade de tempo.

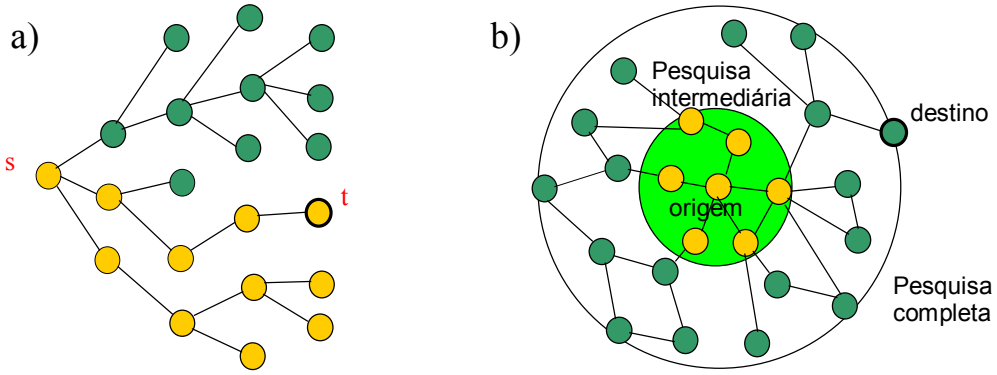


FIGURA 4.2 – Pesquisa em profundidade (*DFS*) e em largura (*BFS*).

4.3.3 Pesquisa ordenada

Este termo aparece em [NEVE71] como qualquer pesquisa em que os nodos abertos são ordenados por um critério qualquer para sua posterior expansão. Assim, o termo se aplica a qualquer algoritmo de pesquisa, pois os primeiros implicitamente possuem uma determinada ordenação: em *DFS*, o último a ser gerado é o primeiro a ser expandido (*LIFO*), e em *BFS* o primeiro a ser gerado é o primeiro a ser expandido (*FIFO*). Entretanto, este termo só toma sentido quando os custos associados requerem um passo de ordenação, principalmente com o surgimento das novas técnicas de pesquisa informada, como segue.

4.3.4 Pesquisa heurística ou informada

Este tipo de pesquisa ocorre quando a ordenação dos nodos usa informações sobre a estrutura do grafo não representada por *(V,E)*, de forma a tentar descobrir quais nodos são mais promissores na pesquisa. O termo "heurística" (descobrir) é usado inicialmente por Samuel em 1959 ([NIL 71]), Lin em 1965 ([PEA 84]) e [HAR 68], ao passo que o termo "informada" é usado por Doran e Mickie em 1966. Outra denominação muito comum utilizada é "**pesquisa primeiro o melhor"** (*best-first search*). Formalmente, define-se uma função *f* para cada nodo *n* gerado pelo algoritmo, onde *f(n) = g(n) + h(n)*. O algoritmo **A***, apresentado em [HAR 68], expande os nodos por ordem crescente de *f(n)*. Computa-se *g(n)* como a soma dos custos do caminho descoberto até o nodo *n* (como em Dijkstra), e *h(n)* como a estimativa do custo do caminho restante de *n* até o destino *t* (Fig. 4.3). A função *h* é chamada de **função de estimação** e deve ser provida ao algoritmo como informação extra ao grafo, podendo ser computada com mais ou menos efeito dependendo da estrutura deste. Desta forma, a pesquisa heurística expande primeiro os nodos com maior probabilidade de pertencerem ao caminho ótimo, e é considerada como pesquisa com perímetro de avaliação constante em contraposição com a pesquisa com perímetro de profundidade constante que resulta de *BFS* ([VEM 91]). Já em [HAR 68] é apresentado um teorema mostrando a excelência de **A*** em uma determinada classe de algoritmos (explicada adiante), o que é confirmado em [DEC 85], além de diversas outras propriedades importantes. O algoritmo A* é apresentado na Fig. 4.4, segundo a definição mais exata encontrada em [DEC 85]. As operações no passo 6 exigem a atualização dos valores atribuídos aos nodos fechados e abertos. A reabertura de nodos não ocorre sob determinadas condições (explicadas adiante), mas a atualização de nodos abertos sempre acontece e implica na reordenação da lista de nodos abertos, já que esta deve ser mantida ordenada pelos valores de *f(n)*.

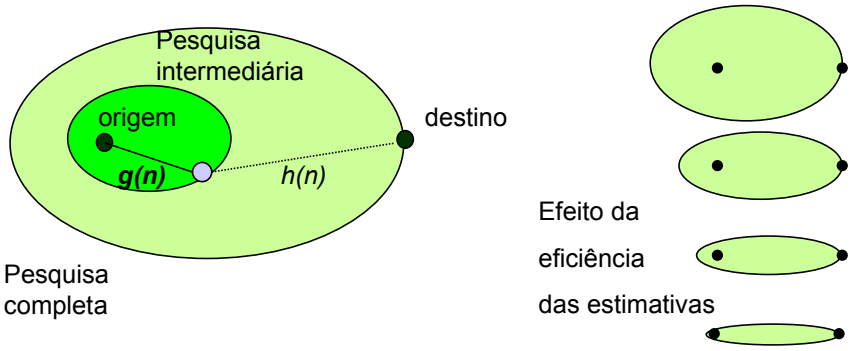


FIGURA 4.3 – Pesquisa heurística ou informada: algoritmo A*.

Algoritmo A*
1. Marcar *s* como aberto e calcular *f(s)*
2. Se a lista de abertos está vazia, terminar porque não existe solução.
3. Selecionar da lista de abertos o nodo *n* com menor *f(n)*. (Resolver empates arbitrariamente, mas sempre em favor do destino), retirá-lo da lista de nodos abertos e colocá-lo na lista de nodos fechados.
4. Se $n = t$, terminar com sucesso, traçando o caminho ótimo com os ponteiros armazenados de *t* a *s*.
5. Caso contrário, expandir *n* pela aplicação do operador de sucessão.
6. Para cada nodo *m* sucessor de *n*: calcular *f(m)*; se m não estiver nem na lista de fechados e nem na lista de abertos, adicioná-lo à lista de abertos anotando *f(m)* e um ponteiro para *n*; caso contrário, comparar o antigo valor de *f(m)* com o novo. Se o novo for menor, substituir o valor e o ponteiro antigos, e movendo m para lista de abertos se estava na lista de fechados.
7. Voltar para o passo 2.

FIGURA 4.4 – O Algoritmo A*.

4.3.5 Pesquisa bidirecional

Os grafos aos quais se aplica a pesquisa podem ou não ser dirigidos. Mesmo nos casos em que é dirigido, pode-se dispor de operadores de sucessão reversíveis. Um operador é reversível quando existe uma função $S_r(n_2)$ que retorna n_1 como um dos sucessores sempre que a função sucessora $S(n_1)$ retorna n_2 como sucessor. Se, além disto, se pode facilmente identificar um mesmo nodo do espaço de estados, pode-se aplicar pesquisa bidirecional. A **pesquisa bidirecional** se constitui de dois processos de pesquisa idênticos e simultâneos, um partindo da origem e usando o operador *S* e outro partindo do destino e usando o sucessor S_r (Fig. 4.5). O processo termina após algum nodo ser reconhecido por ambas as frentes de pesquisa. Esta técnica foi introduzida em [POH 69] utilizando o algoritmo de *BFS*. Seu potencial reside no fato de que se o caminho mais curto tem um tamanho *l*, então duas pesquisas em árvores de profundidade *l/2* serão bem menores que uma só pesquisa de profundidade *l* quando o fator de ramificação do grafo for maior do que 1. De fato, verifica-se que está técnica reduz muito o número de nodos expandidos em aplicações práticas quando não se considera nenhuma heurística.

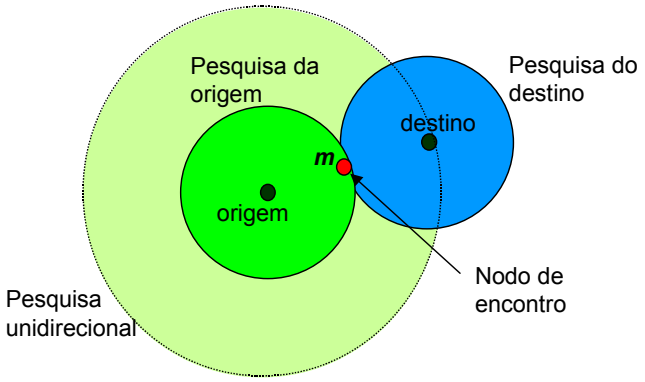


FIGURA 4.5 – Pesquisa bidirecional: duas frentes opostas.

Um fator importante em pesquisa bidirecional é o **critério de escolha de direção**.

O critério mais simples é a **alternância**. A cada nodo expandido muda-se o sentido da expansão, de modo que ambas têm a mesma oportunidade. Entretanto, a estrutura e os custos do grafo podem ser bastante diversos em torno da origem e em torno do destino. Nicholson, em 1966, (citado por [KWA 89]) propôs o **princípio da eqüidistância** para pesquisa bidirecional informada, pelo qual deve-se expandir a partir da frente que possui o nodo mais próximo do destino. Pohl apresenta em 1969 o **princípio da cardinalidade** ([POH 69] referenciado em [POH 71]), pelo qual deve-se expandir sempre a partir daquela frente que tem o menor número de nodos abertos, princípio alimentado por argumentos teóricos e experimentação. É importante notar, entretanto, que estes princípios foram encontrados muito prematuramente. Ao observar o comportamento do A* a partir da origem e a partir do destino, se vê que é mais eficiente quando parte da posição cujo acesso é mais difícil, com maiores custos. Este comportamento sugere maior investigação

4.3.6 Pesquisa heurística bidirecional

Motivados pelos ganhos obtidos com as técnicas heurística e bidirecional, surgiram diversos trabalhos tentando aliar ambas para se obter melhor desempenho. O primeiro trabalho foi apresentado em [POH 71] com o algoritmo *BHPA*. Nenhum resultado numérico é reportado, mas é demonstrada a dificuldade de manter as informações necessárias e identificar o término do processo de forma que seja admissível. Lá aparece pela primeira vez o chamado problema das **frentes desencontradas (***missing fronts problem***)**. Na tentativa de explicar a razão pela qual o algoritmo heurístico bidirecional teve pior desempenho quando comparado com **A***, verificou-se que, quando há caminhos equivalentes entre origem e destino, é possível que as duas pesquisas progridam por caminhos diferentes, duplicando os esforços em vez de se encontrarem no meio do mesmo caminho. Este problema foi considerado por muito tempo como a razão do fracasso, mas, como será esclarecido adiante, não o é.

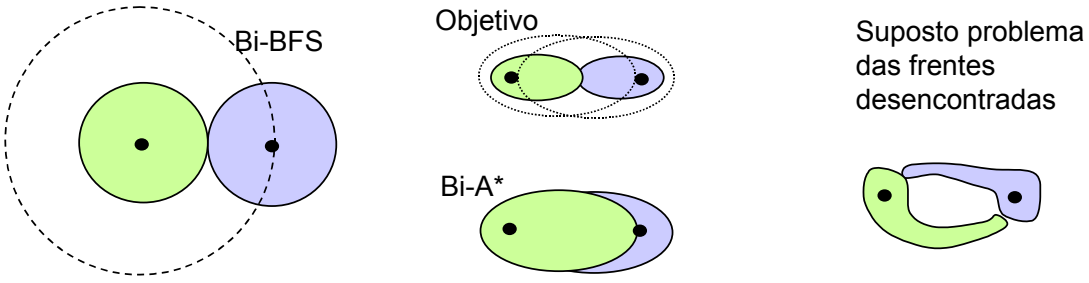


FIGURA 4.6 – Objetivo e realidade da pesquisa heurística bidirecional.

Mais tarde, James Kwa identificou corretamente que após as frentes se encontrarem, ambas passavam a expandir os mesmos nodos, e também que havia diversas oportunidades de eliminar determinados nodos do processo, provando-se que não mais poderiam pertencer ao caminho ótimo ou que o caminho ótimo até eles já havia sido encontrado. A operação de poda (*pruning*) já havia sido estudada por Doran e Mickie em 1966 e Doran em 1977 (segundo [NIL 71]). De maneira semelhante, eliminando-se as interseções através das operações que chamou de *nipping, pruning, trimming* e *screening*, e adiantando as condições de término, Kwa obteve o algoritmo **BS*** que foi provado ser admissível ([Kwa89]). Seu esforço, entretanto, ainda não foi suficiente para justificar a existência da pesquisa heurística bidirecional. Em média, ele

é melhor que os algoritmos heurísticos bidirecionais anteriores, mas um pouco pior do que o unidirecional **A***, tendo somente utilidade teórica.

4.3.7 *Wave-shapping* e destinos intermediários.

Estes termos são usados para designar aqueles processos de pesquisa bidirecional nos quais a função de estimação não usa o nodo destino *t* como objetivo, mas sim os nodos abertos presentes na frente de pesquisa contrária. A idéia aparece em [POH 71], segundo o qual já havia sido proposta por Doran e Mickie. Em [POH 71], são apenas mencionados os termos "*shapping"* e "posições intermediárias", referindo-se ao método de escolher um dos últimos nodos expandidos pela frente contrária como o destino para os cálculos estimados da frente em questão. Na época foram feitos alguns comentários sobre a freqüência de atualização deste novo destino e seus possíveis problemas.

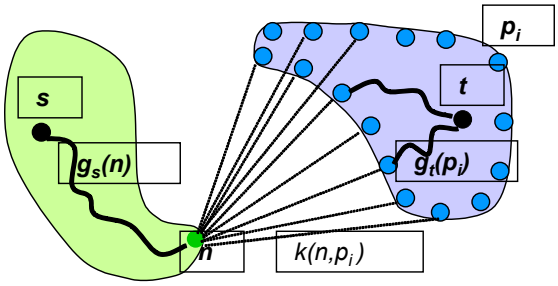


FIGURA 4.7 – *Wave-shapping*: cálculo de distância para nodos da frente oposta.

Com o tempo, consagrou-se o termo ***wave-shapping*** para designar os algoritmos em que, para cada nodo *n* sendo gerado, calcula-se o valor de *h(n)* como a menor das distâncias de *n* até cada um dos nodos abertos da frente oposta somada ao custo *g* de cada um destes nodos [CHA 77]. Não há dúvidas de que esta é a melhor forma de promover o encontro das duas frentes, mas tem um custo computacional muito mais alto para cada nodo expandido. Em geral, o tempo adicional não compensa a redução no número de nodos expandidos. O uso de nodos intermediários já mencionado por [POH 71] deu origem à técnica conhecida como ***D-node retargeting*** ([POL 84]). Neste tipo de pesquisa, um dos nodos de uma frente é escolhido como o destino para a frente oposta. Existem algoritmos em que não somente este nodo D é selecionado para cálculo de destino, como é usado para acelerar a pesquisa de forma não admissível. Assim, pode-se obter algoritmos muito mais rápidos para encontrar caminhos quando se pode abrir mão de que o caminho seja ótimo. A técnica também se assemelha a um tipo de pesquisa denominado de **pesquisa por estágios**. Neste caso, a pesquisa pára de tempos em tempos, e, do conjunto de nodos presentes na sua frente, uma avaliação mais criteriosa é feita para que apenas uns poucos nodos sejam selecionados como origem para uma nova pesquisa intermediária. Quando pode-se calcular funções de estimação de qualidade diferente em determinados momentos, pode-se ter algoritmos eficientes assim, em domínios específicos.

4.3.8 Pesquisa por perímetro

Embora possa ser considera um caso particular de pesquisa bidirecional heurística, e em especial de *wave-shapping*, esta técnica surgiu para evitar os inconvenientes encontrados nas duas, ao mesmo tempo em que se exploram as vantagens potenciais sobre os algoritmos unidirecionais, a saber, sobre o A*. A

pesquisa por perímetro se constitui em dois processos de pesquisa **seguidos**, com objetivos diferentes, partindo o primeiro do destino e o segundo da origem. O primeiro realiza uma pesquisa do tipo *BFS* até um perímetro de tamanho previamente estipulado. Após isto, uma pesquisa usando outra técnica, que pode variar de *DFS* até A*, é feita a partir da origem, mas tendo como destino todos os nodos resultantes no perímetro da primeira pesquisa. O termo surgiu em [DIL 94], e é usando em outros trabalhos recentes, como em [KAI 95] e [MAN 95], onde generalizações maiores são feitas, mas já se encontram processos semelhantes anteriormente [GHO 91]. Este tipo de pesquisa é importante porque apresenta menor sensibilidade à escolha da origem se comparado com A*, e vence facilmente as dificuldades encontradas quando o nodo destino se situa envolto em um espaço complexo e de difícil estimação. Nestes casos ela apresenta melhores resultados se comparado com A*, a um custo relativamente baixo de computação. O custo por nodo expandido ainda é maior, pois todos os nodos do perímetro precisam ser examinados, mas não cresce descontroladamente como em *wave-shapping*. O mais interessante no entanto é o fato de que a pesquisa por perímetro demonstra de forma prática a viabilidade de algoritmos que superem o A*, conforme a teoria indicava, e, como veremos, é uma particularização do novo algoritmo apresentado neste trabalho.

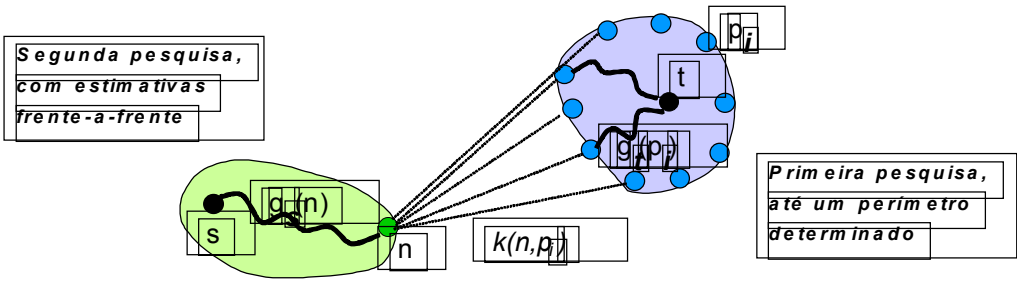


FIGURA 4.8 – Pesquisa por perímetro: duas pesquisas seguidas (*BFS* e A*).

4.3.9 Outras técnicas de pesquisa

Aqui se relacionam algumas outras técnicas particulares de pesquisa para fins de referência, as quais ou são ineficientes, ou são de interesse particular de um determinado domínio. São elas:

- Pesquisa bidirecional com pares de nodos – A idéia aparece em [POH 71], mas é desenvolvida em [ECK 94], e usa apenas uma lista aberta com pares de nodos mais próximos tomados um de cada frente de pesquisa. Requer tempo de processamento alto (> $O(n^2)$) e muita memória (> $O(n^2)$).
- Pesquisa com admissibilidade limitada – Em [POH 70] encontra-se um estudo sobre o erro em pesquisas heurísticas, mas um trabalho mais interessante é apresentado em [KOL 93], onde a percentagem de erro da solução pode ser controlada por um fator ε, de forma que o algoritmo é chamado de ε–admissível.
- Pesquisa com memória limitada – Diversas técnicas são usadas para a execução de pesquisa eficiente quando a memória é limitada, como por exemplo, em [KAI 95]. Outros trabalhos nesta categoria são citados em [KOR 93].
- Pesquisa em tempo real – A pesquisa em tempo real se distingue das demais pelo fato de que exige que algum agente esteja fisicamente na posição sendo pesquisada, como um robô em um mapa. Assim, não se pode pesquisar caminhos

em paralelo, e emprega-se processos semelhantes a *DFS* [ISH 95].

- Pesquisa com ilhas ou subobjetivos – Em determinados domínios pode-se identificar que os caminhos ótimos devem obrigatoriamente passar por determinados nodos intermediários, como pontes sobre um rio. Neste caso, usar estes nodos como objetivo de pesquisas parciais acelera o processo de pesquisa completo. Em [DIL 95] é apresentado um algoritmo admissível que considera subobjetivos possíveis (não obrigatórios).

4.4 Propriedades do Algoritmo A*

O algoritmo A* é o mais importante da classe dos algoritmos heurísticos unidirecionais. Como é demonstrado em [DEC 85] ele é ótimo dentre todos os algoritmos desta classe se um conjunto bem pequeno de condições é satisfeito. Se algumas delas não o são, [DEC 85] também demonstra que embora o A* não seja ótimo, não existe outro algoritmo que se possa garantir ser. Algumas destas propriedades não foram adequadamente exploradas para a pesquisa bidirecional. Por esta razão, se faz aqui uma síntese que serve de base para o desenvolvimento de um algoritmo bidirecional mais eficiente.

4.4.1 Valores ótimos

Antes de revisar as propriedades específicas do algoritmo A* cabe definir alguns valores ótimos conforme a terminologia normalmente empregada na área. Os valores de $f^*(n)$, $g^*(n)$ e $h^*(n)$ serão estimados respectivamente pelas funções f, g e h durante a execução do algoritmo A*. Seja:

$P^*_{n\text{-}m}$	caminho ótimo de n a m, $n,m \in V$
$P^*(n)$	caminho ótimo de s a t obrigado a passar por n, $n \in V$
$k^*(n,m)$	custo do caminho ótimo entre n e m, $n,m \in V$
$g^*(n)$	$= k^*(s,n)$, custo do caminho ótimo entre s e n, $n \in V$
$h^*(n)$	$= k^*(n,t)$, custo do caminho ótimo entre n e t, $n \in V$
$f^*(n)$	$= g^*(n) + h^*(n)$, custo do caminho $P^*(n)$, $n \in V$
C^*	$= k^*(s,t)$, custo do caminho ótimo $P^*_{s\text{-}t}$

4.4.2 Propriedades dos valores ótimos

Os valores ótimos possuem algumas propriedades importantes para compreender as propriedades que as estimativas possuem ou não, e também para que sejam exploradas adequadamente em domínios específicos. Por exemplo, pode-se conhecer a gama de valores que uma função pode assumir, ou quais valores são independentes entre si. As propriedades de valores ótimos são:

- **Consistência** – $k^*(n1,n3) \leq k^*(n1,n2) + k^*(n3,n2)\ \forall\ n1,n2,n3 \in V$
- Corolários – $g^*(n1) \leq k^*(n1,n2) + g^*(n2)$, e $h^*(n1) \leq k^*(n1,n2) + h^*(n2)$
- **Ordenação** – $g^*(n_i) \leq g^*(n_{i+1})$, $h^*(n_i) \geq h^*(n_{i+1})$, $f^*(n_i) = f^*(n_{i+1}), \forall\ n_i, n_{i+1} \in P^*_{s\text{-}t}$
- **Independência** – Os valores de $g^*(n)\ \forall\ n \in V$ são completa e exclusivamente determinados pela escolha do nodo s, e não dependem do nodo destino t. Os valores de $h^*(n)\ \forall\ n \in V$ são completa e exclusivamente determinados pela

escolha do nodo *t*, e não dependem do nodo origem *s*.

4.4.3 Propriedades das estimativas

A função de estimação *h* pode ou não ter as propriedades definidas abaixo. A admissibilidade é necessária para que qualquer algoritmo seja admissível. A consistência permite que a função *f* seja monótona, evitando que um mesmo nodo seja expandido mais de uma vez pela garantia de que o caminho ótimo até ele já tenha sido descoberto. Como citado em [HAR 68], a maioria das funções de estimação que se pode elaborar são consistentes. Quanto mais informada for uma função, maior será seu poder de poda. O **poder de poda** é a capacidade que a função confere ao algoritmo para julgar que um nodo não pode estar no caminho ótimo, através da computação de um valor mais alto de *f(n)*. Pode-se facilmente demonstrar também que se uma função mais informada é disponível, de forma alguma reduzir os valores de *h* ajudará a encontrar o caminho ótimo mais rapidamente. Entretanto, [HAR 68] já mencionava que se há uma complexidade de tempo maior do que *O(1)* no cálculo de *h(n)*, então usar uma função menos informada mas de menor complexidade pode acelerar a pesquisa. Formalmente, as propriedades são definidas do seguinte modo:

- *h* é **admissível** se $h(n) \leq h^*(n)$ para qualquer $n \in V$ (subestimado, Fig. 4.9a);
- *h* é **monótona** se $h(n_1) \leq c(n_1,n_2) + h(n_2)\ \forall\ n_1,n_2 \in V \mid n_2 \in S(n_1)$;
- *h* é **consistente** se $h(n1) \leq k^*(n_1,n_2) + h(n_2)\ \forall\ n_1,n_2 \in V$. Prova-se facilmente que *h* é consistente se for monótona, por indução nos sucessores (Fig. 4.9b);
- h_2 é **mais informada** do que h_1 se $h_1(n) \leq h_2(n)\ \forall\ n \in V$;
- *h* é **perfeita em *V'*** se $h(n) = h^*(n)\ \forall\ n \in V'$, sendo *V'* subconjunto de *V*;
- *h* é **estática** se apresenta o mesmo valor entre nodos em qualquer tempo;
- *h* é **dinâmica** se apresenta valores diferentes em diferentes momentos na execução do algoritmo;

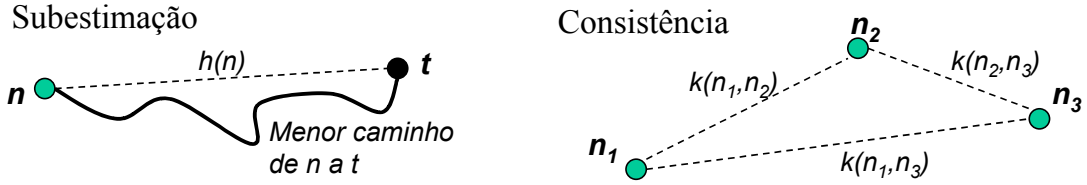


FIGURA 4.9 – subestimação e consistência de funções heurísticas

4.4.4 Empates

Um **empate** ocorre quando mais de um nodo na lista de abertos possui o mesmo valor de *f*. É verdade que isto é pouco freqüente em muitos domínios, e também que pode-se minimizar sua ocorrência usando funções *f* contínuas em vez de discretas [PEA 84]. Entretanto, em alguns domínios regulares, como é o caso de roteamento quando a área destinada às conexões ainda está em grande parte vazia, a ocorrência de empates pode ser altíssima, e se deseja justamente que todos os caminhos sejam considerados eqüivalentes (aqui supõe-se que mudanças de direção já estão sendo consideradas com custo mais alto). Além disto, o desempenho e a excelência de um algoritmo de pesquisa dependem fortemente do número de empates e da **regra de quebra de empates**. No algoritmo A*, assume-se que os empates devem ser quebrados

arbitrariamente, desde que em favor do nodo destino, isto é: se o destino for um dos nodos empatados, ele deve ser selecionado primeiro [HAR 68]. Para uma comparação justa entre diferentes algoritmos, deve-se empregar em ambos a mesma regra. Então, em princípio usar-se-á uma generalização da regra proposta por [HAR 68] e [DEC 85]: dentre os nodos com mesmo valor de *f*, toma-se para expansão o que tiver maior valor de *g*. Esta regra não somente satisfaz a preferência pelo destino, como, em situações com estimações perfeitas, provoca uma espécie de subpesquisa em profundidade até o destino. Para empates que ainda ocorram com *f* e *g*, técnicas mais específicas de direcionamento podem ser empregadas, como será visto em roteamento. Um **empate crítico** ocorre quando $f(n_1) = f(n_2) = C^*$. Como será visto adiante, empates não críticos não têm efeito sobre o resultado da pesquisa, mas os críticos têm. Em princípio não é possível determinar durante a execução do algoritmo se um empate será crítico ou não, já que não se conhece o valor de C^*. Sua ocorrência no domínio, entretanto deve ser avaliada, e, como será visto, em algoritmos bidirecionais existe um modo de identificar tais empates, apesar de não ser possível determinar quantos há.

4.4.5 Propriedades características do A*

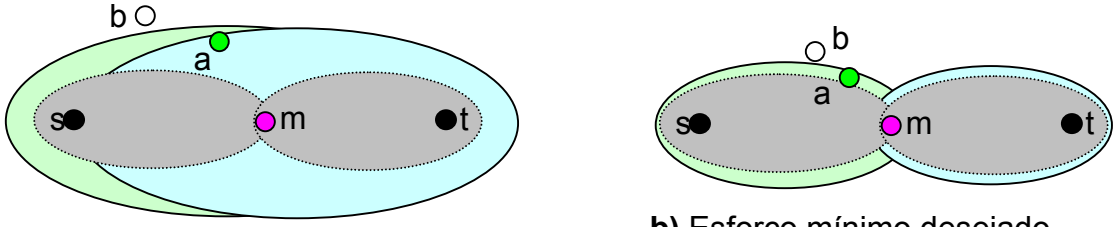
As propriedades listadas a seguir foram extraídas de [HAR 68] [NIL 71] [DEC 85] e [PEA 84], onde se pode verificar os lemas e teoremas que as suportam.

- **Continuidade** – Em qualquer tempo antes do término, para qualquer nodo *n* não fechado existe um nodo *n'* aberto pertencente a $P^*_{s\text{-}n}$ com $g(n') = g^*(n')$;
- **Perfeição** – A* é completo para grafos localmente finitos;
- **Admissibilidade** – A* é admissível se a função de estimação for admissível;
- **Condição suficiente para expansão** – O algoritmo A* expande todos os nodos com $f(n) < C^*$;
- **Condição necessária para expansão** – Somente nodos com $f(n) \leq C^*$ podem ser expandidos pelo algoritmo;
- **Indeterminismo em empates críticos** – O número de nodos expandidos pelo algoritmo com $f(n) = C^*$ depende da instância do problema e da regra de quebra de empates;
- **Monotonia** – quando a função de estimação é consistente, se n_2 é expandido depois de n_1 então $f(n_1) \geq f(n_2)$. Isto implica em que o nodo com menor $f(n)$ possui $g(n) = g^*(n)$, e A* nunca reabre nodos já expandidos;
- **Dominância** – Se h_2 é mais informada do que h_1, então A* usando h_2 não pode expandir mais nodos do que A* usando h_1;
- **Excelência** – Se as estimativas são consistentes e não ocorrem empates críticos, A* é ótimo dentre todos os algoritmos heurísticos unidirecionais não mais informados do que ele.

4.5 Observações sobre pesquisa heurística e bidirecional

Nenhum algoritmo heurístico bidirecional genérico apresentado até então possui desempenho médio melhor do que o algoritmo unidirecional A*. Por outro lado, não se pode demonstrar que A* é ótimo com relação a algoritmos que não são unidirecionais. De fato, comprova-se claramente que técnicas bidirecionais específicas possuem vantagens em diversos casos. Especial atenção deve ser dada a pesquisa por perímetro, a qual apresenta bom desempenho [MAN 95]. Além da maior dificuldade de

implementação e dedução de teoremas, o argumento mais usado para justificar a ineficiência dos algoritmos heurísticos bidirecionais tem sido o problema das frentes desencontradas. Entretanto, demonstrou-se recentemente que as frentes de pesquisa se encontram facilmente mesmo sem técnicas de direcionamento, mas que um grande esforço de pesquisa adicional é necessário para demonstrar que o caminho encontrado é ótimo [KOL 93]. A Fig. 4.10a ilustra este comportamento. Desta forma, o potencial da pesquisa heurística bidirecional reapareceu, mas ainda não havia sido proposto um algoritmo simples e genérico que fosse eficiente, como enfatiza [KAI 95].



a) Esforço necessário em Bi-A*

b) Esforço mínimo desejado em pesquisa bidirecional

FIGURA 4.10 – Esforço de algoritmos bidirecionais. s) origem; t) destino; m) primeiro encontro; a) nodo que pode estar em caminho ótimo; b) nodo que pode ser podado.

A primeira observação importante é de que o problema das frentes desencontradas somente pode existir no caso de haver empates críticos, devido às condições necessária e suficiente para expansão. A existência de empates críticos é uma realidade, mas em geral estes são limitados, e então, no caso de haver perdas, estas não podem ser muito grandes, e não anulariam os ganhos médios obtidos. De fato, a observação de [KOL 93] revela que o problema encontrado é a falta de poder de poda da função de estimação tradicional quando usada nos algoritmos bidirecionais. A poda não considera o esforço de pesquisa já feito pela frente oposta. Isto nos permite relembrar um princípio da pesquisa heurística: **o poder de um algoritmo heurístico admissível não está em quão rápido ele pode encontrar um caminho unindo origem a destino, mas em quão rápido (e eficientemente) ele pode computar valores mais altos de *f(n)* para os nodos que expande, de modo a ter maior poder de poda**.

A segunda observação importante é a de que **a função *g* de uma frente de pesquisa corresponde a função *h* da frente oposta, e vice-versa**. Este fato é inicialmente observado na tentativa de utilizar menos memória por nodo fechado no espaço de estados. A diferença destas funções é que são estimativas de f^* e h^* "no sentido contrário", isto é, enquanto g sobrestima g^* da sua própria frente, h subestima g^* da frente oposta. Estes valores, portanto, não podem ser misturados em ambas as frentes. Entretanto, com estimação consistente, a cada nodo n expandido se tem $g(n) = g^*(n)$, e então este valor pode ser usado como $h^*(n)$ pela frente oposta. Este comportamento sugere o conceito de **visibilidade**. As listas de nodos fechados podem ser visíveis por ambas as frentes, enquanto que as listas de nodos abertos possuem informações estimadas que somente são bem interpretadas pela frente que as gerou.

A última observação importante diz respeito ao tipo de estimação. Em [KAI 95] encontram-se explicitamente mencionadas as três classes de estimação, embora seja com nomes ligeiramente diferentes: **a) estática; b) dinâmica; e c) frente a frente**. A primeira representa uma função h usual de A*. A segunda classe é dada quando se

dispõe de algum meio pelo qual se possa calcular valores maiores de *h* para um mesmo nodo quando se está mais próximo do destino. Já a terceira representa os algoritmos de *wave-shapping*. A estimativa dinâmica é mais eficiente do que a estática, e a estimativa frente a frente é a mais eficiente de todas em número de nodos expandidos, mas possui a desvantagem do altíssimo custo computacional, sendo, portanto, desaconselhada.

4.6 O Algoritmo LCS*

As funções de estimação dinâmicas surgiram provavelmente aliadas a características especiais da estrutura de determinados domínios. Porém, com o conceito de visibilidade, se percebe ser possível utilizar estimação dinâmica alimentada pela frente de pesquisa oposta de forma a aumentar o poder de poda do algoritmo. A contribuição da frente oposta deve ser um único valor, evitando-se cálculos não lineares em tempo, e esta contribuição dá o caráter cooperativo do algoritmo. A visibilidade é implementada armazenando referências aos nodos do espaço nas listas de nodos abertos de cada frente. Os valores estimados de *g* e *h* são então privados, e um mesmo nodo pode aparecer várias vezes, evitando atualizações nestas listas. Já os conjuntos fechados são armazenados em uma estrutura pública, e contém somente os valores ótimos *g** já encontrados, junto com os ponteiros para os antecessores no caminho ótimo. O algoritmo que usa esta técnica deriva diretamente do BS* de Kwa, e é batizado como LCS* (*Lowerbound Cooperative Search*). As informações armazenadas e o pseudo-código do algoritmo aparecem em [JOH 2000a], cujo texto se encontra no anexo 3. Foi provado que LCS* é completo e admissível [JOH 2000b] e estas provas podem ser observadas no texto do anexo 4. O algoritmo somente pode ser mais eficiente do que o A*, remediando a situação da Fig. 4.10a, quando usa a estimação dinâmica, cuja implementação é definida a seguir.

4.6.1 Função de estimação dinâmica

Tomando todos os nodos abertos da frente que parte do destino *t*, se pode calcular a mínima diferença entre *g(p)* e *k(p,t)*. Sabe-se que *k(p,t)* é igual ao valor de *h(p)* estimado pela frente da origem. Então, este resultado é o mínimo custo adicional possível para chegar ao destino, através dos nodos abertos e fechados da frente oposta, que será acrescido ao valor estático de estimativas *h* de qualquer nodo fora daquele conjunto fechado, quando um caminho for encontrado. Este valor será chamado de **resistência** da frente oposta, e representado por Ω_t. De forma semelhante se tem Ω_s.

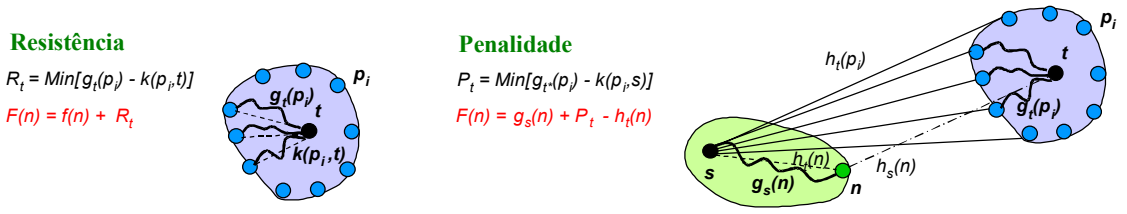


FIGURA 4.11 – Cálculo de valores de resistência e penalidade.

Acontece que quanto melhor for a função de estimação, menores serão os valores de resistência, já que corresponderão ao lado "oposto" (menor) da pesquisa da frente

oposta. Isto ocorre justamente em conseqüência da estimativa daquela frente oposta. Para compensar este efeito, se usa um outro par de valores que denominamos de penalidades. O Valor de **penalidade** de uma frente é o mínimo dentre os custos do melhor caminho possível entre *s* e *t* que passa por cada um dos nodos abertos da frente menos a distância estimada deste nodo até a origem da própria frente. Neste valor está automaticamente computado o custo adicional ($g - h$) e o custo de posicionamento contrário dos nodos ($h_s + h_t$), mas para que seja adicionado a uma estimativa da outra frente, esta precisa subtrair o custo estimado até cada nodo seu, para que este não seja computado duas vezes.

Estes valores podem ser computados e adicionados às estimativas, desde que se observe os seguintes efeitos:

- Um valor qualquer pode ser adicionado em uma função de estimação *h* qualquer preservando todas as suas propriedades e tornando-a mais informada. Entretanto, se o valor de *h* é dinâmico, sua alteração deve ser executada simultaneamente em todos os nodos da lista de abertos;
- Os valores de resistência definidos acima são válidos como custo adicional para penetrar a frente oposta até o destino passando não somente pelos nodos fechados, mas também pelos nodos abertos. Em um algoritmo bidirecional, pode acontecer de nodos fechados em diferentes frentes serem vizinhos. Em conseqüência, cada frente terá nodos abertos que estão além da fronteira oposta aberta, já na região fechada. Para que o valor de resistência seja admissível, deve-se então calculá-lo como com base nos nodos antecessores dos nodos abertos da frente em questão, exceto quando o nodo aberto for o próprio destino, quando o valor de resistência é zero;
- Para calcular Ω_s a cada operação na lista de nodos *m* abertos, seria necessário que a lista estivesse ordenada não somente por *f(m),* mas também por *g(m) – k(m,s)*. Isto poderia ser implementado em duas filas de prioridade com referências para os elementos correspondentes. Acontece que uma fila de prioridade implementada com *Heap*, que tem menor complexidade, não admite operações que não sejam inclusão e tomada do primeiro elemento, e, portanto inviabiliza a operação conjunta com outra fila ordenada por outro critério;

Apesar da aparente complexidade no uso da resistência, sua implementação pode ser mais simples do que se imagina. Ocorre que somando ou não algum valor particular à função *h* de todos os nodos *m* de uma lista de abertos, isto em nada muda a ordem com que os nodos se encontram nela. Na verdade, o valor absoluto de *f* somente é utilizado pelo algoritmo nas condições de término, quando é comparado com L_{min}, que é o custo do menor caminho encontrado até o momento, para poda. Então, basta substituir as condições de término e poda comparando-se os valores de *f* com $L_{min} - \Omega_t$. Isto resolve o problema de atualização da ordenação dos nodos abertos. Resta ainda o problema de calcular Ω_t sem usar mais tempo ou mais memória. Ora, antes de ser encontrado um caminho, L_{min} não tem valor definido (corresponde a infinito), e nenhuma poda é feita. Somente após um caminho ser encontrado, deve-se computar os valores de resistência e penalidade. Em vez de se manter as listas de abertos duplamente ordenadas, é possível consultá-las em pequenos intervalos para computar os novos valores. Estes intervalos devem ser proporcionais aos tamanhos destas listas, para que, a cada atualização, contribuam significativamente para o aumento de Ω e *P*. Este processo mantém a complexidade temporal aproximadamente linear, de forma amortizada, e não interfere

na implementação das listas.

A estimação dinâmica, o cálculo de resistência (*min idea*), penalidade (*max idea*), e suas propriedades já haviam sido identificadas em [KAI 96], mas não foram empregadas em um algoritmo bidirecional genérico da mesma forma que aqui, somente em pesquisa por perímetro com algoritmos A* e IDA* ([KOR 93]) executados em seqüência, não simultaneamente. A execução em seqüência pressupõe decidir quando a primeira pesquisa deve parar. Para que esta decisão fosse tomada no momento certo, seria necessário conhecer o tamanho do caminho, o que é uma incógnita na maioria dos problemas. Erros nessa estimativa podem comprometer o desempenho do processo.

O método de pesquisas em seqüência, ou por perímetro, entretanto, é especialmente adequado à limitação de memória, como explorado em [KAI 97], onde é classificado como método não tradicional de pesquisa bidirecional heurística. Na prática, isto significa que se o espaço é de tamanho tal que as árvores de pesquisa não caibam em memória, executa-se a primeira pesquisa de forma que ocupe quase toda a memória e a segunda pesquisa com um algoritmo de profundidade do tipo IDA*[6]. Já em aplicações como roteamento, onde em geral é necessário que o grafo todo esteja em memória, esta vantagem não existe, e o principal objetivo ainda é obter o menor número de nodos visitados. Neste tipo de aplicação, LCS* apresentará os melhores resultados, independendo da estimativa do tamanho do caminho ótimo.

Kaindl também não observou que os valores de resistência e penalidade podem (e devem) ser usados em conjunto. O valor de penalidade pode ser menor que a própria estimativa plana quando o nodo sendo expandido está mais distante da origem do que o destino está. Então, o valor a ser usado com a função *h* deve ser o máximo entre os computados com resistência e com penalidade.

4.6.2 Critério de escolha de direção

Como critério de escolha de direção, se usa inicialmente o princípio da cardinalidade. Mas com vistas às aplicações em roteamento, deve-se prever a ocorrência de muitos empates críticos em áreas vazias. Nestes casos, A* usando a regra de empates já descrita é absolutamente ótimo, pois expande somente os nodos de um dos caminhos ótimos. LCS* expandiria duas vezes, já que é bidirecional e pode tomar caminhos ótimos diferentes. Assim acrescenta-se ao critério de escolha de direção uma condição pela qual, se uma frente está progredindo com estimações perfeitas (os novos valores de *f* gerados são sempre os mesmos), então LCS* mantém a expansão nesta frente ao invés de alternar ou usar a cardinalidade para decisão.

4.6.3 Implementação genérica de LCS*

Foi feita uma implementação genérica do algoritmo em C++ como uma classe *template*, a qual pode operar sobre qualquer domínio de grafo que tenha uma interface adequada para nodos, arcos e iteradores. A implementação usa uma *binary heap* para

[6] O algoritmo IDA* [KOR 93] emprega pesquisa heurística do tipo A*, mas dividida por estágios, sendo que nos estágios são feitas pesquisas DFS em vez de BFS, cada qual com uma profundidade limite maior. Como em outras pesquisas DFS, não é necessário armazenar listas de nodos abertos ou fechados, sendo econômico em termos de memória (linear em relação ao tamanho do caminho), embora gaste mais tempo.

manter as listas de nodos ordenados, oferecendo a operação de tomar o menor elemento em tempo constante, inserção em tempo *O(log n)* no pior caso, e tendo iteradores que permitem que a lista seja percorrida para computar os valores de resistência e penalidade. A atualização destes valores é feita após ser expandido um número de nodos igual a duas vezes a soma dos tamanhos das listas de abertos.

4.6.4 Pesquisa com LCS* em Grafos Genéricos

Grafos aleatórios e genéricos para testar o algoritmo LCS* foram gerados em um plano pela seleção de pontos arbitrários neste espaço. A distância em linha reta entre os pontos é usada como custo de um arco entre eles, se houver, e também como função de estimativa. Os grafos **aleatórios** são aqueles que têm apenas um arco por nodo, conectando-o a um outro nodo qualquer do grafo, de forma que possuem uma estrutura bastante irregular (Fig. 4.12a). Já os grafos **geométricos** são aqueles nos quais há mais arcos, mas estes estão limitados a existir entre pontos que se encontrem a uma distância máxima uns dos outros, conforme mostra a Fig. 4.12b.

a) Grafos Aleatórios b) Grafos Geométricos

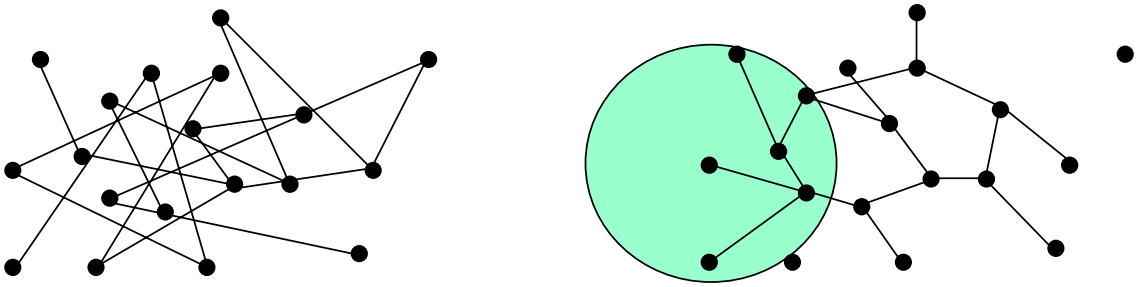


FIGURA 4.12 – Grafos aleatórios e geométricos em um plano.

A Tab. 4.2 mostra o número de nodos expandidos pelos algoritmos LCS* e A* a partir de origem e destino, sendo depois classificados em pior e melhor escolha para o A*, em um grafo aleatório. O grafo contém 70 nodos, e foram pesquisados todos os caminhos possíveis (70 * 69), de forma que este experimento demonstra a superioridade de LCS* sobre o universo de problemas. Grafos aleatórios podem não ser completamente conectados, e por isto há caminhos não existentes. A segunda coluna mostra que LCS* ainda é superior ao desempenho médio de A* se estes caminhos forem desconsiderados.

TABELA 4.2 – Nodos expandidos por A* e LCS* em grafos aleatórios.

Algoritimo	aleatório	solúveis
média A*	126156	78306
melhor A*	71576	61016
pior A*	180360	95596
LCS*	77710	59869

Já em grafos geométricos, LCS* não apresenta uma clara superioridade. A Tab. 4.3 mostra um grafo com 70 nodos e 10% de probabilidade de haver uma aresta entre um par de nodos que se encontre a uma distância máxima correspondente a 0.3 vezes o tamanho do plano. Quando este grafo é alterado para que haja ao menos um arco por

nodo, eliminando a presença de nodos isolados, LCS* perde para A*. Grafos Geométricos com custos de arcos iguais a distância permite estimativas muito precisas, e por esta razão o algoritmo A* visita praticamente somente os nodos no caminho ótimo, não podendo ser superado. Já o LCS* precisa expandir alguns nodos a mais para garantir suas condições de término, e por isto apresenta menor desempenho no universo de caminhos deste grafo.

TABELA 4.3 – Nodos expandidos por A* e LCS* em grafos geométricos.

Algoritimo	geométrico	com mais um arco
média A*	39095	28406
melhor A*	24438	21428
pior A*	53752	35384
LCS*	38662	34130

4.6.5 Pesquisa com LCS* em labirintos

Alguns domínios possuem características bem peculiares que dificilmente são imitadas pela homogeneidade de grafos genéricos. Para considerar tais exemplos, foram gerados conjuntos de labirintos, usando pesquisas aleatórias que produzem árvores embutidas em uma área quadrada. A Fig. 4.13 mostra pequenos exemplos de diferentes tipos de labirintos gerados com pesquisa em largura, em profundidade simples, e em profundidade com reinício aleatório, os quais possuem diferentes graus de dificuldade.

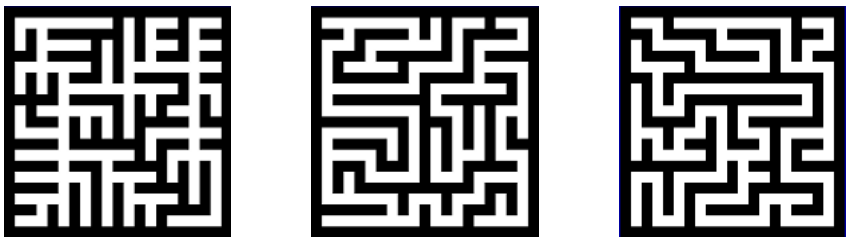


FIGURA 4.13 – Labirintos gerados com *BFS*, *DFS* e Multi-*DFS*.

A Tab. 4.4 mostra as razões entre o número de nodos expandidos por LCS* e A*, e entre seus tempos de execução. Neste domínio, para quaisquer dois nodos, existe um único caminho entre eles, e o problema é encontrá-lo com o menor esforço. Os grafos com uma pesquisa *DFS* são os mais simples, pois suas ramificações são curtas, e por esta razão o desempenho de A* e LCS* é semelhante, mas LCS* é claramente superior nos labirintos mais difíceis. Outros labirintos em que LCS* supera muito A* aparecem no anexo 3, Tab. 2.

TABELA 4.4 – Razões entre LCS e A* nos diferentes tipos de labirintos.

Algoritmo	Tempo	Nodos
BFS	0.698	0.661
DFS	1.035	0.975
Multi-DFS	0.612	0.595

4.6.6 Pesquisa com LCS* em grades 2D

Aqui são apresentados resultados de LCS* e A* em uma grade bidimensional, um grafo regular de grau máximo 4, tendo custos associados a cada nodo. O custo do arco entre um nodo e seu vizinho é a média dos custos de ambos. Os custos são atribuídos através de um preenchimento aleatório segundo 4 parâmetros: média, variação, mínimo e máximo. A média representa o valor básico do custo. Para cada nodo, é então gerado um valor aleatório dentro da variação dada, centrado em 0. Este valor é adicionado à média. Se o resultado for menor do que o mínimo, então o custo mínimo é usado. Se o resultado é maior do que o máximo, um valor representando infinito é usado, pelo qual a célula é considerada como inacessível. Desta forma, gera-se valores dentro do intervalo compreendido pelos valores mínimo e máximo, e cuja probabilidade de ocorrência dos extremos é determinada pela média e pelo valor de variação, como na Fig. 4.14. Em todos os exemplos, a função de estimação do espaço usa a distância cartesiana multiplicada pelo menor valor de custo assinalado a um nodo da grade.

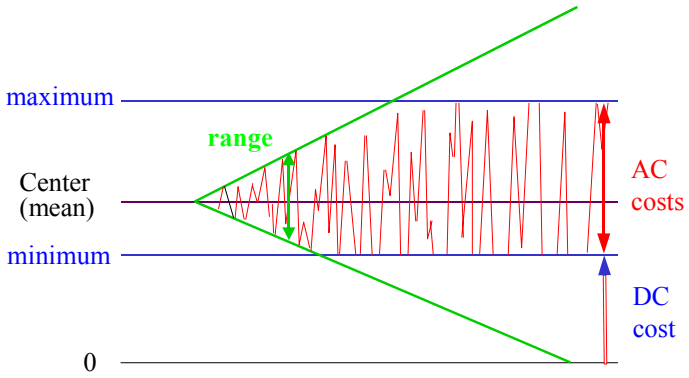


FIGURA 4.14 – Parâmetros para geração de exemplos aleatórios.

Estes parâmetros permitem gerar exemplos aleatórios representativos de situações práticas, mas não são nem muito genéricos e nem muito específicos. Grafos em geral podem ter grau e fator de ramificação bem maiores, mas nestes experimentos o ganho potencial é limitado pelo grau 4. Se uma grade de três dimensões é usada para representar roteamento, não só o grau se torna maior, como a função custo se torna mais complicada, e ganhos maiores podem ocorrer. Entretanto, esta grade representa bem as aplicações de roteamento em duas camadas ou roteamento global, e pode-se ter idéias realistas do resultado que será obtido nestes casos. Em todos os casos conta-se o número total de nodos expandidos em 20 caminhos com origem e destino aleatórios dentro de cada uma das grades. O algoritmo A* é executado duas vezes, uma partindo da origem e outra partindo do destino, já que é sensível a esta escolha. Deve-se comparar o resultado de LCS* com a média de ambas, mas para fins de desenvolvimento, compara-se com o melhor resultado de A*, já que indica que a pesquisa que parte de um lado é mais promissora do que a outra.

A Fig. 4.15 mostra um amplo espectro de custos, desde intervalo 0, quando as heurísticas são perfeitas, passando por uma região de custos bem distribuídos, até atingir um espaço em que há bastante saturação e espaços vazios em igual proporção.

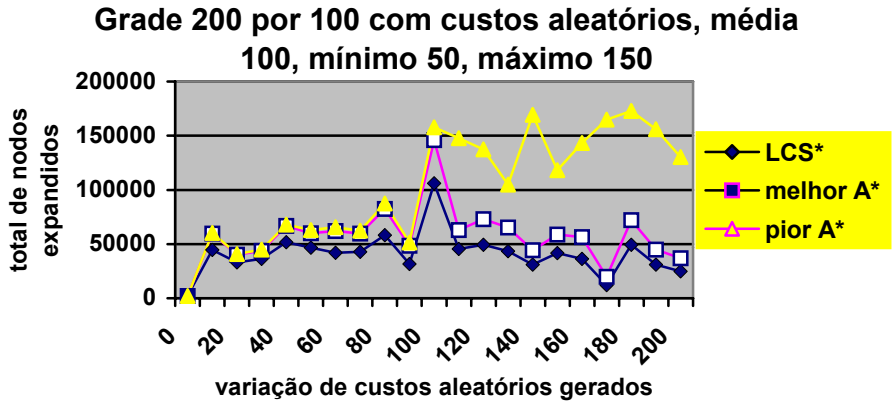


FIGURA 4.15 – Regimes de custos genéricos: perfeição, distribuição e saturação.

As figuras mostram o resultado da execução de muitas pesquisas. Apesar do bom desempenho médio do LCS*, há instâncias patológicas em que ele perde para ambas as possibilidades de A*, as quais devem ser analisadas. Esta perda se dá pela existência de muitos empates críticos. O caso mais típico encontrado é bastante interessante, e uma análise detalhada deste pode conduzir a um melhor critério de escolha de direção. Acontece que um empate crítico depende do valor de *f*, uma estimativa, e, embora esteja associado à seqüência de custos do grafo, depende da direção de pesquisa. Quando se tem custos e estimativas iguais (perfeitas) nas duas regiões que envolvem origem e destino, mas existe uma região intermediária com custos maiores, a seqüência de estimativas perfeitas só é reconhecida como crítica por uma determinada frente quando está próxima ao destino. Desta forma, o A* reconhece esta seqüência em ambas as direções, mas o LCS* não, já que cada frente sua só faz a primeira metade do caminho.

4.6.7 Tempos reais de execução

A Fig. 4.16 mostra a razão entre LCS* e A* considerando número de nodos expandidos e tempo real de execução na CPU, medido com funções dentro do código. Observa-se nestes exemplos uma incrível coerência entre essas duas medidas, e também o fenômeno de que às vezes LCS* ganha mais em tempo do que em número de nodos. Este comportamento pode parecer anômalo, já que o algoritmo LCS* executa mais operações por nodo expandido do que o A*. Entretanto, o tempo de expansão é dominado pelos tempos de inserção de nodos nas listas de abertos, e cada lista de uma frente do LCS* é menor do que a lista única mantida pelo A*, unidirecional.

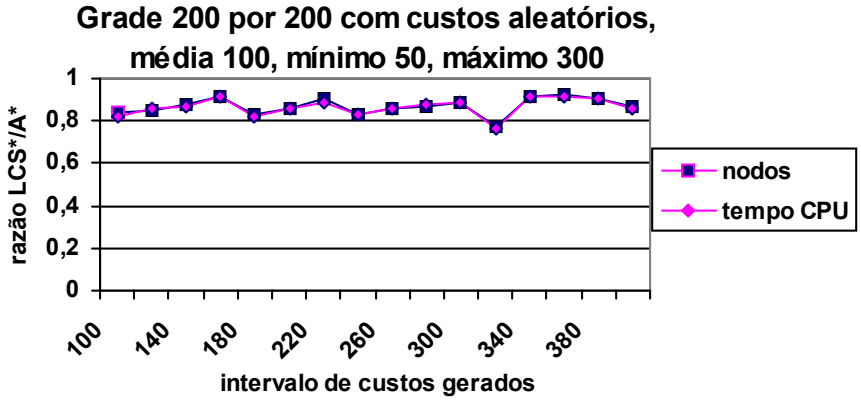


FIGURA 4.16 – Coerência entre número de nodos expandidos e tempo de CPU.

4.6.8 Pesquisa com ε-admissibilidade

A identificação do LCS* como algoritmo admissível é importante do ponto de vista teórico. Na prática, porém, geralmente não é necessário encontrar o caminho absolutamente ótimo, mas pode ser aceitável encontrar um dos melhores caminhos, se soubermos que ele está dentro de uma margem de erro satisfatória em relação ao caminho ótimo. Para tal, usa-se um método de pesquisa com um fator de admissibilidade ε entre 0 e 1 (1=100%). No algoritmo A*, isto é considerado dividindo-se os valores de *h* por ε. Isto provoca sobreestimação, e como resultado garante que o custo do caminho encontrado será no mínimo C^*/ε.

Esse método também poderia ser aplicado a LCS*, mas no caso do algoritmo LCS*, bidirecional, existe outra forma de computar a admissibilidade. Usa-se a mesma função de estimação, mas, uma vez que foi encontrado o primeiro caminho, compara-se o menor valor de *f* de cada frente com o valor de L_{min} encontrado até então. Já que os valores de *f* somente aumentam, é impossível encontrar um valor de L_{min} menor que os *f* presentes, e pode-se calcular então o fator de admissibilidade de que dispomos, comparando-o com um limite pré determinado. Este método é mais eficiente do que o usado no A* quando a admissibilidade desejada está entre 90 e 100%, conforme mostra a Fig. 4.17. Observa-se nesta Fig. também que este método não é eficiente quando o intervalo de custos é pequeno (na esquerda), pois a função sobreestimada no A* age em toda a pesquisa, enquanto que o cálculo de ε. por L_{min} somente poda os últimos nodos, sendo adequado em pesquisas mais difíceis.

Convém citar também que em diversos domínios a margem de erro quanto ao caminho ótimo não é importante, desde que se o encontre rapidamente. Diversos outros algoritmos e heurísticas podem ser empregadas para encontrar caminhos mais rapidamente, sem garantia qualquer de sua qualidade. Estes não devem ser desprezados, já que, se encontram o caminho rapidamente, provavelmente não é um dos caminhos mais longos em termos de número de nodos. Mesmo em roteamento, se os custos são mais ou menos bem distribuídos, tais algoritmos podem dar resultados muito bons.

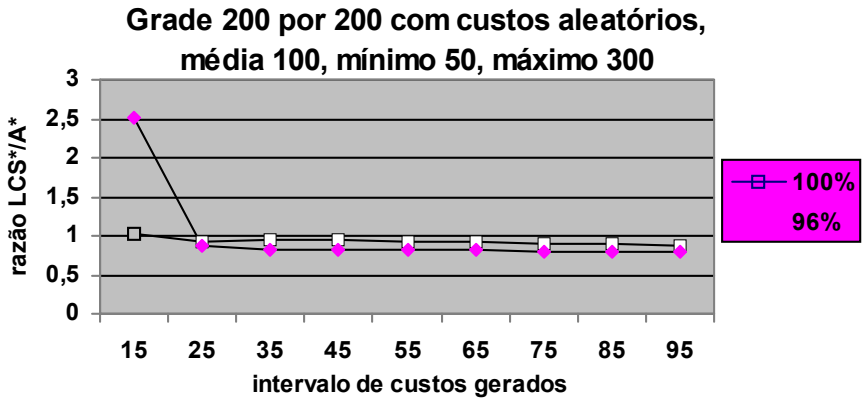


FIGURA 4.17 – Efeito da pesquisa com ε-admissibilidade nos algoritmos.

4.6.9 Dificuldade da pesquisa

Em todos os domínios, foi identificado que LCS* tem ganhos significativos quando trata problemas com heurísticas pobres. Isto significa que em domínios onde a qualidade da heurística varia, LCS* sempre ganha nos problemas mais difíceis, e A* nos mais fáceis. Assim, a economia de tempo do LCS* facilmente compensa sua perda. Para quantificar esta afirmação, observe-se as Tab. 4.5 e 4.6. A primeira corresponde a problemas fáceis, onde A* ganha, mas poucos nodos são pesquisados. A Tab. foi gerada com custos mínimo 100, máximo 300, e centro igual a *100-intervalo/4*. Já a Tab. 4.5 corresponde a grades com centro 40, mínimo 20, máximo 300, e intervalo variando, que representam problemas bem mais difíceis. Estes exemplos servem para confirmar o fato de que LCS* sempre ganha nos problemas que exigem a expansão de muitos nodos, relativamente, em conseqüência de deficiência na função de estimação.

TABELA 4.5 – Número de nodos expandidos por A* e LCS* em grades fáceis.

intervalo	LCS*	melhor A*	pior A*	razão LCS/A* médio
5	12025	8949	23492	0,741369
15	52045	34299	55477	1,15944
25	44333	38099	53329	0,96979
35	40742	30799	58729	0,910151
45	31378	24126	44394	0,915879
55	71337	52542	83414	1,04941
65	42325	30098	60438	0,934987
75	65254	46097	91681	0,947234
85	50292	36225	61154	1,03292
95	63868	44733	72327	1,0912

TABELA 4.6 – Número de nodos expandidos por A* e LCS* em grades difíceis.

intervalo	LCS*	melhor A*	pior A*	razão LCS/A* médio
5	234104	224369	230062	1,03032
15	173956	177840	180861	0,969925
25	193740	217070	227038	0,87249
35	267162	294617	311479	0,881583
45	169381	186498	199426	0,877795
55	278125	307399	340856	0,858074
65	208926	226713	243328	0,888971
75	280964	315514	342046	0,854565

85	194213	207964	222377	0,902603
95	246533	277839	294893	0,860902

5 Roteamento com Algoritmos de Pesquisa

Os algoritmos de pesquisa aplicados a roteamento de circuitos surgiram a partir de [LEE 61], onde é apresentado um algoritmo *BFS* básico, com o conceito de marcação. Já que permitem encontrar caminhos em labirintos formados por obstruções, estes algoritmos passaram a ser chamados em *EDA* de *maze runners* ou *maze routers*. Para roteamento detalhado ou global com estes algoritmos, o grafo é uma representação abstrata do espaço que todas as conexões devem compartilhar. Na maioria dos casos este grafo será uma grade, estando cada nodo associado à uma coordenada física, como o cruzamento de trilhas verticais e horizontais (roteamento detalhado), ou a um conjunto destas posições (roteamento global de área), conforme ilustrado na Fig. 2.11. Este grafo não representa as redes e suas árvores de conexão, e os algoritmos de pesquisa se concentram unicamente em encontrar um caminho para unir duas partes de uma rede até então separadas. A formação das redes como árvores é um problema anterior, mas pode ser feito automaticamente pelos algoritmos de pesquisa de caminhos quando estes permitem múltiplos pontos de origem e destino, conforme será visto adiante.

A presença de várias camadas de roteamento é facilmente aceita por algoritmos de pesquisa. O roteamento de um circuito plano com duas camadas ainda pode ser feito em uma grade de apenas duas dimensões se a direção das conexões é estritamente relacionada às camadas. Assim é implementado o sistema MARTE. Entretanto, com mais de duas camadas é necessário implementar uma grade de três dimensões para roteamento detalhado, a qual é também atraente para roteamento global, embora não necessária. Roteamento global com uma grande de três dimensões fará ao mesmo tempo a distribuição das conexões sobre a área e o assinalamento das camadas corretas para cada uma delas. Para isto, necessitará de um modelo de custos adequado, do qual o desempenho do sistema dependerá.

5.1 Roteamento detalhado com *maze router*

O algoritmo de Lee é composto de três fases: expansão, traçado e reinicialização. A fase de traçado tem um custo computacional bem pequeno se comparada com as demais. Já a fase de reinicialização, praticamente ignorada em IA, é de grande importância em *EDA*. Ela consiste em apagar da grade as informações anotadas pela pesquisa anterior, para que a próxima ocorra corretamente. Pode-se evitar o acesso a todas as posições da grade controlando a área abrangida pela pesquisa imediatamente anterior, mas mesmo assim é uma tarefa de alto custo. Isto pode ser evitado se a marcação incluir um código de senha diferente para cada pesquisa. Assim, a grade só precisa ser apagada após todos os valores de senha terem sido utilizados.

Em conseqüência da importância de fazer roteamento detalhado com algoritmos de pesquisa, grande ênfase foi dada em reduzir a quantidade de memória necessária para cada posição da grade. A partir do processo inicial de marcação com uma seqüência de números inteiros (correspondente à função g com custos unitários), foram sendo utilizadas novas técnicas, como apenas anotar um ponteiro para o nodo antecessor. Ao final se obteve a mínima quantidade de informação necessária por posição, que é um *bit*. Conforme proposto por [AKE 67], a seqüência de marcação com bits duplicados

"0011001100..." permite distinguir o antecessor do sucessor em *BFS*. Em [JOH 95] mostra-se que esta mesma seqüência não pode ser usada com origens múltiplas (Fig. 8a do anexo 2), pois sua reversão não é identificada, mas uma seqüência maior de valores duplicados pode: "0011223300112233", a qual somente é mais econômica do que manter ponteiros para os vizinhos quando estes são mais de 4 (Fig. 8b do anexo 2).

Como mencionado no Cap. 2, outra principal desvantagem dos *maze routers* é a quantidade de tempo necessária para sua execução, já que visitam uma grande área na grade para cada conexão sendo feita. A Fig. 5.1 mostra o tempo de CPU em função da área do circuito para realizar o roteamento detalhado com um algoritmo de Lee puro, implementado no sistema Marte. Para regimes bem definidos de custos e sem a presença de muitos obstáculos, o tempo de processamento é quadrático em relação ao comprimento total de conexões, como mostra a Fig. 5.2.

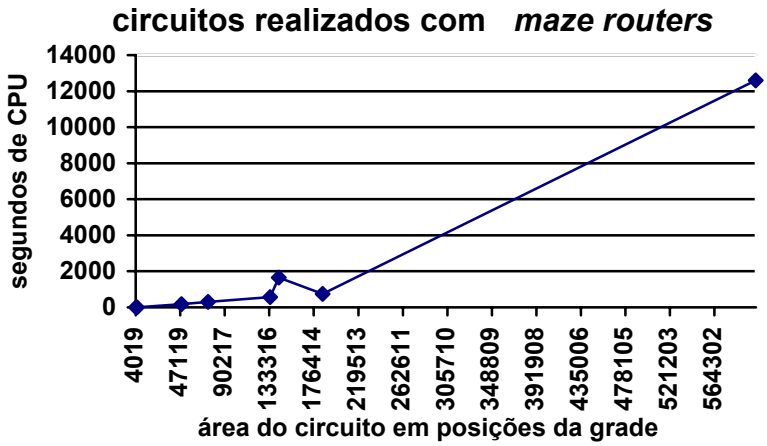


FIGURA 5.1 – Tempo de CPU em relação à área do circuito.

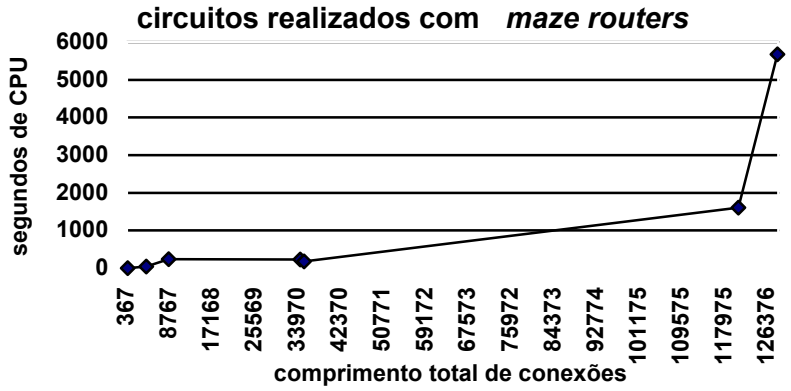


FIGURA 5.2 – Tempo de CPU em relação ao comprimento das conexões.

Para diminuir a quantidade de nodos expandidos, há trabalhos usando pesquisa em profundidade ([PRE 88] e [SOU 78]), distância para o destino à semelhança de A* ([RUB 74]), e limitações da área pesquisada em geral. Em especial, destaca-se a técnica de limitação em forma de "L", apresentada em [TAD 80]. São criados canais em forma de "L" e processadas todas as redes do circuito. Muitas não conseguem ser roteadas no espaço definido, e então são consideradas novamente em uma próxima passagem onde o

tamanho deste canal é aumentado. Se executado com custos não uniformes, este processo corresponderia a uma pesquisa não admissível, pois exclui parte do grafo, mas em se tratando de roteamento detalhado sem custos, ao contrário, evita pesquisar caminhos com muitas voltas. Ao mesmo tempo em que expande menos nodos, esta técnica reduz o número de bloqueios entre conexões, posto que faz conexões mais retas inicialmente. O impacto do uso de canalização pode ser observado na Tab. 5.1.

TABELA 5.1 – Efeito do uso de canalização e direção livre no MARTE.

	circuito		*Maze router* básico		MARTE otimizado	
nome	área	redes	CPU (s)	% rot	CPU (s)	% rot
s28-2	72x80	79	4.0	97.46	6.0	98.73
s386-2	168x216	546	107.0	98.71	26.0	100
s510-2	168x288	590	189.0	97.45	48.0	99.66
c880-2	280x264	819	325.0	97.80	76.0	99.87
c499-2	350x384	990	631.0	99.19	68.0	99.89
s1494-2	350x408	1870	1379.0	96.89	425.0	98.12

Com o objetivo de evitar o congestionamento, diversos trabalhos utilizam custos não unitários, como [RUB 74] [HIG 83] e [KOR 82]. Aumentando o custo das posições vizinhas a uma conexão, mesmo em roteamento detalhado, faz com que as rotas evitem passar por regiões mais congestionadas, distribuindo melhor as conexões na área. Os regimes de custos aplicados a roteamento detalhado são bem restritos, como por exemplo, um valor entre 0 e 3, e não se encontra muitos dados comparando estes resultados. Considerar custos implica na necessidade de ordenar as listas de nodos abertos, o que requer tempo $O(\log n)$ onde n é o tamanho das listas. Entretanto, com custos limitados a um pequeno intervalo, é perfeitamente viável implementar uma série de listas FIFO indexadas pelos valores de custo, o que resulta em tempo de inserção constante.

Realizar o roteamento detalhado sem custos ou com poucos valores de custos é mais rápido que realizar o roteamento global do mesmo circuito, já que o roteamento global deve considerar muitos valores de custo para modelar obstruções e congestionamento, e exige implementação das listas ordenadas de nodos abertos. Por exemplo, o circuito c499 requer apenas 12 segundos de CPU para ter seu roteamento detalhado feito diretamente pelo sistema Marte e entre 21 a 27 segundos para ter seu roteamento global definido pelo algoritmo LCS*, na mesma máquina. A variação se deve ao uso de diferentes esquemas de custo, e usa-se uma grade de 3 dimensões mas com apenas duas camadas, e onde cada célula global corresponde a uma área de 10 por 10 trilhas no roteamento detalhado. Isto leva a conclusão de que, embora os *maze routers* sofram o problema de complexidade, este problema é maior em roteamento global, apesar do grafo ser muito menor. Por esta razão, é importante o uso de pesquisa heurística para diminuir a área visitada, bem como de novos algoritmos, sejam eles de pesquisa, como o LCS*, sejam de outra natureza.

5.2 Roteamento com o Algoritmo LCS*

O algoritmo LCS* é mais eficiente para pesquisa de caminhos ótimos em grafos em geral. Estes caminhos são importantes para roteamento, seja pelo melhor

desempenho que algumas conexões requerem, seja pelo fato de se poder tratá-las uma a uma. O algoritmo também dispensa manter muitas informações nos nodos do grafo, o que também é benéfico para roteamento. Apesar disto, apenas prover ganhos constantes (limitados pela estrutura do grafo) em tempo e memória não é um fator muito apelativo quando as principais preocupações são pela obtenção de rotas de melhor qualidade e maior convergência no processo. Assim, a aplicação de LCS* para roteamento deve incluir outros tipos de otimização e particularidades.

Em todas os experimentos, busca-se comparar a eficiência de diferentes algoritmos, medida como número de operações (de expansão), ou tempo de CPU. A qualidade do roteamento, representada pelo custo de cada conexão, **é exatamente a mesma em todos estes algoritmos de pesquisa**, BFS (Lee), A* ou LCS*, visto que são todos admissíveis. Apenas a forma de atribuir os custos pode influenciar nesta qualidade.

5.2.1 Pesquisa de caminhos em grades 2D

A Fig. 5.3 apresenta resultados de pesquisa de caminhos aleatórios em uma grade 2D com custos bem distribuídos e com mais saturação, mas tendo maior proporção de espaços vazios, tentando modelar melhor uma situação intermediária de roteamento com custos. Observa-se facilmente que o algoritmo LCS* ganha não só da média, mas da melhor possibilidade de A* em todos estes regimes. Para aplicações de roteamento detalhado sem custo, o que ocorre é que ou os nodos terão um custo (fixo), representando o comprimento da conexão, ou serão inacessíveis. As Figs. 5.4 e 5.5 usam valores mínimo e máximo iguais para reproduzir este comportamento, enquanto variam a média do intervalo gerado para produzir diferentes probabilidades de obstruções.

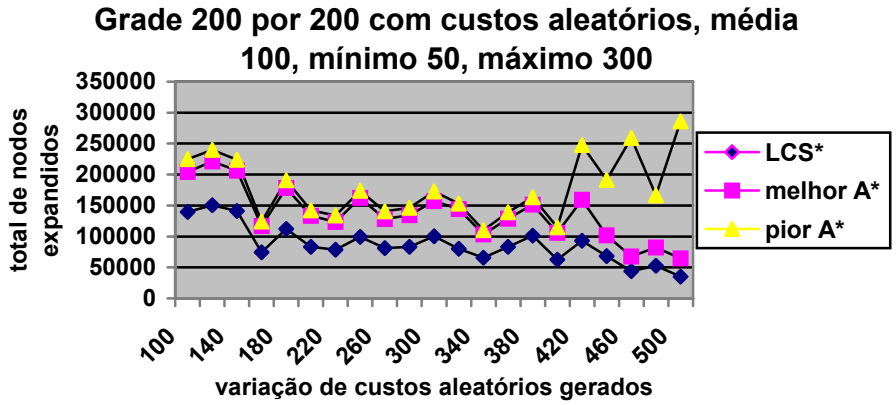


FIGURA 5.3 – Regimes de custos genéricos com mais espaços vazios.

Observa-se que o algoritmo LCS* se mantém mais eficiente do que a média e o melhor dos A*. A Fig. 5.5 torna evidente que quando há pouca probabilidade de obstrução os caminhos são curtos e encontrados com pouco esforço, ao passo que se o destino está isolado da origem, todo o espaço de um dos lados precisa ser expandido. Assim, LCS* se comporta como o melhor dos A*. A Fig. 5.5 mostra resultados médios com mais exemplos na região de transição entre o regime de caminhos ótimos e o regime de caminhos não existentes.

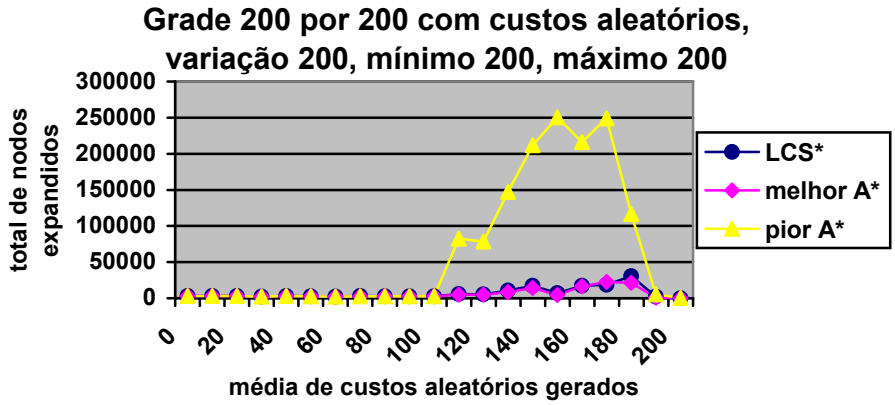


FIGURA 5.4 – Regimes de custos fixos com obstáculos.

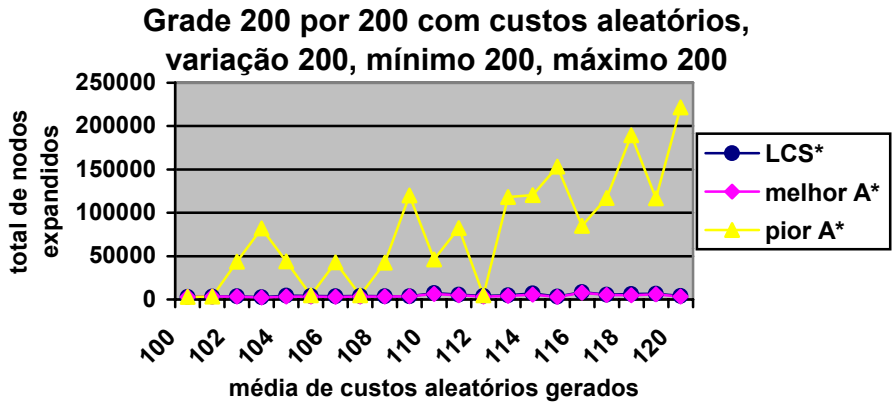


FIGURA 5.5 – Transição de regimes de custos fixos para saturação.

Em aplicações de roteamento, casos em que não haja mais caminho devem ser claramente evitados. Isto significa que o regime de custos das Figs. 5.4 e 5.5 não deve acontecer em um sistema bem sintonizado. Em vez deste, deve haver um regime de custos direcionados por camada, pelo qual as situações mais saturadas serão um pouco semelhantes a labirintos, mas não terão muitos caminhos impossíveis. A Fig. 5.6 mostra uma situação extrema de regime de custos semelhante a roteamento, com muitos espaços e bloqueios como nos exemplos da Fig. 5.5, mas sem considerar aqueles para os quais não havia solução. Este tipo de problema, artificial, deve representar as piores situações práticas de custos em grades. Observa-se então, que neste regime o algoritmo LCS* é comparável à média dos A*, mas perde para o melhor deles.

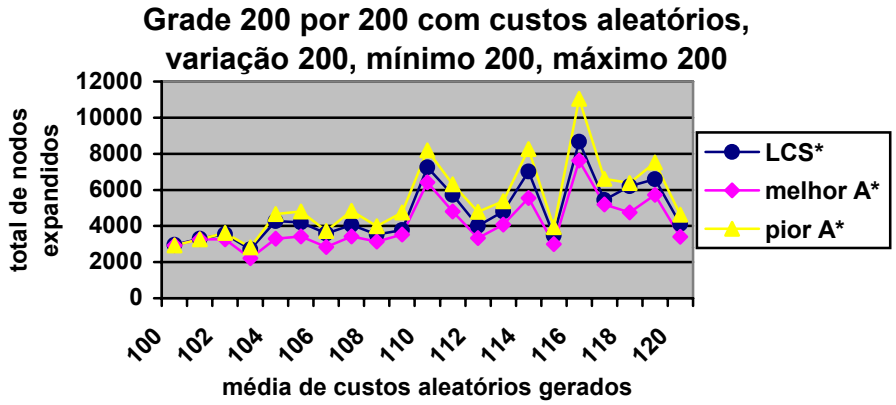


FIGURA 5.6 – Regime artificial de custos fixos sem bloqueios completos.

5.2.2 Pesquisa de caminhos em grades 3D

Para realizar o roteamento global de circuitos com o novo algoritmo LCS*, foi implementado uma ferramenta que modela o espaço como uma grade 3D, as redes do circuito com seus pinos, grupos e rotas, e utiliza um novo modelo de custos flexível, detalhado a seguir. Neste ambiente foram executados diversos testes com os algoritmos A* e LCS* e circuitos reais posicionados para matrizes Marcela [GUN 95]. Alguns destes resultados se encontram em [HEN 2000], e comprovam os mesmos efeitos já observados quanto ao desempenho dos algoritmos. A distribuição de posições de redes reais em vez de pontos origem e destino aleatórios não influencia no desempenho dos algoritmos, e a variação dos custos permanece sendo a principal questão. A Fig. 5.7 mostra o número de nodos expandidos por ambos os algoritmos para roteamento do circuito c499 com diferentes variações de custo, definidos de forma semelhante aos exemplos de grade 2D. Observa-se que quando a função heurística é eficiente, o algoritmo A* expande menos nodos, e que o algoritmo LCS* continua sendo vitorioso nas instâncias mais difíceis. Os tempos de execução também permanecem proporcionais, como demonstra a Fig. 5.8.

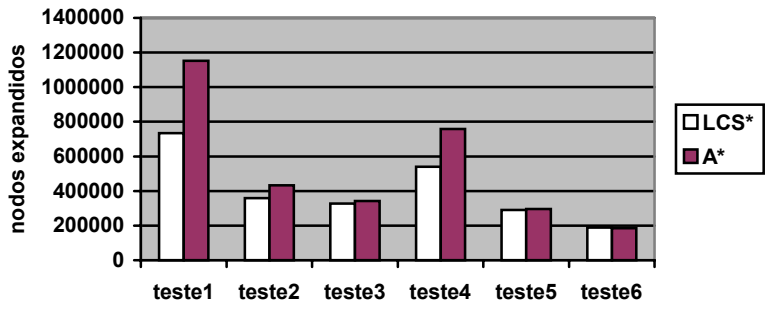


FIGURA 5.7 – Roteamento do circuito c499 com diferentes regimes de custo.

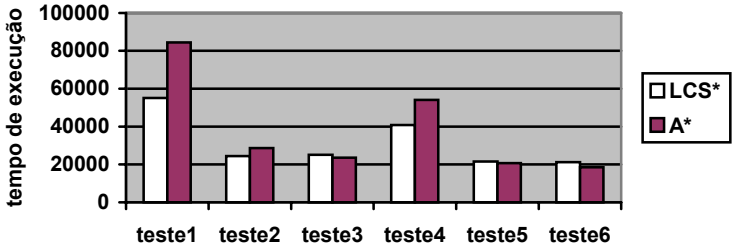


FIGURA 5.8 – Tempos de execução para roteamento do circuito c499.

5.3 Pesquisa com múltiplos destinos

Como mencionado anteriormente, a existência de múltiplas posições de destino é facilmente assimilada pelo algoritmo, mas muda a estrutura do grafo. A estrutura deste é capturada pela função de estimação, que, neste caso, é prejudicada. Para calcular as estimativas na presença de múltiplos destinos há quatro possibilidades. A primeira, tradicional, é calcular a distância do nodo até o retângulo que envolve todos os destinos (Fig. 5.9a). Esta estimativa se torna péssima quando o nodo está dentro deste retângulo, o que ocorrerá com freqüência em roteamento.

A segunda possibilidade, proposta por Caldwell em [CAL 96], está em calcular a distância para cada um dos destinos e escolher o valor do mais próximo (Fig. 5.9b). Ela exige um tempo de computação linear em relação ao número de destinos, mas provê estimativas muito melhores. Se os destinos são somente outros pinos da rede, o seu número é normalmente bastante limitado, e isto não implica em perda de desempenho. Se, entretanto, caminhos inteiros já encontrados forem usados como destino, o que é comum em roteamento, o número de pontos é grande, e pode não compensar verificar todos. Para evitar isto, pode-se reduzir o número de computações pelo cálculo de distância entre ponto e reta. Assim, mesmo quando o destino é uma rota já traçada, considera-se somente seus trechos retos e calcula-se eficientemente o mais próximo.

Neste trabalho, se propõe uma extensão a esta técnica, que é o direcionamento ao destino mais próximo, a qual tem admissibilidade garantida. Pode-se demonstrar que se um nodo *n* está mais próximo de um destino t_x, então, sempre que um nodo *n'* filho de *n* for gerado, se *h(n') < h(n)* então *n'* está também mais próximo de t_x do que de qualquer outro destino t_i. Assim pode-se definir intervalos na grade nos quais não é necessário recalcular qual dos destinos é mais próximo, como na Fig. 5.9c. Nesta figura, os nodos gerados que exijem recalcular o destino mais próximo são aproximadamente os do triângulo hachurado, enquanto que os dos dois retângulos hachurados não precisam. A figura. ilustra relativamente a partir do nodo que aparece no centro, mas o mesmo vale desde a origem *s*.

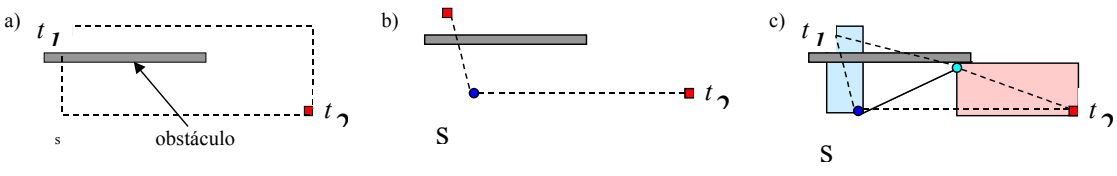


FIGURA 5.9 – Diferentes formas de calcular o destino.

5.4 Formação de redes

A pesquisa com múltiplos destinos será usada para conectar um pino fonte (*driver*) a um conjunto de pinos de entrada de portas (*sinks*). Após ter sido encontrada a primeira conexão, é necessário unir dois conjuntos de pontos entre si. Assim, o problema é, genericamente, o roteamento entre múltiplas origens e múltiplos destinos. Isto não exige nenhuma outra adaptação do algoritmo. Acontece, entretanto, que a forma das redes de múltiplos terminais é muito importante, e depende não somente da posição mas da ordem com que são executadas cada uma das pesquisas parciais que agrega um novo destino à rede. Árvores mínimas são desejadas para minimizar o comprimento total de conexões, e árvores com caminhos mínimos devem ser geradas para as conexões nas quais os caminhos desde a origem até cada um dos destinos devem ser equilibrados. Considera-se aqui apenas a formação de redes de conexões[7].

Um aspecto muito interessante dos algoritmos de pesquisa heurística é que permitem que custos arbitrários sejam atribuídos aos nodos vizinhos da origem, os quais representam as múltiplas origens reais no roteamento. Pois então, artificialmente manipulando estes valores, pode-se gerar redes que variam desde árvores mínimas até árvores de caminhos mínimos. Usa-se valores de *g* iguais a *0* para dar igual prioridade a todas as origens, ou valores incrementados a partir do *driver* real para dar maior prioridade às posições mais próximas da origem real, permitindo caminhos *driver-sink* melhores. A função *h* também pode ser inicialmente deformada, assumindo o valor *0* nas posições de origem para que estas tenham prioridade sobre outros caminhos de igual custo. Estas propostas também se encontram em [Cal96], mas não foram exploradas na literatura. Pode ser interessante também efetuar uma única pesquisa do *driver* para os *sinks*, terminando somente quando o último deles for encontrado. Este processo reconsidera a expansão de nodos intermediários para várias conexões em árvores de caminho mínimo, mas não permite operações de poda na pesquisa.

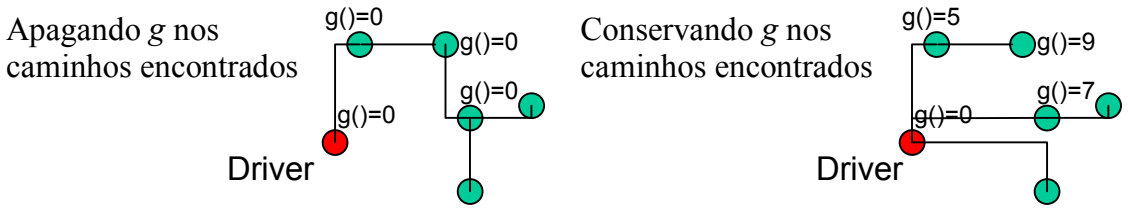


FIGURA 5.10 – Formação de árvores com controle de *g(n)*.

5.5 Modelo de custo

É o modelo de custos, entretanto, o que tem maior impacto tanto sobre o desempenho do algoritmo quando sobre a qualidade dos caminhos que encontra. Estes modelos devem ser melhor compreendidos, à luz de critérios exatos, mas são muitos, e com muitos objetivos diferentes. O custo de uma parte ou de um caminho completo

[7] Em geral, existe também o problema de formação de árvores com reforçadores (*buffers*), Estes problemas alteram a estrutura do circuito pela inserção de células, e são considerados externos aos algoritmos de roteamento aqui estudados. A inserção de reforçadores deve ser feita durante ou após o posicionamento, usando estimativas de roteamento. Mesmo nos casos em que este processo é realizado (para algumas conexões) durante o roteamento, os algoritmos aqui descritos serão utilizados como servos, desconhecendo o problema mais geral.

pode representar diversos fatores, os quais podem ser conflitantes, como, por exemplo:

- Comprimento da conexão;
- Quantidade de recursos necessários para sua implementação;
- Dificuldade de realizá-la pela presença de obstáculos;
- Congestionamento devido a outras conexões;
- Desempenho elétrico da conexão, em função de RC;

Observa-se que a quantidade de recursos aumenta com o comprimento das conexões, enquanto que o desempenho diminui proporcionalmente, ao passo que a dificuldade e congestionamento são fatores quase exclusivamente independentes. Somente não o são porque se rotas maiores são feitas, estas são influenciadas por mais obstáculos, e impõem congestionamento maior. Esta correlação de fatores fez com que tradicionalmente o modelo de custo fosse dado por um único valor, tendo uma parte constante e uma variável. A parte constante modela o comprimento das conexões em posições da grade, e, portanto, desempenho e quantidade de material. A parte variável modela a presença de obstáculos e congestionamento, sendo este modelo semelhante ao preenchimento aleatório usado nos testes aqui citados.

Entretanto, considerando as características de tecnologia de interconexões e a necessidade de precisão, é fácil perceber que custos que modelam estes fatores podem ser conflitantes entre si. Por exemplo, se uma determinada camada possui maior largura e espaçamento de trilhas para prover menor constante *RC*[8], uma rede roteada por ela tem maior custo de implementação e menor custo de desempenho. Pode-se previamente associar conexões de tipos diferentes (locais, semiglobais, e globais) a camadas diferentes, mas este pré assinalamento será um grau de liberdade a menos para os algoritmos que tratarão cada parte deste roteamento.

Uma grade de roteamento tipicamente retorna um único valor como custo do movimento entre duas posições adjacentes. Artigos que analisem detalhadamente modelos de custo são escassos. Geralmente os trabalhos de roteamento global apenas mencionam um determinado esquema de custos que foi implementado, comparando-o com outras alternativas que representam apenas pequenas variações, como funções de custo por dificuldade lineares ou quadráticas e sua influência na convergência, mas sem maiores mudanças na natureza de seu significado.

Um meio automático de fazer com que as conexões pequenas usem preferencialmente os níveis inferiores enquanto as conexões longas usam os superiores é associar um valor muito elevado de custo para as vias, e manter os demais custos mais econômicos nas camadas superiores. Assim rotas pequenas suportarão os custos um pouco maiores nas camadas inferiores, e somente quando a pesquisa se tornar grande, irá aceitar os custos de mudar de camada, pois a economia de custos da camada superior compensa esta perda. Este mecanismo é efetivo, mas apresenta um mau efeito colateral no processo de pesquisa: a deficiência da função heurística. Na pesquisa pelos caminhos longos, uma grande quantidade de área será pesquisada nos níveis inferiores até vencer a barreira imposta pelas vias. Neste caso, mesmo o algoritmo LCS* não terá um

[8] O atraso de um sinal em uma conexão é proporcional ao seu comprimento e à constante *RC* desta. A **constante *RC*** é determinada pelos valores de capacitância *C* e resistência *R* por unidade de distância. A alteração das larguras, alturas e espaçamento de conexões em uma determinada camada de roteamento permite definir menores constantes *RC* (ver Introdução).

desempenho satisfatório, já que ambos destino e origem devem vencer estas barreiras de custo.

Para resolver o problema de controle (e significado dos custos) e ao mesmo tempo evitar que a função heurística seja ruim, é proposto um modelo de custos diferenciados, que são considerados com um controle de pesos a cada pesquisa. Dividem-se os custos em três grupos: dificuldade, material e desempenho. O custo de dificuldade é armazenado em cada posição da grade, na forma de três valores para cada arco: obstáculo *obst*, demanda *demand* e tomado *taken*. Os custos *obst* são fixos, enquanto os outros podem ser atualizados durante o roteamento. A separação entre *demand* e *taken* serve a dois propósitos. Primeiro, permitir que o roteamento seja executado em diversas passagens a fim de distribuir o congestionamento. A cada passagem, o que foi assinalado na passagem anterior é considerado apenas como demanda, e *taken* é zerado e atualizado. A segunda finalidade está em usar ou não um limite de congestionamento. Os valores de *demand* são considerados sem limites, enquanto que *obst* mais *taken* mais a largura da conexão atual não podem superar a capacidade da *GRC*.

Já os custos de material e desempenho independem da posição em que se está na grade, e são associados às camadas em forma de tabelas. Assim, cada camada possui um custo w_{lay} de material, associado à largura das trilhas, e um custo t_{lay} associado ao desempenho da camada em termos de atraso, função de sua característica RC. Para prever que conexões de larguras diferentes usem uma mesma camada, cada conexão também deve ter valores característicos de custo w_{net} de material e t_{net} de atraso. Desta forma, a função de custo é definida pelas seguintes fórmulas e condições:

$$C_{dif} = (a_1\ obst + a_2\ demand + a_3\ taken) + w_{net};$$
$$C_{mat} = (w_{net} < w_{lay}?\ w_{lay} : w_{net})\ *\ b;$$
$$C_{per} = (w_{net} < w_{lay}?\ t_{lay} : t_{net})\ *\ c;$$
$$\textbf{If}\ (\ obst + taken + w_{net} > w_{available}\ )$$
$$\textbf{then}\ Cost = \infty$$
$$\textbf{else}\ Cost = (C_{dif} + C_{mat} + C_{per})\ /\ fator;$$

O uso deste modelo precisa ser sintonizado com muitos experimentos, mas é bastante flexível e permite um controle muito preciso dos objetivos e da eficiência da pesquisa. Os valores para w_{lay} e t_{lay} podem corresponder a valores reais de tamanho e atraso, mas isto não é importante neste modelo. Em operações de estimativa do desempenho de conexões sim, é indispensável que valores de atraso e tamanho correspondam à realidade. O modelo proposto tem a única finalidade de dar controlabilidade ao algoritmo de pesquisa, fazendo com que ele escolha rotas preferencialmente em uma ou outra camada. Assim, qualquer função côncava das camadas basta para que, na soma, pesada, se tenha uma camada com custo mínimo de movimento. A Tab. 5.2 mostra duas funções côncavas e como diferentes proporções de fatores *b* e *c* afetam o custo de movimento em cada camada, desconsiderada a componente de dificuldade.

Convém observar neste momento que, sendo o número de camadas limitado a menos de uma dezena, essa prioridade que se está implementando com a soma de funções côncavas tem um efeito restrito. Isto ocorre por que é preciso considerar ainda que as camadas têm direções restritas associadas, na maior parte das vezes. Por exemplo, no caso da Tab. 5.2, uma trecho de conexão em um sentido pode usar apenas

as camadas 1, 3 ou 5, e se estiver no outro sentido, apenas as camadas 2 ou 4. Isto significa que outras formas mais simples de pré-assinalamento de camadas podem ser empregadas para obter o mesmo efeito, ou com outros algoritmos, ou, ao menos, limitando o espaço no qual os algoritmos de pesquisa rodam. Este é um dos motivos pelos quais já se disse que a grade de roteamento não é verdadeiramente de três dimensões, e que roteamento em duas camadas ainda é grande parte do problema. De qualquer forma, o modelo aqui apresentado melhora o controle de algoritmos de pesquisa quando aplicados a múltiplas camadas, permitindo encontrar conexões que priorizem um ou outro critério.

TABELA 5.2 – Funções côncavas para w_{lay} e t_{lay} permitem controle.

camada	w_{lay}	t_{lay}	$b = c$	$b = 2c$	$b = 3c$	$b = 6c$
1	10	50	60	70	80	**110**
2	13	35	48	61	**74**	113
3	20	20	**40**	**60**	80	140
4	37	15	48	89	126	237
5	50	10	60	110	160	310

Quanto ao critério de dificuldade, na primeira etapa de roteamento, pode-se considerar apenas *obst* e atualizar apenas *demand*, fazendo uma estimativa de congestionamento. Em etapas subsequentes pode-se considerar os três valores de dificuldade, dando maior ou menor importância para a demanda, e mesmo para *taken*, desde que não ultrapasse o limite das *GRC*s $\boldsymbol{w_{available}}$. Conexões que não tenham satisfeito seus requisitos de atraso podem ser roteadas com maior peso *c* e menores pesos *a* e *b*. A maior vantagem do modelo, no entanto, está em permitir que este controle seja feito eficientemente, evitando pesquisas extensas. Isto é possível porque a cada pesquisa, os valores de $\boldsymbol{w_{net}}$ $\boldsymbol{t_{net}}$ $\boldsymbol{a_1}$ $\boldsymbol{a_2}$ $\boldsymbol{a_3}$ $\boldsymbol{b}$ e $\boldsymbol{c}$ são considerados constantes pelo algoritmo. Isto permite calcular um valor de c_{min} muito mais alto, conferindo maior poder de poda à função heurística, se comparado com o modelo anterior onde há custos altos para as vias e muito baixos para movimentos normais.

Espera-se que, em trabalhos futuros, se disponha de mais dados relacionando estes fatores ao esforço de pesquisa e à qualidade das conexões encontradas.

6 Roteamento com o Algoritmo LEGAL

Este Capítulo versa sobre um novo algoritmo, chamado LEGAL, cujo princípio de funcionamento pode ser aplicado em situações diversas. A principal motivação para seu desenvolvimento advém do problema de roteamento de área, caracterizado em seguida. Ao contrário dos algoritmos de roteamento rede a rede com pesquisa de caminhos, o algoritmo visto aqui não visa dar maior controle ou obter o roteamento mais otimizado para determinada conexão ou situação. O objetivo principal deste trabalho é o de estabelecer um método extremamente rápido para o roteamento de grandes áreas sem a necessidade de processos iterativos, de otimização combinatória, intensivos em CPU, mantendo, é claro, a qualidade geral da solução. Três exemplos de implementação são apresentados como indicativos de que estes objetivos podem ser atingidos.

6.1 Roteamento de Área

O roteamento de área (*area routing*) é de grande importância atualmente, devido à disponibilidade de várias camadas para roteamento. Estas camadas permitem a busca de circuitos com "*zero routing footprint*", através da eliminação dos espaços dedicados exclusivamente para roteamento. Segundo [SHE 95] (pág. 15), são usados algoritmos *greedy* e *maze* para endereçar o problema de roteamento de área. Ênfase deve ser dada novamente ao fato de que o que caracteriza o roteamento de área é a inexistência de estrutura e divisões naturais do problema, e a possível existência de obstáculos, em geral pequenos e restritos (a uma camada, a um sentido, a uma região, etc...) Dentre as diversas abordagens possíveis para tratar um problema completo de roteamento de área, pode-se distinguir quatro principais: roteamento detalhado completo rede a rede; decomposição em caixas de conexão; roteamento hierárquico; e roteamento planar.

Uma das dificuldades deste problema é que a quantidade de detalhes e o tamanho do problema o tornam sensível a muitos fatores, desde o posicionamento dos terminais das células até detalhes de implementação. Assim, é pouco comum existir comparações confiáveis entre estas classes de soluções. De todos estes, os dois primeiros ainda são os mais utilizados. Tratar as redes individualmente com algoritmos de pesquisa põe estes a prova, mas ao mesmo tempo é indispensável em situações específicas. A técnica de decomposição em caixas de conexão é avaliada em maiores detalhes a seguir. O roteamento de área está também muito relacionado com aspectos da metodologia de síntese física. Convém lembrar, por exemplo, que as técnicas mais usadas para roteamento de área, roteamento global, e roteamento em geral, são gulosas em relação à solução global do problema. Isto é, mesmo quando os algoritmos aplicados a uma rede individual ou a um subproblema restrito são exatos, no seu conjunto eles dependem de ordem e de decisões previamente feitas. Processos de otimização combinatória são empregados, mas avaliar uma parte considerável do espaço de soluções tem custo computacional muito alto, e então a dependência de decisões feitas permanece. Estas merecem ser cuidadosamente avaliadas, pois há grandes chances de especificação de problemas insolúveis (vide [JOH 97a]). Se não forem tomados cuidados especiais, isto pode acontecer com freqüência na abordagem de decomposição em caixas de conexão. Além do problema das dependências, em processos iterativos que caracterizam a otimização combinatória, pode ocorrer o problema dos ciclos, onde o processo entra em laços de alterações sem convergir.

6.1.1 Decomposição de área em caixas de conexão

A inexistência dos espaços dedicados para roteamento restrito dificulta muito a subdivisão do problema. Nesta abordagem o problema original é decomposto em pequenas regiões chamadas de células de roteamento global, ou *GRC*s [Kao95]. A divisão feita é artificial, e, portanto, a definição dos subproblemas se torna bastante arbitrária (Fig. 6.1). Para que as interfaces entre as *GRC*s sejam definidas, especifica-se o problema de **assinalamento de pontos de cruzamento**, ou *CPA*. Daí provém a necessidade de resolver o problema de *CPA* coerentemente entre as caixas de conexão que devem ser posteriormente roteados, que correspondem às *GRC*s. Esta abordagem é praticável. Entretanto, sua otimização é muito difícil. A dificuldade advém de dois principais fatores. Em primeiro lugar, há forte dependência entre *CPA* e roteamento interno dos *GRCs*, o que se traduzirá em dependências fortes entre todos os problemas de roteamento detalhado. Em segundo lugar, há grande dificuldade em otimizar o uso de cada caixa de conexão, pois se for limitado muito o número de conexões em cada um, eles serão subutilizados, e se o roteamento global especificar um número maior de conexões, pode facilmente gerar problemas de caixas de conexão insolúveis.

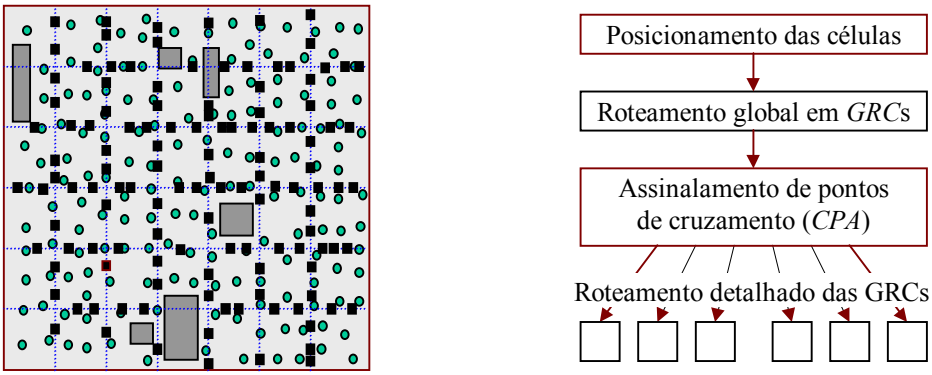


FIGURA 6.1 – Decomposição do Roteamento de Área.

6.1.2 Roteamento detalhado de área sem decomposição

Em contraposição a esta abordagem, pode-se buscar uma metodologia onde o roteamento detalhado seja realizado integralmente, e não em partes restritas. Esta é a visão particular do autor. A localidade das decisões pode ser explorada através da localidade do próprio algoritmo, mas sem as divisões artificiais e rígidas das *GRC*s. Este é justamente o modo pelo qual operam os algoritmos *greedy*. Assim, pode-se realizar o roteamento global que atribui as conexões às *GRC*s, e utilizar esta informação como guia para um roteamento detalhado de toda a área. Este roteamento é detalhado porque não deve mudar as formas e a distribuição das redes globalmente. Ele é integral porque irá considerar as dependências entre as *GRC*s, e sua localidade é processar o roteamento linha a linha, ou coluna a coluna. Um algoritmo *greedy* peca por não saber se sua convergência está sendo para um mínimo local ou global, sendo isto baseado nas decisões que tomou inicialmente. Então, avaliando a solução global e restringindo as decisões do algoritmo *greedy* a questões locais evita-se quase totalmente esta deficiência.

6.2 O Algoritmo LEGAL

O Algoritmo LEGAL foi proposto em [JOH 94], e utiliza os princípios de dois dos mais importantes algoritmos de roteamento de canal, o algoritmo *Left-Edge* e o algoritmo *Greedy*. A motivação para seu desenvolvimento advém dos seguintes fatos:

- os algoritmos *maze-router* são deficientes quanto ao problema de ordenação de conexões, o qual pode ser minimizado mas não solucionado;
- o tempo de execução destes algoritmos é exagerado, pois repetidas avaliações detalhadas da mesma área são feitas para cada conexão;
- a abordagem incremental não produz soluções que aproveitem bem os espaços, pois a alocação dos recursos pode ser considerada como aleatória, ou desorganizada;
- os algoritmos restritos produzem soluções com maior qualidade, eficiência e aproveitamento, mas só se aplicam a problemas bem limitados;
- não se deve tomar decisões globais e locais, ou locais entre si, com separação absoluta, para evitar que as decisões de um problema tenham impacto negativo na qualidade dos demais, ou do problema global;

A partir da observação destes fatos, se procurou desenvolver um novo algoritmo que apresente maior quantidade e melhor qualidade de resultados. A quantidade representa maior número de conexões realizadas no mesmo espaço, e também maior garantia de que uma solução será encontrada. Além destes objetivos principais, busca-se como objetivo secundário reduzir o tempo de execução. As estratégias usadas para perseguir estes objetivos foram: 1) realizar todas as conexões em paralelo; 2) usar os princípios de alocação de recursos e demandas dos algoritmos de roteamento de canal; 3) fazer análise detalhada da área simultaneamente em vez de repetidas vezes.

6.2.1 Funcionamento dos algoritmos de roteamento de canal

A principal diferença que chama a atenção ao se comparar os algoritmos de roteamento de canal *Left-Edge* e *Greedy* é que enquanto o primeiro processa o canal linha por linha, o segundo o faz coluna a coluna. Isto reflete muito mais do que simplesmente a orientação da varredura, pois condiciona na verdade os objetivos que são usados para solucionar o problema. Para o *Left-Edge*, o objetivo é buscar a máxima utilização de cada trilha, que corresponde a cada linha que toma, assinalando a estas os segmentos necessários à solução. Assim, ele obtém uma solução conjunta para todo o problema, que é ótima, guardadas as restrições iniciais. Entretanto, desconsidera a ocorrência de conflitos verticais, o que pode inviabilizar a solução em circuitos VLSI. O algoritmo *Greedy* tem como objetivo minimizar o número de trilhas (ou linhas) ocupadas por cada rede, enquanto também as aproxima do destino mais próximo. Para isto, ele usa ao máximo as conexões verticais, não se preocupando com sua otimização, pois a necessidade de roteamento vertical em canais é desprezível, em comparação com a necessidade horizontal, embora a área disponível para ambas as camadas seja a mesma. Este algoritmo apresenta uma boa solução heurística também através de uma abordagem integral, porém partindo sempre da análise do que falta fazer a partir do que já está feito.

Quando se observa o comportamento que estes dois algoritmos teriam em uma

grande área quadrada (não uma pequena região como uma caixa de conexão), percebe-se o seguinte: O algoritmo *Greedy* desconsideraria o aproveitamento das colunas, que são as etapas que avalia a cada passo, e também ficaria sobrecarregado na escolha de quais operações deve fazer se a seleção destas for muito acurada (como selecionar o conjunto de possíveis segmentos verticais que libere mais trilhas); O algoritmo *Left-Edge*, por sua vez, somente consideraria o aproveitamento das linhas, que é o que avalia a cada passo, ignorando completamente a grande necessidade de roteamento vertical, que inviabilizaria sua solução devido aos muitos conflitos; Desta observação tira-se duas conclusões proveitosas, normalizando a orientação com que os algoritmos avaliam a área: 1) Um deles considera exatamente o que falta considerar no outro; 2) Portanto, é possível unir estes dois tipos de análise, encontrando os pontos certos de contato entre o controle dos dois e produzindo um algoritmo que considere os princípios básicos de ambos

6.2.2 Funcionamento do algoritmo LEGAL

O algoritmo é chamado de LEGAL como abreviação de "*Left-Edge Greedy ALgorithm*", em referência aos algoritmos de canais dos quais provêm suas idéias básicas. O algoritmo LEGAL processa o circuito linha a linha, tendo como controle principal passos semelhantes ao algoritmo *Greedy*, onde as redes em roteamento são assinaladas às colunas. A cada linha, são identificados os segmentos necessários ou úteis à solução do problema, verificados os que são possíveis de implementar, e selecionado um grupo conforme prioridade e segundo o critério *Left-Edge*. A Fig. 6.2 apresenta o esqueleto de um algoritmo LEGAL.

LEGAL:

```
for each net pin;
        put pin in Terminal[pin.y];
for (line=0; line<circuit.y; ++lne) {
        extend live columns;
        legal_step1(line);
        legal_step2(line);
        legal_step3(line);
        legal_step4(line);
        }
```

legal_step1:

```
for each pin in Terminal[line] {
        calculate target_x; // esquerda ou direita
        found = scan_step1(pin.x,target_x,&x_found);
        if (not found)
                error("Cannot include new pin");
        if (Column[x_found] == 0)
                legal_move(pin.x,x_found);
        else
                legal_union(pin.x,x_found);
        }
```

FIGURA 6.2 – Esqueleto de um algoritmo LEGAL genérico.

A prioridade de segmentos horizontais possíveis está implícita nos passos *Greedy*. O significado deste processo é o seguinte: enquanto cada linha do circuito é roteada, dependências prévias são resolvidas, e o espaço restante é usado para fazer o que ainda pode ser feito em avanço. Essa é uma visão simplificada das prioridades. Cada passo faz uma tarefa específica: 1 – adiciona novos pinos; 2 – une redes presentes em mais de uma coluna (divididas); 3 – aproxima redes divididas; 4 – move redes para posições horizontais mais próximas de seu futuro destino (ver abaixo). Há funções separadas para identificar cada uma dessas situações, e também funções específicas de busca horizontal para encontrar novas posições que possam ser usadas por conexões em cada um destes passos. A Fig. 6.2 apresenta apenas as rotinas do passo 1. As funções *legal_move* e *legal_union* são funções genéricas que processam movimentos horizontais de uma conexão e união de redes divididas em duas colunas, respectivamente. Elas atualizam as informações de onde estão as redes, seus estados, e geram a conexão recém feita.

No algoritmo *Greedy* o destino de uma conexão somente pode ser para cima, para baixo, ou permanecer onde está, quando existem novos terminais em cima e embaixo. Já no contexto do algoritmo LEGAL, o melhor caminho para cada conexão pode ser bem variado, em função das quase infinitas possibilidades de posicionamento dos muitos terminais de uma rede. Para que o algoritmo LEGAL possa selecionar os segmentos desejáveis é necessário o uso de informações detalhadas sobre o objetivo de cada conexão, que irá determinar a forma das redes. O objetivo intermediário de cada conexão pode ser encontrado através de dois principais modos: utilizando um roteamento global prévio e restritivo, ou utilizando um planejamento global interno ao algoritmo, calculado durante o roteamento detalhado. Esta última opção é bastante adequada, sendo necessária mesmo com o roteamento global prévio, pois suas informações estão intimamente relacionadas com o algoritmo detalhado.

6.3 Implementações do algoritmo LEGAL

A dificuldade de aplicação e de obtenção de resultados com um algoritmo desta natureza é evidente, em função da complexidade do problema de roteamento de área e das inúmeras possibilidades de decisões do algoritmo. Por esta razão, três implementações são utilizadas para demonstrar que este tipo de algoritmo é viável. Cada uma delas possui suas particularidades, que não são poucas, e é apreciada de forma diferente, para mostrar as características que se deseja.

6.3.1 Resultados do algoritmo LEGAL no sistema MARTE

Pode-se implementar algoritmos LEGAL em diversos modelos de roteamento. Dentro do ambiente MARTE [JOH 94], foi feita a implementação de um protótipo adequado para o problema de roteamento com grande número de obstáculos e terminais em forma de barra vertical, que já demonstrava sua viabilidade. Os terminais com diversos pontos de acesso levaram à definição de novos passos para o algoritmo, segundo prioridades diferentes de terminais que estão no primeiro ou no último ponto de acesso. Tal protótipo podia realizar o roteamento de pequenos circuitos (Fig. 6.3 esquerda) em tempo ordens de grandeza menor, com uma pequena perda na qualidade, se comparado com um sistema baseado em *maze routers* usando diversas técnicas de otimização (Fig. 6.3, direita) [JOH 94].

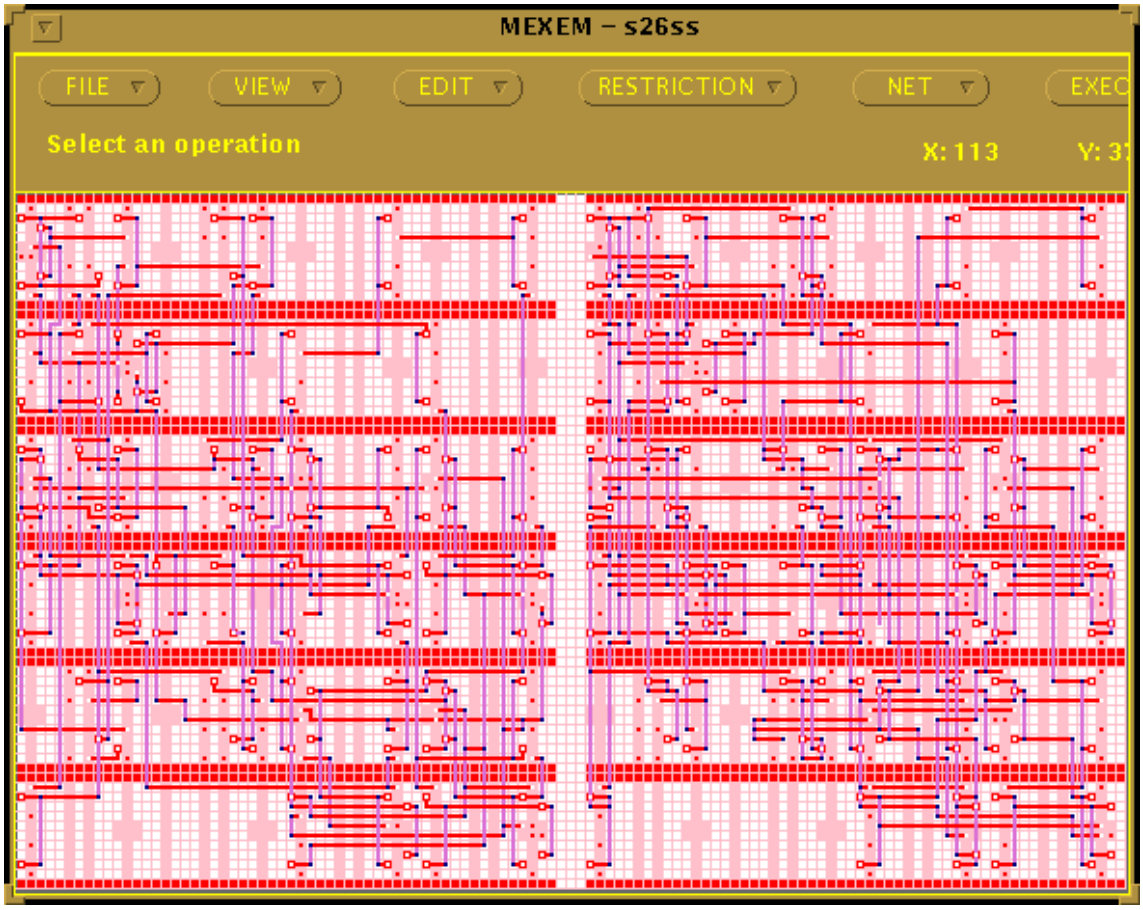


FIGURA 6.3 – Roteamento simbólico com *maze routers* e com o algoritmo LEGAL no sistema MARTE.

A principal razão para o pouco número de experiências realizadas foi que a implementação do protótipo feita rapidamente não dispunha de estruturas de dados adequadas. Em especial, a estrutura de representação de terminais por partes usada no MARTE não preservava a identidade destes, e as muitas referências que o algoritmo LEGAL devia manter a estes terminais falhavam quando eram atualizados. A estrutura de informações da linha sendo roteada pelo LEGAL também não era muito bem definida. Deve-se separar as informações de uma rota sendo traçada em uma coluna (mas que pode mudar de coluna) de informações de acesso e ocupação de uma determinada posição horizontal na linha atual. Mesmo assim, embora o protótipo apresentasse falhas de funcionamento que o impedissem de tratar circuitos muito grandes, pôde-se demonstrar a viabilidade de um algoritmo deste tipo, considerando o mesmo ambiente de células transparentes com restrições às camadas de roteamento.

6.3.2 Roteamento global com o algoritmo LEGAL no sistema GAROTA

No sistema GAROTA, um algoritmo mínimo inicialmente implementado foi substituído por uma implementação do LEGAL dedicada a este contexto. O algoritmo faz o roteamento global canal a canal, de baixo para cima, criando segmentos locais e assinalando, a estes, pinos das bandas abaixo e acima respeitando as capacidades dos canais. O algoritmo opera com 5 passos conforme as prioridades naturais neste contexto:

1. última chance para conectar pinos na banda inferior ainda isolados entre si e cuja rede não aparece na próxima banda;
2. propaga o roteamento assinalando cada pino da banda inferior a um segmento que conecte um pino da mesma rede na próxima banda;
3. une segmentos diferentes da mesma rede se há espaço no canal;
4. adiciona pinos da próxima banda a segmentos de rede já existentes;
5. gera novos segmentos para conectar pinos da próxima banda se há espaço;

A implementação demonstrou-se bastante eficiente, mantendo a velocidade característica de um algoritmo direto, sendo imperceptível a mudança. O novo roteador global acomoda as conexões de circuitos bem grandes com facilidade, reduzindo os problemas de sobrecarga nos canais a casos raros. Este é o algoritmo atualmente em uso no sistema GAROTA, e cujos resultados foram considerados no Cap. 3.

6.3.3 Roteamento detalhado com o algoritmo LEGAL

Esta Seção apresenta uma especificação mais precisa do algoritmo LEGAL através de uma implementação simples, cujo código fonte se encontra no anexo 6. O código usa o sistema de tratamento de exceções e de leitura UFDS do sistema GAROTA. A implementação, denominada de ***ilegal***, considera apenas áreas vazias, terminais com um único ponto de acesso, e redes de apenas dois terminais, ou seja, conexões ponto a ponto. A vantagem de se ter estas limitações está em comparar dois algoritmos bem definidos, sem deixar que decisões complexas de formação de redes interfiram no resultado. A partir desta implementação, podemos melhor identificar quais os elementos envolvidos, alternativas, escolhas, decisões, modelos e relações do algoritmo, principalmente com roteamento global.

O algoritmo opera com quatro passos principais: inclusão de novos terminais, união de redes partidas (em mais de uma coluna), aproximação de redes partidas, e aproximação de destino futuro. Para cada um dos passos há uma rotina inicial que procura a posição adequada para a coluna sendo tratada, considerando a posição-objetivo. Nesta implementação simples, a posição-objetivo é dada pela ordem dos terminais nas redes partidas, ou pela coordenada do próximo terminal em uma rede presente em apenas uma coluna. Os terminais são previamente ordenados em y e x para facilitar sua referência.

As operações executadas em qualquer passo podem ser apenas união ou movimento, sendo que o movimento pode partir diretamente do terminal ou de uma rede já presente em uma coluna. Não há necessidade de planejamento global e seleção de colunas para continuar, como na implementação do sistema MARTE, já que agora as redes têm somente dois pinos cada. A única medida implementada para evitar decisões globais erradas foi a inclusão de condições nas rotinas de busca dos passos 3 e 4, as quais impedem que as redes sejam assinaladas a colunas livres quando estas já não possuem outras colunas livres adjacentes. Isto procura evitar que novos terminais não encontrem uma coluna livre para serem inseridos no passo 1, mas certamente um mecanismo mais exato de reserva de colunas pode ser empregado.

A Tab. 6.1 mostra os tempos de *CPU* e a percentagem de conexões realizadas por um *maze router* básico (apenas com substituição e processamento de *doglegs*), pelo sistema MARTE com todas otimizações (ordenação, canalização e direções livres), e

pela implementação ***ilegal*** (Fig. 6.4). Os exemplos foram extraídos de circuitos do banco de testes ISCAS, posicionados em matrizes de portas MARCELA [GUN 95]. A geração dos exemplos decompôs as redes de múltiplos terminais em um conjunto de redes de 2 terminais cada, ignorando a conectividade entre estes pares. Com este processo, as redes menores ficam simplificadas, mas as redes maiores exigem mais roteamento, já que, sendo distintas, têm rotas que não podem colidir ou unirem-se.

TABELA 6.1 – Desempenho da implementação ***ilegal*** e do MARTE.

	circuito		MARTE otimizado		***ilegal***	
nome	área	redes	CPU (s)	% rot	CPU (s)	% rot
s28-2	72x80	79	6.0	98.73	0.0	98.73
s386-2	168x216	546	26.0	100	2.0	100
s510-2	168x288	590	48.0	99.66	3.0	97.79
c880-2	280x264	819	76.0	99.87	4.0	99.75
c499-2	350x384	990	68.0	99.89	6.0	100
s1494-2	350x408	1870	425.0	98.12	12.0	89.51

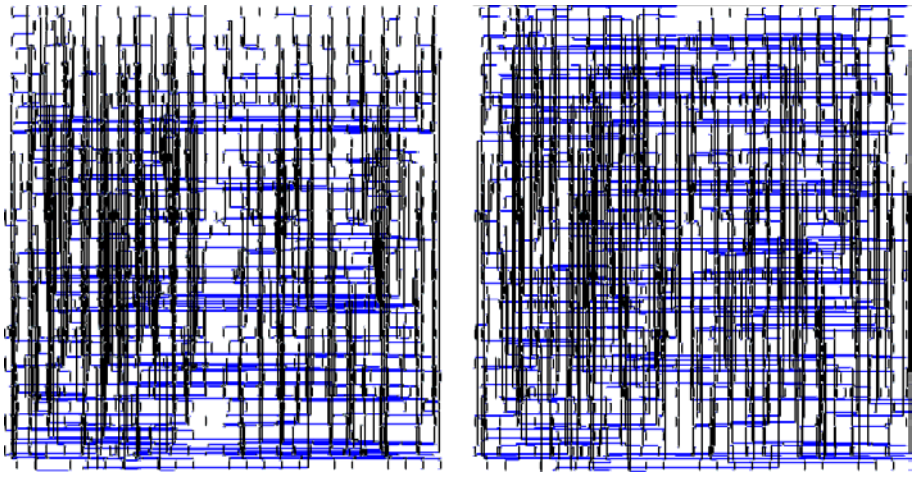


FIGURA 6.4 – Leiaute das conexões do c499-2 roteadas com o ***ilegal*** e com os *maze routers* otimizados do sistema MARTE.

Observa-se que esta implementação do algoritmo LEGAL possui uma capacidade de conexão semelhante à do sistema MARTE com todas otimizações, mas gasta um tempo de CPU insignificante. Os tempos foram tomados em uma antiga estação SUN® Sparc 1 para que pudessem ser medidos externamente. O último circuito exemplo, s1494-2 demonstra, ao contrário, a fragilidade do algoritmo LEGAL quanto à distribuição global das conexões (Fig. 6.5), sendo que nesse caso os roteadores rede a rede ainda se demonstram mais eficientes (Fig. 6.6). Esta limitação pode ser aliviada com o mecanismo de reservas de colunas já mencionado, mas o problema é certamente mais relacionado a decisões globais de rotas, e não locais. Por esta razão, o principal ponto a desenvolver em um algoritmo deste tipo está na previsão de um controle global já determinado por outro algoritmo. Este controle deve definir e atualizar a posição de destino de cada coluna sendo roteada (*target_x*) de modo que o LEGAL apenas decida como cumprir isto. Deve ser usada uma rotina adicional para controlar os valores de *target_x*, e as rotinas de busca (*scan*) também precisam ser alteradas, para que o algoritmo cumpra as decisões globais e tome as melhores decisões detalhadas.

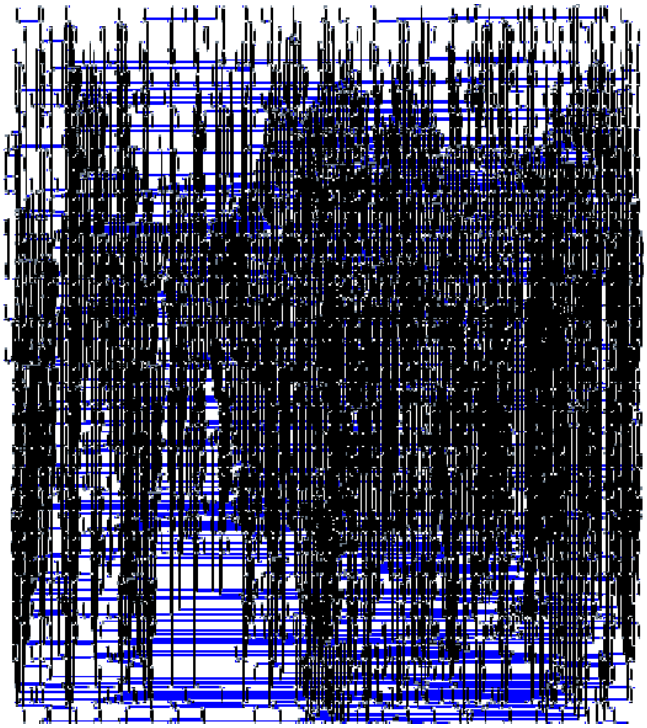


FIGURA 6.5 – Leiaute das conexões do s1494-2 roteadas com o ***ilegal***.

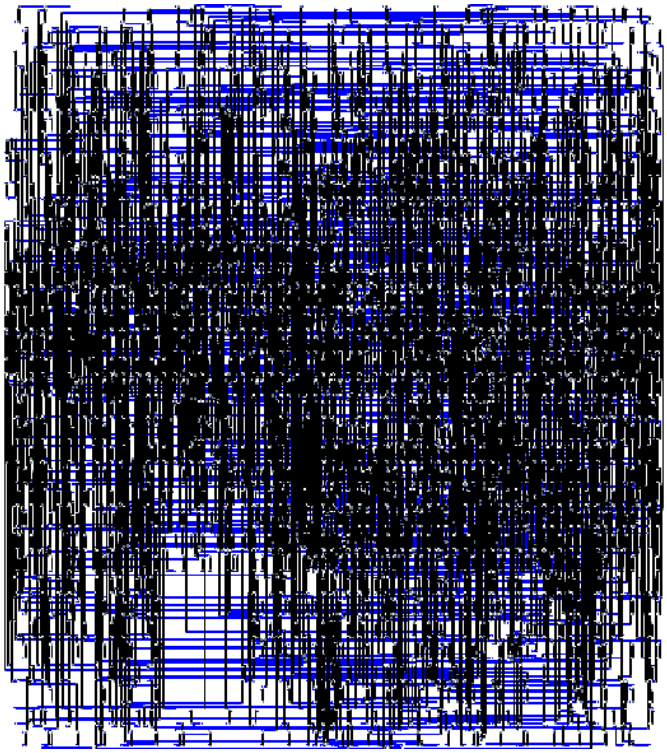


FIGURA 6.6 – Leiaute das conexões do s1494-2 roteadas com *maze routers*.

De qualquer forma, comprova-se a tese de que é possível realizar o roteamento detalhado simultâneo das redes de um circuito, e de que este processo pode ser tão eficiente quanto outros algoritmos já conhecidos, porém muito mais rápido. Espera-se que a implementação deste planejamento e o tratamento de redes de múltiplos terminais favoreçam o algoritmo LEGAL e o tornem uma alternativa muito mais atraente se comparado com algoritmos de pesquisa. Convém destacar também que, na medida em

que é mais difícil a implementação de um algoritmo deste tipo pela presença de maior número de obstáculos, é mais apropriado o uso de algoritmos baseados em pesquisa rede a rede. Em contrapartida, em áreas mais vazias, nas quais os *maze routers* são mais desorganizados, um algoritmo LEGAL opera mais facilmente, de forma que temos técnicas complementares para tratar um universo maior de situações.

A Tab. 6.2 mostra o comprimento total de conexões geradas para os circuitos roteados com o sistema MARTE e com a implementação ***ilegal***, separado em horizontal e vertical. Observa-se que, de fato, a implementação ***ilegal*** produz roteamento mais longo. Isto não é anormal, porém, pois observa-se que as otimizações do sistema MARTE também implicam em maior comprimento final de conexões em relação à versão não otimizada, para proporcionar as vantagens de maior velocidade e maior taxa de roteamento. Ou seja, paga-se com conexões um pouco maiores para encontrar uma solução global aceitável.

TABELA 6.2 – Comprimento de conxões no MARTE e no ***ilegal***.

	maze router simples		MARTE otimizado		***ilegal***	
circuito	H	V	H	V	H	V
s28-2	908	903	917	929	1017	997
s386-2	8354	13876	8685	14183	9379	14512
s510-2	11576	21577	11954	21751	12438	22130
c880-2	17370	23841	17964	24669	19071	25110
c499-2	23884	37804	24239	38421	25430	39046
s1494-2			55506	87521	52777	85429

6.4 Propostas para futuros estudos

Deve-se tomar cuidado na avaliação de qualidade com alguns aspectos específicos, como proximidade das conexões e compactação do leiaute. A proximidade e a compactação podem ou não ser vantajosas. São vantajosas quando usadas como recursos necessários para se ter maior densidade de conexões, e desvantajosas quando o algoritmo abusa do poder de compactação, e desnecessariamente comprime as conexões em pequenas regiões quando há espaços disponíveis na vizinhança. Neste caso, a proximidade dos elementos reduz a confiabilidade no processo de fabricação e aumenta efeitos elétricos parasitas. A compactação pode ter também um outro efeito colateral danoso que é a dificuldade de se alterar conexões individuais com um algoritmo de pesquisa, após terem sido estabelecidas pelo LEGAL.

Para considerar estes e outros aspectos importantes em roteamento, se propõem estudos específicos nos quais a viabilidade de implementação de determinados mecanismos em um algoritmo LEGAL é verificada. Estes estudos diferem das demais comparações pela maior complexidade de implementação e dificuldade de comparação. As principais condições com as quais se deseja trabalhar são:

- Inserção de espaços no roteamento – para minimizar efeitos de acoplamento e distribuir melhor o congestionamento, pode ser necessário, em um algoritmo LEGAL, controlar a densidade de conexões sendo gerada, e interferir no processo

pela inclusão proposital de espaços durante a realização das conexões;

- Inserção de espaços no posicionamento – o posicionamento absoluto de um circuito pode precisar por natureza de mais conexões do que é possível ser realizado sobre ele. Em circuitos baseados em bandas, pode-se deslocá-las durante o próprio processo de roteamento para que todas as conexões sejam completadas. Em projetos de altíssimo desempenho isto pode não ser desejável, pois provocará alteração nas previsões já feitas sobre a qualidade elétrica das conexões, mas como técnica geral é algo inovador e muito superior às soluções utilizadas;
- Roteamento com até 4 camadas – o algoritmo LEGAL se aplica inicialmente para o roteamento de sinais em duas camadas, o que é plenamente justificado pelas próprias diferenças entre elas. Estas diferenças fazem com que as conexões sejam classificadas em: locais, semiglobais e globais, de forma que utilizem camadas diferentes. Entretanto, tendo mais de 6 camadas metálicas com características equivalentes, é interessante estudar a viabilidade de implementação de algoritmos semelhantes para 3 ou 4 camadas.
- Roteamento com conexões de largura variável – até determinado ponto, conexões que necessitam larguras específicas podem ser realizadas associando-as a diferentes camadas de roteamento, como descrito acima. Já em casos críticos de desempenho elétrico, é necessário que o algoritmo de roteamento acomode na mesma área e em mesma camada conexões com diferentes larguras e espaçamentos, onde não existe grade explícita. Pode-se imaginar a possibilidade de implementação de um algoritmo de varredura semelhante ao LEGAL, sem que as conexões devam estar em uma grade. Já que o LEGAL opera linha a linha, é uma alternativa atraente implementar uma grade implícita ou uma micro-grade, pois mesmo que a estrutura tenha maior consumo de memória, será armazenada apenas localmente.

7 Conclusões

As conexões em um circuito integrado atual merecem destacada atenção durante a etapa de síntese. Foi visto que suas características demandam modelos mais precisos, algoritmos mais rápidos, e também um processo convergente para encontrar uma solução aceitável em um espaço intratável de possibilidades. Este trabalho apresentou uma reflexão crítica e objetiva sobre o conjunto de problemas e algoritmos que ocorrem no roteamento destas conexões. A eficiência e a convergência deste processo foram avaliadas no Cap. 3 usando exemplos de sistemas de roteamento reais. Esta é a contribuição que foi dada para a questão de metodologia, comprovada com o desenvolvimento do sistema GAROTA. Apesar do pequeno número de elementos, o sistema envolve uma considerável riqueza de detalhes, cuja compreensão é indispensável na aplicação de quaisquer métodos conhecidos de roteamento.

Dois novos algoritmos foram propostos para problemas notáveis na área. O primeiro algoritmo, LCS*, tem grande potencial de acelerar processos de pesquisa em geral, o que pode beneficiar inúmeras aplicações em áreas variadas da ciência da computação. Dos admissíveis, LCS* é o primeiro algoritmo heurístico bidirecional genérico que pode ser mais eficiente que o A*, efeito este que vinha sendo perseguido há 30 anos. Algoritmos heurísticos bidirecionais anteriores apenas apresentavam ganhos em comparação com a pesquisa heurística unidirecional quando limitados a determinadas condições muito específicas. Para roteamento usando algoritmos de pesquisa, a principal contribuição está na comparação dos algoritmos heurísticos com os não heurísticos, e no uso de um modelo de custo que codifica corretamente os objetivos de cada conexão ou etapa. Esse modelo permite explorar melhor as relações entre dificuldade por congestionamento, custo e desempenho de conexões, e é portanto adequado tanto para a convergência do processo quanto para a melhor qualidade de conexões consideradas críticas.

O segundo algoritmo apresentado é o LEGAL, cujo potencial está em realizar o roteamento detalhado de área de modo integral, sem divisões, e para todas as redes do circuito. Este algoritmo faz análise detalhada do espaço uma única vez, e acomoda as conexões de forma a bem aproveitar o espaço, tendo sido baseado nos algoritmos de roteamento de canal mais eficientes. As incertezas de um processo guloso são evitadas prevendo o uso de informações de roteamento global, de forma que as decisões do algoritmo sejam locais, usando também prioridades inteligentes na escolha de conexões. Este tipo de algoritmo foi experimentado com três diferentes implementações, tendo sido apresentada uma definição simplificada precisa. Se propõe estudar a viabilidade de sua implementação em domínios com outros objetivos particulares como minimização de proximidade ou carência de espaço no posicionamento.

Desta forma, o trabalho apresentou diversas propostas originais e importantes aplicadas a problemas de roteamento de circuitos VLSI. A validade destas propostas foi comprovada com a implementação de sistemas reais, desenvolvimento de provas de propriedades formais, e com a condução de experimentos que demonstram efeitos que ocorrem em situações gerais e particulares.

Anexo 1 Artigo sobre o sistema GAROTA

A seguir se encontra o artigo "*Functional Design of GAROTA: Gate Array Router of ÁGATA system*", publicado nos anais do SBCCI 1997 [JOH 97c]. O artigo descreve o projeto funcional do sistema, não entrando em aspectos de implementação, e sua leitura é recomendada para compreensão do Cap. 3. São apresentados detalhes da arquitetura da matriz de *gate array*, dos problemas de roteamento, o fluxo de síntese, bem como resultados e otimizações sugeridas após as primeiras versões, algumas das quais foram posteriormente implementadas.

Anexo 2 Artigo sobre o sistema MARTE

Este anexo contém o artigo intitulado "*A Full Over-the-Cell Routing Model*", publicado nos anais do IFIP VLSI 95 [JOH 95]. O trabalho apresenta o sistema de roteamento MARTE como uma ferramenta adequada para o roteamento sobre células transparentes. O trabalho defende o uso deste modelo para geração automática de blocos em lógica aleatória, e mostra que o roteamento pode ser realizado com sucesso. A densidade de integração atingida supera aquela dos circuitos com células padrão, e o uso de geradores de células transparentes permite utilizar portas complexas que reduzem o número de transistores no circuito. Neste modelo não há espaço dedicado para as conexões, e o pequeno número de camadas metálicas, já usadas no projeto das células, provoca a existência de inúmeras restrições ao roteamento. Por esta razão os algoritmos *maze routers* foram escolhidos, mas a proposta de desenvolvimento de um algoritmo de roteamento simultâneo já era sugerida.

Anexo 3 Artigo sobre o algoritmo LCS*

A seguir encontra-se uma cópia do primeiro artigo internacional sobre o algoritmo LCS*, publicado nos anais do ICSC, CI2000 [JOH 2000a]. Esta é incluída aqui por conter a apresentação e o pseudo-código do algoritmo, além de alguns outros gráficos com testes. Caso o leitor tenha interesse em implementar o algoritmo LCS* e encontre alguma dificuldade em interpretar o pseudo código e montar as estruturas necessárias, é estimulado a entrar em contato com o autor através do endereço eletrônico johann@inf.ufrgs.br, johann@pucrs.campus2.br, ou visitando às páginas do grupo de microeletrônica (GME) da UFRGS em "http://www.inf.ufrgs.br/gme".

Anexo 4 Artigo com os teoremas sobre LCS*

Esta é uma cópia do artigo intitulado "*Admissibility Proofs for the LCS* Algorithm*", apresentando no SBIA 2000 e publicado na série *Lecture Notes on Artificial Intelligence*, LANAI-1952, [JOH 2000b]. O artigo apresenta os teoremas que provam a perfeição e admissibilidade do algoritmo LCS*.

Anexo 5 – Matriz *Gate Array* GA2500

A Fig. A5.1 mostra o leiaute da matriz GA2500 projetada no CTI, para cuja arquitetura foi desenvolvido o sistema GAROTA. A matriz possui 12520 transistores, 122 *pads* dos quais 112 são configuráveis como entrada, saída ou bidirecionais. A Fig. A5.2 mostra o canto inferior esquerdo da mesma matriz, onde se pode observar detalhes de posicionamento das bandas e canais de roteamento contendo os *underpasses*. A Fig. A5.3 mostra detalhes da máscara de metal de um circuito para esta mesma matriz, onde se vê a personalização de células, roteamento nos canais, e os padrões de roteamento de *pads*.

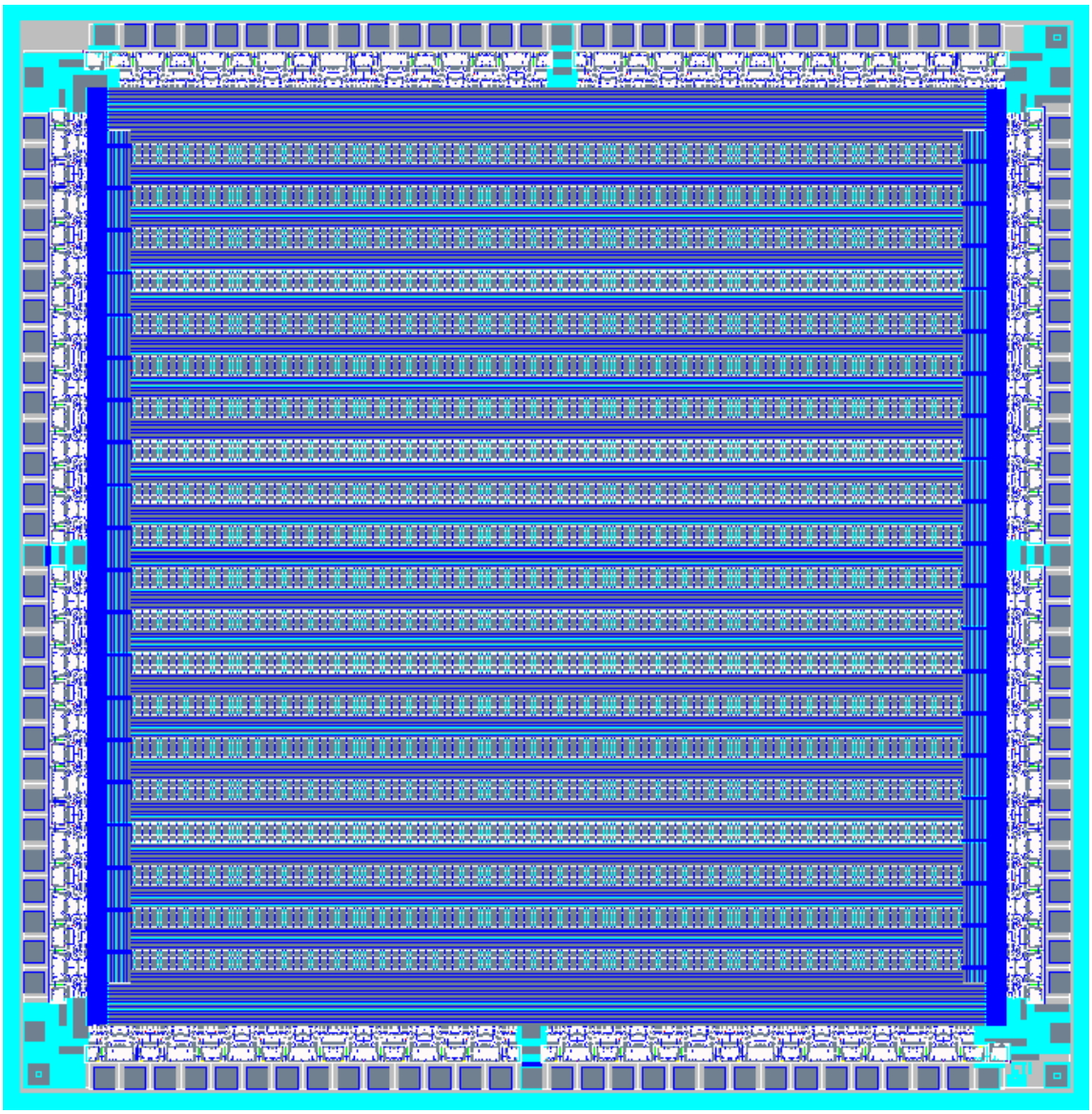


FIGURA A5.1 – Leiaute da matriz GA2500.

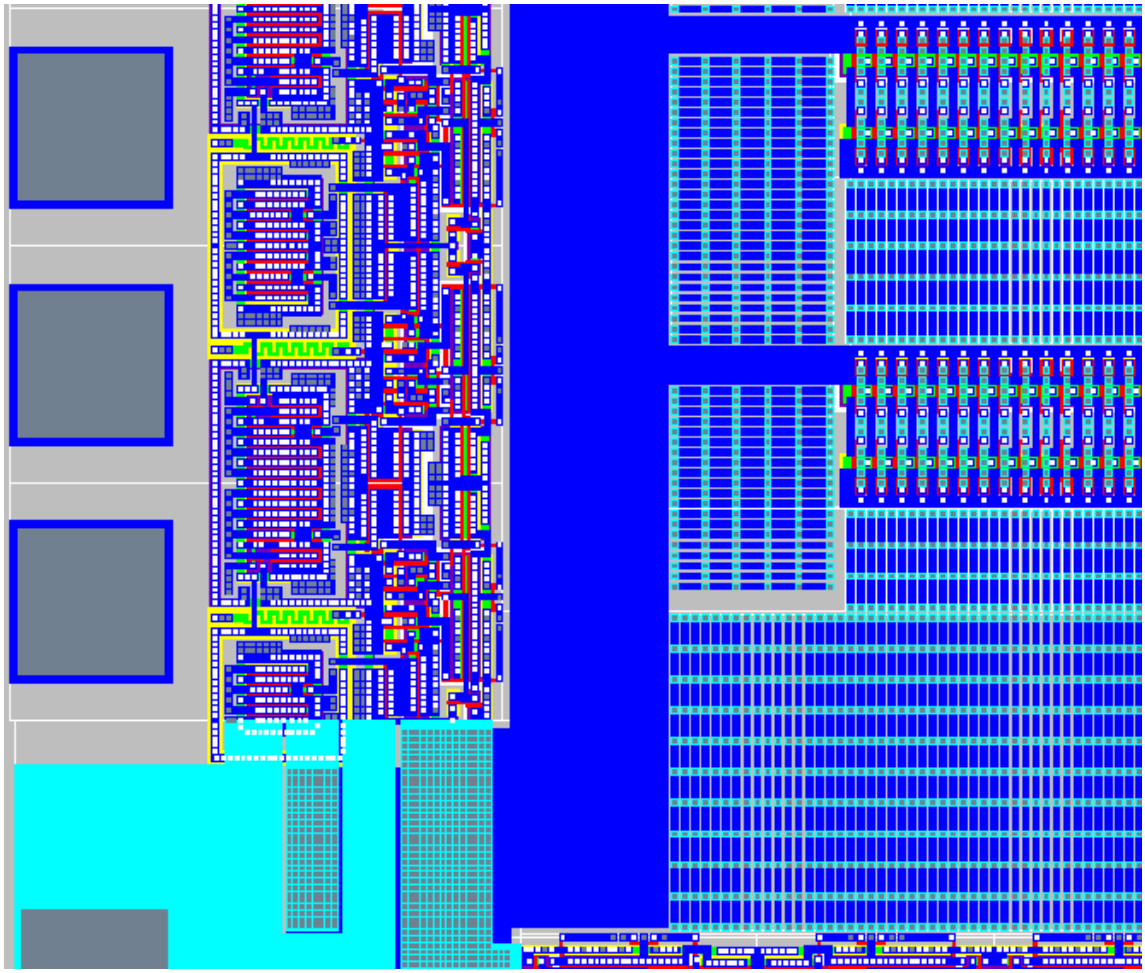


FIGURA A5.2 – Canto inferior esquerdo da matriz GA2500.

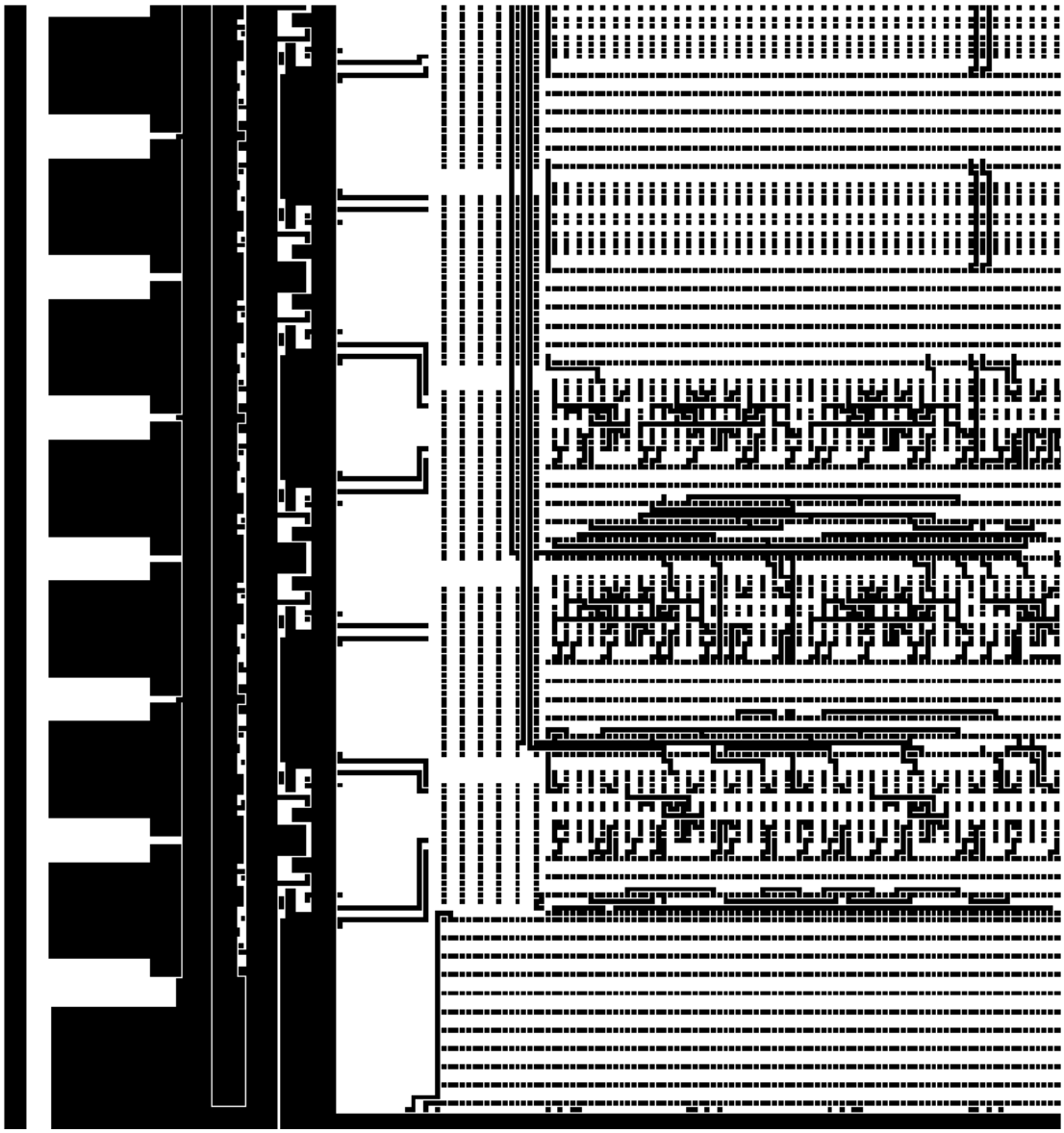


FIGURA A5.3 – Metalização do canto inferior esquerdo de um circuito.

Anexo 6 – Código fonte *ilegal*

```
/*************************************************************************
*
*  ilegal.h : an Implementation of the Left-Edge Greedy ALgorithm.
*
*  PPGC - UFRGS - Marcelo Johann - feb. 2001 - BLESSED BE THE LORD!
*
*  Last Update: 22/02/2001 at PUCRS-Uruguaiana
*
*************************************************************************/

#ifndef H_ILEGAL
#define H_ILEGAL

/****** INCLUDES ********************************************************/

#include "ufds_par.h"
#include "merrors.h"

/****** DEFINES *********************************************************/

#define FALSE       0
#define TRUE        1
#define MAX_PATH          32

/****** STRUCTURES ******************************************************/

typedef struct legal_net
  {
  char              *name;
  char              split;
  char              alive;
  char              first_col;
  int               x1;
  int                     y1;
  int                     x2;
  int                     y2;
  struct legal_routing    *routing;
  struct legal_net  *next;
  } NET;

typedef struct legal_pin
  {
  int               x;
  int                     y;
  struct legal_net  *net;
  struct legal_pin  *next;
  } PIN;

typedef struct legal_routing
  {
  int                     x;
  int                     y;
  char              layer;
  char              *path;
  struct legal_routing    *next;
  } ROUTING;

typedef struct legal_column
  {
  NET                     *net;
  char              layer_used;
  char              layer_this;
  int                     target_x;
  int                     last_path;
  } COLUMN;

/****** PROTOTYPES ******************************************************/
```

```
int    ilegal       (int argc, char **argv);
int    read_netlist (char * file);
int    store_path          (NET* p, int x, int y, char *path, char layer);
int    run_legal           ( );
int    legal_step1         (int line);
int    legal_step2         (int line);
int    legal_step3         (int line);
int    legal_step4         (int line);
int    scan_step1          (int line, PIN *pin, int *x, int *found, int
target_x );
int    scan_step2          (int line, NET *net, int col, int *x, int *found);
int    scan_step3          (int line, NET *net, int col, int *x, int *found,
                           int first_col);
int    scan_step4          (int line, NET *net, int col, int *x, int *found);
int    legal_move          (NET* net, int line, int x1, int x2, int col1);
int    legal_union         (NET* net, int line, int x1, int x2, int col1);
int    legal_draw          (NET* net, int line, int x1, int x2, int col1);

int    save_routing        (NET* netlist,char *file,char *file2);

int    insert_ilegal_functions   ( void );
int    area                ( ARG_HEAD *arg_list );
int    net                 ( ARG_HEAD *arg_list );
int    pin                 ( ARG_HEAD *arg_list );
int    ilegal_message      ( int number, char **msg );

/******* END **********************************************************************/

#endif




/*****************************************************************************
*
*  ilegal.c : An Implementation of the Left-Edge Greedy ALgorithm.
*
*  PPGC - UFRGS - Marcelo Johann - feb. 2001 - BLESSED BE THE LORD!
*
*  Last Update: 14/02/2001 at PUCRS-Uruguaiana
*
*****************************************************************************/

#include "ilegal.h"
#include "merrors.c"
#include "mhash.c"
#include "express.c"
#include "ufds_par.c"

/****** VARIABLES *************************************************************/

NET          * Netlist = 0;
PIN          **Terminal = 0;
int          Number_of_nets = 0;
int          Missing_nets = 0;
int          XSize = 0;
int    YSize = 0;
COLUMN       *Column = 0;
char         Buffer[MAX_PATH];

/****** FUNCTIONS *************************************************************/

int    main               (int argc, char **argv)
{
ilegal(argc,argv);
write_errors(get_error(),0);
exit(get_error());
return(reset_warning());
}

/*****************************************************************************/
```

```
int    ilegal       (int argc, char **argv)
{
set_messages_on_demand();
if (argc < 4)
  return( set_error(3,ilegal_message,0,0));
CALL0( read_netlist(argv[1]) );
CALL0( run_legal() );
CALL0( save_routing(Netlist,argv[2],argv[3]) );
return(reset_warning());


/*************************************************************************/
int                 run_legal( )
{
int                 line = 0;
int                 col = 0;
for (line = 0; line < YSize; ++line)
  {
  /*----- keep nets that have to continue */
  for (col = 0; col < XSize; ++col )
    {
    Column[col].layer_used = FALSE;
    Column[col].layer_this = FALSE;
    if ( Column[col].net )
      {
      if ( Column[col].net->alive == FALSE )
         Column[col].net = 0;
      else
        {
        ++Column[col].last_path;
        Column[col].net->first_col = TRUE;
        }
      }
    }
  /*----- execute LEGAL steps */
  CALL0( legal_step1(line) );
  CALL0( legal_step2(line) );
  CALL0( legal_step3(line) );
  CALL0( legal_step4(line) );

  }
return(reset_warning());
}


/*************************************************************************/
int                 legal_step1(int line)
{
PIN    *pin = 0;
int    x=0, found=0;
int    i = 0;
int    target_x = 0;
for (pin = Terminal[line]; pin; pin = pin->next )
  Column[pin->x].layer_used = TRUE;
for (pin = Terminal[line]; pin; pin = pin->next )
  {
  /*----- find target position and insertion column */
  if (pin->net->split == 0)
    if ( pin->net->x2 <= pin->net->x1 )
      target_x = pin->net->x2 + 1;
    else
      target_x = pin->net->x2 - 1;
  else
    if ( pin->net->x1 <= pin->net->x2 )
      target_x = pin->net->x1 + 1;
    else
      target_x = pin->net->x1 - 1;
  CALL( scan_step1(line, pin, &x, &found,target_x),
    set_error(7,ilegal_message,line,0) );
  if (found != TRUE)
    {
    set_error(7,ilegal_message,line,0);
```

```
    pin->net->alive = FALSE;
    ++Missing_nets;
    continue;
    }
  /*----- update column and net information */
  if (Column[x].net == 0)
    {
    ++pin->net->split;
    legal_move(pin->net,line, pin->x,x,FALSE);
    Column[x].target_x = target_x;
    }
  else
    legal_union(pin->net,line, pin->x,x,FALSE);
  }

return(reset_warning());
}


/*****************************************************************************/
int          scan_step1(int line, PIN *pin, int *x, int *found, int target_x )
{
if ( target_x <= pin->x )
  {
  for (*x=pin->x - 1; *x > 0 &&
       (Column[*x].layer_used == FALSE || *x == pin->x ) ; --(*x) )
    if (Column[*x].net == 0 || Column[*x].net == pin->net)
      {
      *found = TRUE;
      return(reset_warning());
      }
  for (*x=pin->x + 1; *x < XSize &&
       (Column[*x].layer_used == FALSE || *x== pin->x ) ; ++(*x) )
    if (Column[*x].net == 0 || Column[*x].net == pin->net)
      {
      *found = TRUE;
      return(reset_warning());
      }
  }
else
  {
  for (*x=pin->x + 1; *x < XSize &&
       (Column[*x].layer_used == FALSE || *x== pin->x ) ; ++(*x) )
    if (Column[*x].net == 0 || Column[*x].net == pin->net)
      {
      *found = TRUE;
      return(reset_warning());
      }
  for (*x=pin->x - 1; *x > 0 &&
       (Column[*x].layer_used == FALSE || *x == pin->x ) ; --(*x) )
    if (Column[*x].net == 0 || Column[*x].net == pin->net)
      {
      *found = TRUE;
      return(reset_warning());
      }
  }

*found = FALSE;
return(reset_warning());
}

/*****************************************************************************/
int    legal_move(NET* net, int line, int x1, int x2, int col1)
{
if (col1==TRUE)
  {
  Column[x1].net = 0;
  Column[x2].target_x = Column[x1].target_x;
  }
Column[x2].net = net;
Column[x2].last_path = 1;
Column[x2].layer_this = TRUE;
legal_draw(net,line,x1,x2,col1);
```

```
return(reset_warning());
}


/*************************************************************************/
int legal_union(NET* net, int line, int x1, int x2, int col1)
{
if (col1==TRUE)
  Column[x1].net = 0;
Column[x2].net = 0;
net->split = 0;
legal_draw(net,line,x1,x2,col1);
/*----- create the second vertical path */
sprintf(Buffer,"%dv\0",Column[x2].last_path);
store_path(net,x2,line - Column[x2].last_path + 1,Buffer,'h');
return(reset_warning());
}

/*************************************************************************
*/
int                 legal_draw(NET* net, int line, int x1, int x2, int col1)
{
int                 x=0;
/*----- create routing paths */
if (col1==TRUE)
  {
  sprintf(Buffer,"%dv\0",Column[x1].last_path);
  store_path(net,x1,line - Column[x1].last_path + 1,Buffer,'h');
  }
if (x2 > x1)
  {
  sprintf(Buffer,"%dh\0",x2-x1 + 1);
  store_path(net,x1,line,Buffer,'m');
  for (x=x1; x<=x2; ++x)
    Column[x].layer_used = TRUE;
  }
else
  {
  sprintf(Buffer,"%dh\0",x1-x2 + 1);
  store_path(net,x2,line,Buffer,'m');
  for (x=x1; x<=x2; ++x)
    Column[x].layer_used = TRUE;
  }
return(reset_warning());
}


/*************************************************************************/
int                 legal_step2(int line)
{
int    col = 0;
NET    *net = 0;
int    x=0, found=0;
int    i = 0;
for (col = 0; col < XSize; ++col )
 if (Column[col].net != 0)
  if (Column[col].net->split > 1)
  {
  /*----- scan for the position of the second pin */
  net = Column[col].net;
  CALL( scan_step2(line, net, col, &x, &found),
    set_error(8,ilegal_message,line,0) );
  if (found == TRUE)
    legal_union(net,line, col,x,TRUE);
  }
return(reset_warning());
}

/*************************************************************************/
int                 scan_step2(int line, NET *net, int col, int *x, int *found)
{
for (*x=col; *x < XSize &&
       ( Column[*x].layer_used == FALSE || Column[*x].layer_this == TRUE );
```

```
++(*x) )
  if (Column[*x].net == net && *x != col)
    {
    *found = TRUE;
    return(reset_warning());
    }
*found = FALSE;
return(reset_warning());
}

/*************************************************************************/
int                 legal_step3(int line)
{
int    col = 0;
NET    *net = 0;
int    x=0, found=0;
int    i = 0;
for (col = 0; col < XSize; ++col )
 if (Column[col].net != 0)
  if (Column[col].net->split > 1)
  {
  /*----- scan for the position of the second pin */
  net = Column[col].net;
  CALL( scan_step3(line, net, col, &x, &found, net->first_col),
    set_error(8,ilegal_message,line,0) );
  if (found == TRUE)
    legal_move(net,line, col,x,TRUE);
  net->first_col = FALSE;
  }
return(reset_warning());
}


/*************************************************************************/
int    scan_step3(int  line,  NET  *net,  int  col,  int  *x,  int  *found,  int
first_col)
{
int    i = 0;
*found = FALSE;
if (first_col == TRUE)
  {
  for (i=col; i < XSize && Column[i].layer_used == FALSE ; ++(i) )
    if (Column[i].net == 0 && i != col)
      {
      if (i < XSize-3 && i > 2)
         if ( Column[i-1].net != 0 && Column[i+1].net != 0 &&
              Column[i-2].net != 0 && Column[i+2].net != 0 )
           continue;
      *x = i;
      *found = TRUE;
      }
  }
else
  {
  for (i=col; i > 0 && Column[i].layer_used == FALSE ; --(i) )
    if (Column[i].net == 0 && i != col)
      {
      if (i < XSize-3 && i > 2)
         if ( Column[i-1].net != 0 && Column[i+1].net != 0 &&
              Column[i-2].net != 0 && Column[i+2].net != 0 )
           continue;
      *x = i;
      *found = TRUE;
      }
  }
return(reset_warning());
}


/*************************************************************************/
int                 legal_step4(int line)
{
int    col = 0;
```

```
NET    *net = 0;
int    x=0, found=0;
int    i = 0;
for (col = 0; col < XSize; ++col )
 if (Column[col].net != 0)
  if (Column[col].net->split == 1)
   if (abs(Column[col].target_x - col) > 1)
    {
    /*----- scan for the position of the second pin */
    net = Column[col].net;
    CALL( scan_step4(line, net, col, &x, &found),
      set_error(8,ilegal_message,line,0) );
    if (found == TRUE)
      legal_move(net,line, col,x,TRUE);
    }
return(reset_warning());
}

/*************************************************************************/
int                 scan_step4(int line, NET *net, int col, int *x, int *found)
{
int                 i = 0;
*found = FALSE;
if (Column[col].target_x > col )
  {
  for (i=col; i < XSize && i <= Column[col].target_x &&
       Column[i].layer_used == FALSE ; ++(i) )
    if (Column[i].net == 0 && i != col )
      {
      if (i < XSize-1 && i > 0)
        if ( Column[i-1].net != 0 && Column[i+1].net != 0)
          continue;
      *x = i;
      *found = TRUE;
      }
  }
else
  {
  for (i=col; i > 0 && i >= Column[col].target_x &&
       Column[i].layer_used == FALSE ; --(i) )
    if (Column[i].net == 0 && i != col)
      {
      if (i < XSize-1 && i > 0)
        if ( Column[i-1].net != 0 && Column[i+1].net != 0)
          continue;
      *x = i;
      *found = TRUE;
      }
  }
return(reset_warning());
}

/*************************************************************************/
int    read_netlist(char * file)
{
FILE *fp = 0;

CALL0( insert_ilegal_functions() );
if (!(fp = fopen(file,"r")))
  return( set_error(2,ilegal_message,0,file) );
CERR( ufds_read_path(fp,file,0),
  set_error(4,ilegal_message,0,file) );

return(reset_warning());
}

/*************************************************************************/
int    store_path          (NET* net, int x, int y, char *path, char layer)
{
ROUTING      *rot = 0;
AMEM(rot,ROUTING,1,0);
AMEM(rot->path,char,strlen(Buffer) + 1,0);
strcpy(rot->path,path);
```

```
rot->x = x;
rot->y = y;
rot->layer = layer;
rot->next = net->routing;
net->routing = rot;
return(reset_warning());
}

/*************************************************************************/
int    save_routing(NET* netlist,char *file,char *file2)
{
FILE         *fp = 0;
FILE         *fp1 = 0;
FILE         *fp2 = 0;
NET          *net = 0;
ROUTING      *rot = 0;
int          elements = 0;
int          number = 0;

fprintf(stderr,"Number of connections:\t%d\n",Number_of_nets);
fprintf(stderr,"Missing connections:\t%d\n",Missing_nets);
fprintf(stderr,"Saving output files...\n");
if (!(fp1 = fopen(file,"w")))
  exit(0);
if (!(fp2 = fopen(file2,"w")))
  exit(0);
for (net = Netlist, number = 0; net; net = net->next, ++number)
  {
  if (net->alive = TRUE) fp = fp1; else fp = fp2;
  for (rot = net->routing, elements = 0; rot; rot = rot->next,++elements );
  if (elements == 0 )
    continue;
  fprintf(fp,"%d %s 1 %d (",number, net->name, elements);
  for (rot = net->routing, elements = 0; rot; rot = rot->next,++elements )
    {
    if (elements != 0 )
      fprintf(fp,";");
    fprintf(fp,"%c %d %d %s",rot->layer,rot->x,rot->y,rot->path);
    }
  fprintf(fp," );\n");
  }
fclose(fp1);
fclose(fp2);
return( reset_warning() );
}

/****** UFDS FUNCTIONS ***************************************************/

int    insert_ilegal_functions    ( void )
{
CALL0( ufds_register("area", area,0,0) );
CALL0( ufds_register("net", net,0,0) );
return( reset_warning() );
}

/*************************************************************************/
int    area                ( ARG_HEAD *arg_list )
{
CALL0( arg_get_int(arg_list,&(XSize)));
CALL0( arg_get_int(arg_list,&(YSize)));
AMEM(Terminal,PIN*,YSize,0);
AMEM(Column,COLUMN,XSize,0);
return(reset_warning());
}

/*************************************************************************/
int    net                 ( ARG_HEAD *arg_list )
{
NET *new_net = 0;
PIN *new_pin = 0;
int temp;
/*----- create and store net */
AMEM(new_net,NET,1,0);
```

```
CALL0( arg_get_string(arg_list,&new_net->name));
CALL0( arg_get_int(arg_list,&(new_net->x1)));
CALL0( arg_get_int(arg_list,&(new_net->y1)));
CALL0( arg_get_int(arg_list,&(new_net->x2)));
CALL0( arg_get_int(arg_list,&(new_net->y2)));
new_net->next = Netlist;
new_net->alive = TRUE;
++Number_of_nets;
Netlist= new_net;
/*----- store the pins by increasing y */
if (new_net->y2 < new_net->y1 || (new_net->y2 == new_net->y1 && new_net->x2 <
new_net->x1 ))
  { temp = new_net->y2; new_net->y2 = new_net->y1; new_net->y1 = temp;
    temp = new_net->x2; new_net->x2 = new_net->x1; new_net->x1 = temp; }
/*----- create and store second pin */
AMEM(new_pin,PIN,1,0);
new_pin->x = new_net->x2;
new_pin->y = new_net->y2;
new_pin->net = new_net;
new_pin->next = Terminal[new_pin->y];
Terminal[new_pin->y] = new_pin;
/*----- create and store first pin */
AMEM(new_pin,PIN,1,0);
new_pin->x = new_net->x1;
new_pin->y = new_net->y1;
new_pin->net = new_net;
new_pin->next = Terminal[new_pin->y];
Terminal[new_pin->y] = new_pin;
return(reset_warning());
}

/*****************************************************************************/
int    ilegal_message( int number, char **msg )
{
switch (number)
  {
case 0: *msg="ILEGAL"; return(0);
case 1: *msg="Cannot allocate memory.\n"; break;
case 2: *msg="%dCannot open file name \"%s\".\n"; break;
case 3: *msg="Missing input and output file names.\n"; break;
case 4: *msg="%dError reading file \"%s\".\n"; break;
case 5: *msg="Pin position x=%d out of range.\n"; break;
case 6: *msg="Pin position y=%d out of range.\n"; break;
case 7: *msg="No columns available to insert new pin at line %d.\n"; break;
default:*msg="Undefined error number %d with string \"%s\".\n"; break;
  };

switch (number)
  {
case 7:return(WARNING);
default:return(RUN_ERROR);
  };
}

/******* END ********************************************************************/
```

Bibliografia

[AKE 67] AKERS, S. B. A Modification of Lee's Path Connection Algorithm. **IEEE Transactions on Electronic Computers**, New York, v.16, n. 1, p.97-98, Feb. 1967.

[BAK 90] BAKOGLU, H. B. **Circuits, Interconnections, and Packaging for VLSI.** Reading, Mass: Addison-Wesley, 1990.

[BAR 2000] JOHN BARR, Needham. **EDA and the Semiconductor Cycle.** Disponível em: <http://www.electronicnews.com/enews/Issue/RegisteredIssues/2000/06052000/z40f-1.asp>. Acesso em: 31 out. 2000.

[BRA 98] BRAND, A. et al**.** Intel's 0.25 Micron, 2.0Volts Logic Process Technology. **Intel Technology Journal**, 3rd quarter 1998 Disponível em: <http://www.intel.com/technology/itj/q31998/articles/art_1.htm>. Acesso em: 31 out. 2000.

[BUR 83a] BURSTEIN, M.; PELAVIN, R. Hierarchical channel routing. In: ACM/IEEE DESIGN AUTOMATION CONFERENCE, 20.,1983. **Proceedings...** New York: IEEE, 1983. p.591-597.

[BUR 83b] BURSTEIN, M.; PELAVIN, R. Hierarchical wire routing. **IEEE Transactions on Computer-Aided Design of ICs and Systems,** New York, v. CAD-2, n. 4, p.223-234, Oct. 1983.

[BUS 97] BUSHROE, R. G. et al. Chip Hierarchical Design System (CHDS): a foundation for timing-driven physical design into the 21st century. In: INTERNATIONAL SYMPOSIUM ON PHYSICAL DESIGN, 2., 1997, Napa Valley. **Proceedings...** New York: ACM SIGDA, 1998. CD-ROM.

[CAL 96] CALDWELL, A.; KAHNG, A. **A Study of Three-Dimensional Gridded A*-Based Routing.** Los Angeles: UCLA VLSI-CAD Lab., 1996. Documento interno não publicado.

[CAR 96] CARRO, Luigi et al. An Environment to Design Digital Circuits Based on the Brazilian Gate Array. In: WORKSHOP IBERCHIP, 2., 1996, São Paulo. **Anais...** São Paulo: LSI/USP, 1996. p. 198-205.

[CAR 97] CARRO, Luigi et al. Ambiente ÁGATA de Projeto Versão Beta 2.0. In: WORKSHOP IBERCHIP, 3., 1997, México. **Anais...** México: Departamento de Ingeneria Electrica, 1997. p.494-503.

[CHA 77] CHAMPEAUX, Dennis de; SINT, Lenie. An Improved Bidirectional Heuristic Search Algorithm. **Journal of the Association for Computer Machinery**, New York, v. 24, n. 2, p.177-191, Apr. 1977.

[CON 90] CONG, Jason; PREAS, B.; LIU, C. L. General Models and Algorithms

for Over-the-Cell Routing in Standard Cell Design. In: ACM/IEEE DESIGN AUTOMATION CONFERENCE, 27., Orlando, US. **Proceedings...** New York: IEEE, 1990. p.709-715.

[CON 97] CONG, Jason. **Challenges and Opportunities for Design Innovatinos in Nanometer Technologies**. Disponível em: <http://ballade.cs.ucla.edu/~cong/talks.html>. Acesso em: 28 fev. 2001.

[DAV 98] DAVIS, Jeffrey A.; DE, Vivek K.; MEINDL, James D. A Stochastic Wire-Length Distribution for Gigascale Integration (GSI) – part I: derivation and validation. **IEEE Transactions on Electron Devices**, New York, v. 45, n. 3, p.580-589, Mar. 1998.

[DEC 85] DECHTER, R.; PEARL, J. Generalized Best-First Search Strategies and the Optimality of A*. **Journal of the Association for Computing Machinery**, New York, v. 32, p.505-536, 1985.

[DEU 76] DEUTSCH, David N. A Dogleg Channel Router. ACM/IEEE DESIGN AUTOMATION CONFERENCE, 13., 1976. **Proceedings...** New York: IEEE, 1976. p.425-433.

[DEU 80] DEUTSCH, David N.; GLICK, Paul. An Over-the-Cell Router. ACM/IEEE DESIGN AUTOMATION CONFERENCE, 19., 1980. **Proceedings...** New York: IEEE, 1980. p.32-39.

[DIL 94] DILLENBURG, J. F.; NELSON, P.C. Perimeter Search. **Artificial Intelligence**, Amsterdam, v. 65, p.165-178, 1994.

[DIL 95] DILLENBURG, J. F.; NELSON, P.C., Improving Search Efficiency Using Possible Subgoals. **Mathematical and Computer Modelling**, [S.l.], v. 22, n. 4-7, p.397-414, 1995.

[DOR 81] DOREAU, Michel T.; KOZIOL, Piotr. T W I G Y : A Topological Algorithm Based Routing System. In: DESIGN AUTOMATION CONFERENCE, 18., 1981. **Proceedings...** New York: IEEE, 1981. p.746-755.

[ECK 94] ECKERLE, Jurgen, OTTMANN, Thomas. An Efficient Data Structure for Bidirectional Heuristic Search. In: EUROPEAN CONFERENCE ON ARTIFICIAL INTELLIGENCE, 11., 1994. **Proceedings...** [S.l.]: John Wiley & Sons, 1994. p. 600-604.

[EDA 97] ELECTRONIC DESIGN AUTOMATION CONSORTIUM. **1996 EDA Revenues Top 2.3 Billion.** Disponível em: <http://www.edac.org/EDAC/News/EDAC17.html>. Acesso em: 20 set.1997.

[EDE 97] EDELSTEIN, D. et al. Full Copper Wiring in a Sub-0.25μm CMOS ULSI Technology. In: IEEE INTERNATONAL ELECTRON DEVICES MEETING (IEDM), Washington, 1997. **Proceedings...** [S.l.]: IEEE, 1997. p.773-776.

[EVE 79] EVEN, Shimon. **Graph Algorithms**. Rockville: Computer Science Press, 1979.

[GAO 94] GAO, Tong; LIU, C. L. Minimum Crosstalk Switchbox Routing. In: IEEE/ACM INTERNATIONAL CONFERENCE ON COMPUTER AIDED DESIGN, 1994, San Jose, CA. **Proceedings...** Los Alamitos: IEEE Press, 1994. p.610.

[GHO 91] GHOSH, S.; MAHANTI, A. Bidirectional Heuristic Search with Limited Resources. **Information Processing Letters**, Amsterdam, v. 40, p.335-340, 1991.

[GUN 95] GÜNTZEL, J.; REIS, R. ASIC Design Using a Sea-of-Cells Approach. In: EUROPEAN DESIGN AND TEST CONFERENCE USER FORM, 1995. **Proceedings...** Los Alamitos: IEEE, 1995. p.255.

[HAM 84] HAMACHI, G. T.; OUSTERHOUT, J. K. A Switvhbox Router with Obstacle Avoidance. In: ACM/IEEE DESIGN AUTOMATION CONFERENCE, 21., 1984. **Proceedings...** New York: IEEE, 1984. p.173-179.

[HAR 68] HART, P.E.; NILSSON, N.J.; RAPHAEL, B. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. **IEEE Transactions on Systems, Science and Cybernetics**, New York, v. SSC-4, p.100-107, 1968.

[HAS 71] HASHIMOTO, A.; STEVENS, J. Wire Routing by Optimizing Channel Assignment within Large Apertures. In: DESIGN AUTOMATION WORKSHOP, 8., 1971, Atlantic City. **Proceedings...** New York: ACM, 1971. p.155-163.

[HIG 83] HIGHTOWER, D. W. The Lee Router Revisited. In: INTERNATIONAL CONFERENCE ON COMPUTER DESIGN, 1983. **Proceedings...** New York: IEEE, 1983. p. 136-139.

[HEN 99] HENTSCHKE, R.; JOHANN, M.; REIS, R. A Specialized Assignment Algorithm for the GAROTA Router. In: UFRGS MICROELECTRONICS SEMINAR, 14., 1999, Pelotas. **Proceedings...** Porto Alegre: UFRGS, 1999. p. 63-66.

[HEN 2000] HENTSCHKE, R. **Um roteador Rede a Rede com o Algoritmo LCS***. Porto Alegre: PPGC/UFRGS, 2000. Projeto de Diplomação.

[HOE 76] HOEL, J. H. Some Variations of Lee's Algorithm. **IEEE Transactions on Computers**, New York, v.c-25, n.1, Jan. 1976.

[INT 2000] INTEL MUSEUM. What is Moore´s Law? Disponível em: <http://www.intel.com/intel/museum/25anniv/hof/moore.htm>. Acesso em: 07 nov. 2000.

[ISH 95] ISHIDA, Toru. Two is not Always Better than One: experiences in real-

time bidirectional search. In: INTERNATIONAL CONFERENCE ON MULTI-AGENT SYSTEMS, ICMAS, 1., 1995, San Francisco, U.S. **Proceedings...** Menlo Park: AAAI Press/The MIT Press, 1995. p.185-192.

[JOH 94] JOHANN, Marcelo O. **Roteamento sobre Células Transparentes**. Porto Alegre: Curso de Pós-Graduação em Ciência da Computação/UFRGS, 1994. Dissertação de Mestrado.

[JOH 95] JOHANN, M.; REIS, R. A Full Over-the-Cell Routing Model. In: IFIP INTERNATIONAL CONFERENCE ON VERY LARGE SCALE INTEGRATION, VLSI, 8., 1995, Chiba, Japan. **Proceedings...** [S.l.:s.n.] 1995.

[JOH 97a] JOHANN, M. **Estrutura de Roteamento em Circuitos VLSI**. Porto Alegre: CPGCC da UFRGS, 1997. 80p. (EQ-15).

[JOH 97b] JOHANN, M. **Projeto Funcional do Roteador GAROTA**: trabalho individual. Porto Alegre: CPGCC da UFRGS, 1997. 54p. (TI-649).

[JOH 97c] JOHANN, M.; CARRO, L.; REIS, R. Functional Design of GAROTA: gate array router of ÁGATA system. In: BRAZILIAN SYMPOSIUM ON INTEGRATED CIRCUIT DESIGN, 10., Gramado, 1997 **Proceedings...** Porto Alegre: CPGCC da UFRGS, 1997. p. 21-30.

[JOH 96] JOHANN, M.; MÜHLEN, D.; REIS, R. UFDS: a practical way of storing and reading data for flexible prototyping of CAD systems. In: UFRGS MICROELECTRONICS SEMINAR, 11., 1996. **Proceedings...** Porto Alegre: CPGCC da UFRGS, 1998. p. 91-96.

[JOH 98] JOHANN, M.; REIS, Ricardo. GAROTA. Version Beta 2.3. In: UFRGS MICROELECTRONICS SEMINAR, 13., 1998. **Proceedings...** Porto Alegre: CPGCC da UFRGS, 1998. p. 95-98.

[JOH 2000a] JOHANN, M. et al. A New Bidirectional Heuristic Shortest Path Search Algorithm. In: INTERNATIONAL ICSC CONGRESS ON ARTIFICIAL INTELLIGENCE AND APPLICATIONS, 2000, Wollongong, Australia. **Proceedings...** [S.l:s.n.], 2000. CD-ROM.

[JOH 2000b] JOHANN, M. et al. Admissibility Proofs for the LCS* Algorithm. In: BRAZILIAN ARTIFICIAL INTELLIGENCE SYMPOSIUM, SBIA, 15., 2000, Atibaia. **Proceedings...** Berlin: Springer-Verlag, 2000. p.236-244. (Lecture Notes on Artificial Intelligence, 1952).

[JOH 2000c] JOHANN, M.; CARRO, L.; REIS, R. Automatic Master-Slice Generation with GAROTA version Beta 2.4. In: WORKSHOP IBERCHIP, 6., 2000, São Paulo. **Anais...** [Campinas:CTI], 2000.

[JOO 86] JOOBBANI, R.; SIEWIOREK, D. WEAVER: A Knowledge-based Routing Expert. **IEEE Design & Test of Computers**, New York, v. 3, n. 1, p.12-23, Feb. 1986.

[KAH 95] KAHNG, Andrew B.; ROBINS, Gabriel. **On Optimal Interconnections for VLSI**. Norwell: Kluwer, 1995. 286p.

[KAH 99] KAHNG, Andrew B. et al. Interconnect Tunning Strategies for High-Performance ICs. In: DESIGN AUTOMATION AND TEST IN EUROPE CONFERENCE, 1999. **Proceedings...** [Los Alamitos: IEEE Computer Society], 1999.

[KAI 95] KAINDL, Hermann et al. How to Use Limited Memory in Heuristic Search, In: INTERNATIONAL JOINT CONFERENCE ON ARTIFICIAL INTELLIGENCE, 14., 1995, Montreal. **Proceedings...** Montreal: AAAI, 1995. v.1, p.236-242.

[KAI 96] KAINDL, Hermann; KAINZ, Gerhard. Dynamic Improvements of Heuristic Evaluations during Search, In: NATIONAL CONFERENCE ON ARTIFICIAL INTELIGENCE, 13., 1996. **Proceedings...** Menlo Park: AAAI, 1996. v.1, p.311-317.

[KAI 97] KAINDL, Hermann; KAINZ, Gerhard. Bidirectional Heuristic Search Reconsidered. **Journal of Artificial Intelligence Research (JAIR)**, [S.l.], v. 7, p. 283-317, Dec. 1997. Disponível em: <http://www.cs.washington.edu/research/jair/abstracts/kaindl97a.html>. Acesso em: 10 set. 1999.

[KAO 95] KAO, Wen-Chung; PARNG, Tai-Ming. Cross Point Assignment With Global Rerouting for General Architecture Designs. **IEEE Transactions on CAD of ICs and Systems**, Los Alamitos, v.14, n.3, p.337, 1995.

[KIR 94] KIRKPATRICK, Desmond A.; SANGIOVANNI-VICENTELLI, Alberto L. Techniques for Crosstalk Avoidance in the Physical Design of High-Performance Digital Systems. In: IEEE/ACM INTERNATIONAL CONFERENCE ON COMPUTER AIDED DESIGN, 1994, San Jose, CA. **Proceedings...** Los Alamitos: IEEE Press, 1994. p.616.

[KOL 93] KOLL A. L.; KAINDL, H. Bidirectional Best-First Search with Bounded Error: Summary of Results, In: THE INTERNATIONAL JOINT CONFERENCE ON ARTIFICIAL INTELLIGENCE**,** 1993. **Proceedings...** [S.l:s.n], 1993. p.217-223.

[KOR 82] KORN, R. K. An Efficient Variable-Cost Maze Router. ACM IEEE DESIGN AUTOMATION CONFERENCE, 19., 1982, Las Vegas, Nevada. **Proceedings...** New York: IEEE, 1982.

[KOR 93] KORF, Richard. E. Linear-space best-first search. **Artificial Intelligence,** Amsterdam, v. 62, p.41-78, 1993.

[KWA 89] KWA, J. B. H. BS*: An Admissible Bidirectional Staged Heuristic Search Algorithm. **Artificial Intelligence**, Amsterdam, v. 38, p.95-109, 1989.

[LEE 61] LEE, C. An Algorithm for Path Connections and It's Applications. **IEEE**

Transactions on Electronic Computers, New York, v.ec-10, p.346-365, Sept. 1961.

[MAN 95] G. Manzini. BIDA*: An Improved Perimeter Search Algorithm. **Artificial Intelligence**, Amsterdam, v.75. p.347-360, 1995.

[MOO 65] MOORE, G. E. Cramming More Components onto Integrated Circuits. **Electronics Magazine**, [S.l.], v.38, p.114-117, Apr. 1965.

[NAI 87] NAIR, Ravi. A Simple Yet Effective Technique for Global Wiring. **IEEE Transactions on Computer-Aided Design**, New York, v.CAD-6, n.2. p.165-172, Mar. 1987.

[NEL 92] NELSON, P. C.; TOPSIS, A. A. Unidirectional and Bidirectional Search Algorithms. **IEEE Software**, Los Alamitos, v. 9, p.77-83, Mar. 1992.

[NIL 71] NILSSON, N. J. **Problem-Solving Methods in Artificial Intelligence**. New York: McGraw-Hill, 1971. p.43--79.

[PEA 84] PEARL, Judea. **Heuristics - Intelligent Search Strategies for Computer Problem Solving.** Reading: Addison-Wesley, 1984. p.73-87.

[POH 69] POHL, I. **Bi-Directional Search and Heuristic Search in Path Problems**. Stanford, CA: Stanford University, 1969. PhD thesis. (SLAC Report n.104).

[POH 70] POHL, I. First Results on the Effect of Error in Heuristic Search. **Machine Intelligence**, Edinburgh, v.5, p.219-236, 1970.

[POH 71] POHL, I. Bi-Directional Search. **Machine Intelligence**, Edinburgh, v.6, p.127-140, 1971.

[POL 84] POLITOWSKI, G.; POHL, I. D-node Retargeting in Bidirectional Heuristic Search, In: THE NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE**,** 1984. **Proceedings...** [S.l:s.n], 1984. p.274-277.

[PRE 88] LORENZETTI, Michael J.; BAEDER, D. Scott. Routing. In: PREAS, B. T. **Physical Design Automation of VLSI Systems**. Menlo Park: Benjamim/Cummings, 1988. p.157-210.

[RYA 95] RYAN, J. Get al. The evolution of interconnection technology at IBM. **IBM Journal of Research & Development**, [S.l.] v. 39, n. 4. p.371, 1995. Disponível em: <http://www.research.ibm.com/journal/rd/ryan/ryan.html>. Acesso em: 31 out. 2000.

[RAH 95] RAHMAT, K. et al. A Scaling Scheme for Interconnect in Deep-Submicron Processes. In: IEEE INTERNATIONAL ELECTRON DEVICE MEETING. **Proceedings...** [S.l.]:IEEE, 1995. p.245-248.

[RIV 82] RIVEST, Ronald L.; FIDUCCIA, Charles M. A Greedy Channel Router.

In: DESIGN AUTOMATION CONFERENCE, 19., 1992, Las Vegas. **Proceedings...** New York: ACM/IEEE, 1982. p.418-424.

[RUB 74] RUBIN, F. The Lee Path Connection Algorithm. **IEEE Transactions on Computers**, New York, v.c-23, n.9, p.907-914, Sept. 1974.

[SAD 95] SADOWSKA, Margozata M.; SARRAFZADEH, Majid. The Crossing Distribution Problem. **IEEE Transactions on CAD of ICs and Systems**, Los Alamitos, v.14, n.4, p.423, 1995.

[SHE 93] SHERWANI, Naveed. **Algorithms for VLSI Physical Design Automation**. Massachusetts: Kluwer, 1993. 538p.

[SHE 95] SHERWANI, Naveed; BHINGARDE, Siddharth; PANYAM, Anand. **Routing in the Third Dimension:** from VLSI chips to MCMs. New York: IEEE, 1995.

[SIA 97] SEMICONDUCTOR INDUSTRY ASSOCIATION (SIA). **The National Technology Roadmap for Semiconductors**. [S.l.], 1997.

[SOU 78] SOUKUP, J. Fast Maze Routers. In: DESIGN AUTOMATION CONFERENCE, 15., 1978. **Proceedings...** New York: IEEE, 1978. p.100-102.

[STA 98] STAMPER, Anthony K. Interconnection Scaling to 1 GHz and Beyond. **MicroNews, IBM Microelectronics**, [S.l.], v.4, n.2, 1998. Disponível em: <http://www.chips.ibm.com/micronews/vol4_no2/interconnection.html>. Acesso em: 03 nov. 2000.

[TAD 80] TADA, F. K.; YOSHIMURA, K.; KAGATA, T. et al. A Fast Maze Router with Iterative Use of Variable Search Space Restriction. In: DESIGN AUTOMATION CONFERENCE, 17., 1980. **Proceedings...** New York: IEEE, 1980. p.250-254.

[THE 2000] THEIS, T. N. The future of interconnection technology. **IBM Journal of Research and Development**, [S.l.], v.44, n.3, 2000. Disponível em: <http://www.research.ibm.com/journal/rd/443/theis.html>. Acesso em: 03 nov. 2000.

[TIN 83] TING, B. S.; TIEN, B. N. Routing Techniques for Gate Array. **IEEE Transactions on Computer-Aided Design**, New York, v.CAD-2, n.4, p.301-312, Oct. 1983.

[TSU 81a] TSUI, Raymond Y.; SMITH, Robert, J. A High-Density Multilayer PCB Router Based on Necessary and Sufficient Conditions for Single Row Routing. In: ACM/IEEE DESIGN AUTOMATION CONFERENCE, 18., 1981. **Proceedings...** New York: IEEE, 1981. p.372-381.

[TSU 81b] TSUKIYAMA, Shuji; KUH, Ernest S.; SHIRAKAWA, Isao. On The Layering Problem of Multilayer PWB Wiring. In: ACM/IEEE DESIGN

AUTOMATION CONFERENCE, 18., 1981. **Proceedings...** New York: IEEE, 1981. p.738-745.

[VEM 91] VEMPATI, Nageshwara Rao; KUMAR, Vipin; KORF, Richard E. Depth-First vs Best-First Search. In: NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE, 9. ,1991. **Proceedings...** Menlo Park: AAAI/MIT, 19991. p. 434-440.