Traçado de Rotas em Circuitos Impressos Multi-face

Este exemplar corresponde à redação final da tese devidamente corrigida e defendida pelo Sr. Luiz Eduardo Buzato e aprovada pela comissão julgadora.

Campinas, 7 de março de 1990.

Orientador: _____

Prof. Dr. Hans Kurt Edmund Liesenberg

Dissertação apresentada ao Instituto de Matemática, Estatística e Ciência da Computação, UNICAMP, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

Traçado de Rotas em Circuitos Impressos Multi-Face[1]

Luiz Eduardo Buzato

Orientador
Prof. Dr. Hans Kurt Edmund Liesenberg

Dissertação apresentada ao Instituto de Matemática, Estatística e Ciência da Computação como requisito parcial para obtenção do título de Mestre em Ciência da Computação.

[1]Este trabalho contou com o apoio financeiro da Sisgraph Ltda.

Aos meus pais, e a
$\mathcal{SB}$, $\mathcal{AB}$,
$\mathcal{RB}^2$,
$\mathcal{CBC}$, $\mathcal{MJB}$,
e $\mathcal{Y}$.
A todos os que se cansaram de me ouvir pronunciar:
"Está quase pronta!".

Conteúdo

Lista de Figuras

Lista de Tabelas

Introdução

Estrutura do Trabalho

O texto é composto por seis Capítulos. Os Capítulos iniciais, 1 e 2, abordam aspectos complementares relacionados ao problema de traçado de rotas. No Capítulo 1 são descritos vários aspectos tecnológicos envolvendo o projeto auxiliado por computador e o processo de fabricação de circuitos impressos. O Capítulo 2 introduz o problema de traçado de rotas sob uma perspectiva teórica, descrevendo os principais algoritmos relacionados ao problema, com um breve estudo de sua complexidade. O resultado obtido ao final deste estudo justifica a procura de soluções heurísticas para o problema, descritas no Capítulo 3. Os Capítulos 4 e 5 tratam, respectivamente, da descrição, implementação e teste do algoritmo proposto neste trabalho. Os resultados obtidos encontram-se no Capítulo 6.

Traçado de Rotas para Circuito Impresso

A tecnologia de *circuitos impressos* baseia-se no emprego de um conjunto de *faces isolantes* e *faces condutoras* superpostas de modo intercalado, formando um laminado denominado *placa*. As faces condutoras são formadas pela deposição de material condutor sobre as faces isolantes[1]. *Furos metalizados* são furos revestidos de material condutor que atravessam perpendicularmente trechos da placa. Estes furos têm duas finalidades básicas:

- Fixação de componentes: os componentes eletrônicos que compõem o circuito são fixados na placa através de furos metalizados que alojam seus pinos. Exceção feita aos *componentes de montagem superficial (CMS)*[2] que dispensam furos metalizados em sua fixação.
- Interligação de condutores pertencentes a planos condutores diferentes: um furo metalizado pode, se necessário, interligar segmentos de condutores pertencentes a faces condutoras diferentes.

Denomina-se *sinal* a um conjunto de pinos eletricamente equivalentes, sendo os pinos interligados por condutores depositados sobre as faces isolantes. Condutores correspondentes a sinais distintos não podem se cruzar em uma mesma face condutora da placa, pois isto resultaria um curto-circuito entre os sinais envolvidos.

[1] Faces condutoras são filmes metálicos, faces isolantes são, normalmente, placas de fibra de vidro (Capítulo 1).

[2] Em Inglês: Surface Mounted Device (SMD).

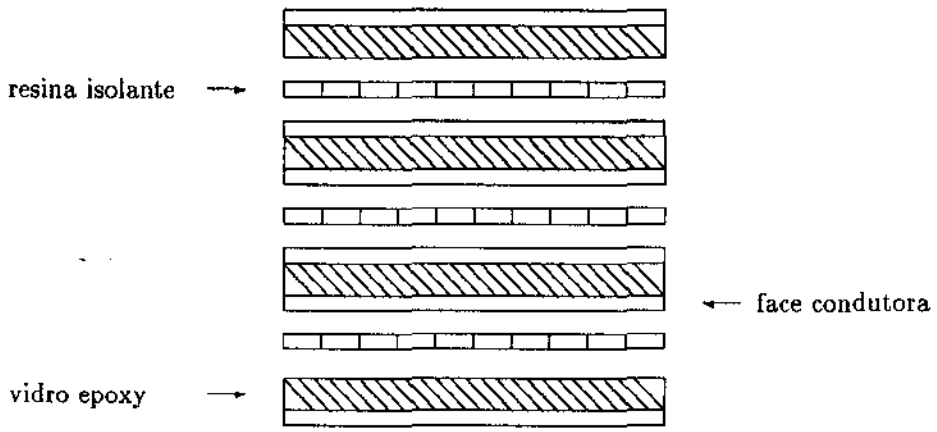


Figura I.1: Placa de Circuito Impresso Multi-Face.

Placas de circuitos impressos são normalmente designadas pelo número de faces condutoras que contém, assim uma placa *dupla-face* contém duas faces condutoras. Placas com um número de faces maior que dois são denominadas placas *multi-face* (Figura I.1).

No início, as placas de circuito impresso eram construídas manualmente devido a baixa quantidade de componentes utilizados, em geral, somente componentes discretos (por ex.: transistores, resistores, capacitores, etc.) [Bre 72], [Lin 79]. A introdução de circuitos integrados[3] alterou sensivelmente a densidade de conexões a serem realizadas, quase inviabilizando a construção manual de tais placas. O projeto manual de uma placa muito densa é lento e suscetível a erros. Isto eleva os custos de produção a níveis que justificam a automação do processo. Nos últimos anos, a escala de integração de circuitos integrados tem aumentado muito rapidamente [Lie 86], tornando a densidade das placas projetadas com estes circuitos tão alta que mesmo os sistemas computadorizados

[3]Combinação de um número de componentes discretos, tais como resistores, capacitores e transistores em um circuito completo construído em um pedaço único de material semicondutor contendo dispositivos ativos e passivos, bem como suas interconexões. Estes circuitos são acondicionados em encapsulamentos com pinos no lado externo que estabelecem a interface do circuito encapsulado com o exterior.

de auxílio ao projeto de circuitos impressos têm dificuldade em resolver todas as conexões necessárias. Antes, placas de face única ou dupla-face mostravam-se totalmente adequadas à implementação dos circuitos. Atualmente, devido à miniaturização dos circuitos a utilização de placas multi-face tornou-se indispensável para acomodar a alta densidade de componentes e conexões existente.

Caracterização do Problema

O problema do traçado de rotas pode ser caracterizado como: "Fornecidos os conjuntos de sinais a serem interligados e a localização dos pinos dos componentes na placa, determinar a distribuição espacial dos condutores que implementam as interligações, respeitando-se as restrições impostas pela tecnologia empregada" [Hig 74], [Lie 80].

Capítulo 1

Aspectos Tecnológicos

1.1 Projeto de Circuitos Impressos

Todo sistema eletrônico complexo pode ser decomposto em circuitos mais simples, módulos, que interligados adequadamente têm comportamento funcional idêntico ao do sistema original. No projeto dos módulos emprega-se geralmente uma metodologia de refinamentos sucessivos [Lie 86]. Parte-se da especificação funcional do módulo e a cada nova fase de projeto define-se com maior detalhe o circuito até a obtenção do esquema[1] final. Cada um desses módulos tem funções bem definidas e pode ser isolado para análise.

Sistemas eletrônicos assim projetados permitem verificações parciais durante a fabricação, agilizando-a e reduzindo o número de falhas de produção. Durante a operação do sistema, o teste, reparo e alteração seriam facilitados se os módulos fossem fabricados em separado e depois interligados através de cabos e conectores apropriados, refletindo diretamente o definido no projeto. Circuitos impressos são apropriados para produzir circuitos modulares pois permitem uma interação eficiente entre diversos tipos de componentes e uma simples fixação de conectores, além de ter um custo de produção relativamente baixo. A maior evidência deste fato é o grande uso de circuitos impressos na indústria eletro-eletrônica mesmo após o advento dos circuitos integrados.

A avaliação de parâmetros tais como custo, tempo de projeto, flexibilidade de reconfiguração, confiabilidade, desempenho, tamanho do lote produzido, escala de miniaturização desejada, etc, tem indicado que um compromisso razoável é atingido quando circuitos integrados são empregados na produção de circuitos básicos, de largo espectro de aplicação, ou de circuitos personalizados/semi-personalizados[2] para aplicações específicas sendo os circuitos impressos empregados como substrato para abrigar esses circuitos básicos e outros componentes eletrônicos. Em geral, esta combinação é a que permite uma maior flexibilidade de projeto. Justifica-se assim o contínuo esforço no desenvolvimento de algoritmos e ambientes integrados de desenvolvimento de circuitos impressos.

---

[1] Em Inglês: schematic. Um esquema é um diagrama composto por símbolos que representam componentes eletro-eletrônicos e linhas, representando condutores que interconectam esses componentes.

[2] Em Inglês: custom/semi-custom chips. Um circuito integrado projetado pelos fabricantes para uma finalidade específica, por ex.: um circuito para uma calculadora de bolso ou um termômetro digital.

Circuitos impressos são projetados levando-se em conta diversos fatores:

- características elétricas do circuito eletrônico (sinais):
  - ddp's[3],
  - correntes elétricas,
  - freqüências,
  - impedâncias,
  - capacitâncias.
- manutenção:
  - a nível de componente,
  - com o circuito em operação[4],
  - com o circuito fora de operação[5],
  - inexistente (circuito descartável).
- ambiente de operação:
  - amplitude térmica do ambiente,
  - presença de resíduos (pó, resíduos químicos, etc.),
  - umidade,
  - sujeito a vibrações, choques mecânicos.
- processo de fabricação:
  - automático,
  - manual,
  - tamanho do lote produzido.

[3] diferença de potencial elétrico.
[4] Em Inglês: on-line.
[5] Em Inglês: off-line.

- componentes e materiais utilizados:
  - custo,
  - disponibilidade,
  - tipo do componente (CMOS[6], NMOS[7],...)
  - características de consumo,
  - dissipação,
  - vida média do componente, etc.

O desempenho e confiabilidade finais do circuito impresso dependem da avaliação correta de fatores como os acima. A avaliação destes fatores não é simples quando o circuito impresso a ser projetado é complexo. Por este motivo foram desenvolvidos os sistemas de apoio a projeto de circuitos impressos auxiliados por computador.

1.2 Projeto de Circuitos Impressos Auxiliado por Computador

Os objetivos do projeto de circuitos impressos auxiliado por computador são a redução do custo e tempo envolvidos entre o início do projeto e o final da fabricação do circuito. Dentro de sistemas de projeto auxiliado por computador ([Int 84a], [Int 84b], [CDC 84]) o projetista pode criar e simular esquemas, projetar placas de circuito impresso utilizando diversas tecnologias e preparar interfaces para sistemas automáticos de fabricação (perfuradoras, insersores de componentes). É freqüente a integração do sistema de projeto de circuitos impressos com outros sistemas auxiliares importantes. Existem ambientes auxiliares para projeto mecânico, análise de processos de fabricação, edição de documentos técnicos, etc.

Outras características exibidas pelos sistemas de projeto incluem: produção de listas de material utilizado, avaliação dos custos de projeto, cronogramas e fluxogramas do processo de industrialização, possibilidade de atribuição de sinais a faces condutoras específicas, utilização de parâmetros para determinar a parada do processo de traçado automático após a resolução de um número pré-fixado de rotas, conexões de largura variável, rotinas especiais para traçado de sinais como barramentos de memória e outras estruturas regulares.

[6] Complementary Metal-Oxide-Semiconductor
[7] N-type silicon channel MOS

Alguns sistemas de projeto de circuitos impressos auxiliados por computador são instalados em computadores com hardware específico desenvolvido especialmente para abrigá-los [Int 86c]. Como resultado desta associação temos sistemas extremamente sofisticados e de alto custo. No Brasil, ainda não existem sistemas sofisticados de apoio ao projeto de circuitos impressos que se comparem aos produzidos no exterior. Faltam até mesmo sistemas compactos, simplificados, que realizem somente as fases de posicionamento e traçado.

O projeto do circuito impresso inicia-se pela conversão do esquema do circuito eletrônico num conjunto de dados adequados ao sistema de projeto auxiliado por computador. Findo o processo de aquisição e consistência dos dados passa-se a fase de posicionamento dos diversos componentes sobre a placa. Nesta fase são levados em conta parâmetros tais como: área útil da placa, sinais críticos, volume dos componentes e a possível interferência entre sinais de alta freqüência [Bre 72]. Os modos de posicionamento normalmente fornecidos pelos sistemas de projeto auxiliado por computador [Int 84a], [CDC 84] são:

- interativo: o projetista especifica interativamente a localização dos componentes. O software de posicionamento interativo imita o projeto manual de circuitos e pode ser usado como passo preliminar ao posicionamento adaptativo ou automático; normalmente é usado para o posicionamento de componentes críticos.

- posicionamento adaptativo: o projetista especifica a localização inicial dos componentes e a seguir o sistema faz a otimização das localizações automaticamente. O posicionamento adaptativo agiliza o posicionamento dos componentes pela seleção automática da mais "apropriada" posição de um componente. O posicionamento é uma função das conexões a serem feitas entre este componente e os demais componentes já posicionados.

- automático: o sistema posiciona os componentes automaticamente em função de parâmetros especificados pelo projetista. Guiado pelos parâmetros o software procura um posicionamento eficiente dos componentes, reduzindo muito a interação com o usuário.

O posicionamento é realizado para tentar-se atingir objetivos tais como: manter a densidade de conexões por canal de conexão[8] dentro de níveis aceitáveis, eliminar efeitos de "cross-talking" isolando sinais críticos, eliminar ecos

[8] Área livre existente entre os componentes após o término do posicionamento e utilizada para alojar conexões.

de sinal (reflexão) e controlar os níveis de dissipação de calor. Em [Int 84a] avalia-se inclusive o alojamento do circuito impresso em seu local de operação, por exemplo, um gabinete de um computador.

Terminada a fase de posicionamento o sistema extrai os dados a serem utilizados na fase seguinte: a de traçado. Para esta fase são necessários, basicamente, dois conjuntos de dados: uma descrição geométrica do circuito e uma lista de sinais. O traçado pode ser realizado de modo automático ou manual. No modo manual o projetista determina a rota que interliga pinos de componentes, sendo auxiliado pelo sistema na verificação das restrições de projeto. No modo automático um novo processo iterativo tem início onde o traçador de rotas tentará determinar qual a disposição espacial de cada rota minimizando o comprimento total da rota e o número de furos metalizados. Este objetivo deve ser alcançado sem que sejam desrespeitadas as restrições impostas ao circuito impresso.

Em alguns sistemas, o projetista pode escolher o método a ser utilizado na fase de interconexão dos componentes (traçado). Os componentes podem ser conectados utilizando-se distâncias definidas por interconexões em árvore[9] ou por interconexões encadeadas[10]. No Capítulo seguinte, a relação entre interconexões em árvore e o traçado de rotas será melhor delineada. Normalmente, árvores são utilizadas quando os pinos de componentes podem ter múltiplas interconexões. Encadeamentos são utilizados quando os pinos podem ter somente uma ligação de entrada e uma de saída; algumas vezes a utilização de encadeamento é obrigatória, por exemplo, quando se utilizam componentes ECL[11][Int 84a].

O processo termina, após várias iterações, com a obtenção da arte-final[12].

1.2.1 Interface com Equipamentos para Fabricação de Circuitos

Sistemas de projeto auxiliado por computador para circuitos impressos fornecem interfaces de software que permitem utilizar os dados gerados durante as diversas fases de projeto na programação de máquinas como: traçadores óticos, perfuradoras automáticas, insersores automáticos de componentes, testadores de circuitos e máquinas automáticas de "wire-wrap".

[9] Em Inglês: Minimum Spanning Trees.

[10] Em Inglês: Chain.

[11] Emiter-Coupled Logic.

[12] Em Inglês: art work; um desenho em escala aumentada dos diferentes layouts de faces condutoras. O desenho é reduzido para produzir uma máscara que será utilizada na impressão do layout do circuito sobre a face condutora.

- Traçadores óticos: os dados geométricos necessários para orientar traçadores óticos são extraídos dos dados definidos durante a fase de traçado e pós-processamento do circuito. Um arquivo é gerado para cada face da placa, incluindo dados sobre as coroas metalizadas, largura dos traços e os comandos para a programação do traçador ótico. Adicionalmente, os usuários podem obter arquivos para o traçado ótico do esquema do circuito, do layout dos componentes, máscaras para solda, máscaras para perfuração, etc.

- Perfuradoras de controle numérico: a interface para a perfuradora trabalha com informações obtidas durante a fase de posicionamento dos componentes e traçado. Nestas fases do projeto da placa os furos para circuitos integrados, componentes discretos e furos metalizados são determinados. O software de interface extrai e formata estes dados numa linguagem padrão que permite, virtualmente, operar qualquer perfuradora de controle numérico. Para reduzir o tempo de perfuração o software inclui funções para otimização do movimento da cabeça da perfuradora. Após a fase de geração do arquivo de comando numérico, o software produz um relatório com sumários do processo de perfuração que permitem estimar o tempo total gasto na produção das placas.

- Insersores automáticos de componentes: esta interface prevê a geração de informações para insersores de componentes axiais (resistores, diodos, etc.) e insersores de circuitos integrados. Os dados são extraídos da fase de posicionamento, onde as coordenadas necessárias para a inserção dos componentes são determinadas. Há procedimentos para a otimização do movimento da cabeça insersora. Para insersores de circuitos integrados o software gera um relatório determinando a posição dos magazines[13] na máquina; devem ficar mais próximos da cabeça de inserção os que contiverem os circuitos integrados mais utilizados. Os insersores axiais são, em geral, alimentados por fitas onde os componentes são fixados. Para otimizar a inserção o software gera a programação da máquina que produz a fita de componentes especificando a seqüência e a orientação dos componentes na fita. Os dados sobre o ponto de dobradura e corte dos contatos dos componentes axiais completam as informações necessárias para a programação dos insersores.

[13] Receptáculos de forma trapezoidal que alojam os componentes fornecidos ao insersor durante sua operação.

- Testador automático de placas: o software prevê testes funcionais para o circuito todo e também testes que isolam e verificam o funcionamento de cada componente na placa. Os dados para programar o testador são obtidos da lista de sinais e de informações sobre os componentes contidas nas bases de dados do sistema.

- "Wire-wrappers" automáticos: a interface gera as informações para operar máquinas automáticas de "wire-wrap" na construção de protótipos de placas. Às vezes, pequenos lotes de placas para teste são produzidos deste modo. A otimização do movimento de cabeça do "wire-wrapper" é importante para minimizar o tempo de produção.

1.2.2 Fases Típicas de Projeto

As fases de projeto usualmente executadas durante o projeto de um circuito impresso com o auxílio de um sistema computadorizado são
[Int 84a], [CDC 84]:

1. Configuração inicial das bases de dados

   A primeira fase de projeto consiste na configuração inicial das bases de dados de componentes e do dicionário de esquemas. O dicionário de esquemas permite ao projetista o fácil acesso à biblioteca de componentes/esquemas utilizando nomes comuns de componentes como chave de acesso ao invés dos números internos de classificação dos componentes. É possível também que o dicionário traga informações sobre o funcionamento do componente para facilitar simulações do circuito.

   Exemplo:

   especificação informal: resistor 1.2kΩ 1/4W 5%
   código interno: CRES12204

2. Geração da lista de sinais a partir do esquema lógico

   Terminado o projeto lógico do circuito, o primeiro passo no projeto da placa é a geração da lista de sinais. Esta lista de sinais é utilizada para converter os dados armazenados no arquivo de projeto em informações utilizáveis pelos próximos passos do projeto.

3. Captura do esquema (fase opcional)

   A lista de sinais pode opcionalmente ser introduzida no sistema neste ponto. Uma interface para captura de esquema é então acionada. As

fases mencionadas antes acontecem quando o projeto lógico do circuito foi realizado em um sistema computadorizado.

4. Verificação do esquema.

   Após o projeto do esquema estar completo e a lista de sinais gerada, a informação contida no esquema deve ser verificada para evitar qualquer inconsistência. O utilitário desenvolvido para este fim examina os dados do esquema procurando por falhas tais como:

   - campos de referência (texto) preenchidos por nomes inválidos,
   - validade do número de pino atribuído,
   - verificação, para cada componente, se todos ou nenhum dos pinos pertencem aos sinais existentes,
   - duplicação de pinos num mesmo componente,
   - número de pinos nos componentes está incorreto, etc.

5. Posicionamento de componentes.

6. Obtenção da lista de conexões.

   Após o posicionamento dos componentes no plano de traçado é possível gerar uma lista de sinais baseada nas informações geométricas agora existentes. A lista de sinais anterior é modificada. Para cada furo metalizado ou pino de componente são adicionadas as coordenadas que determinam sua posição na placa. A lista de sinais de conexões resultante contém os dados necessários à fase de traçado.

7. Geração de interconexões ponto-a-ponto.[14]

   O objetivo principal da geração de interconexões ponto-a-ponto é facilitar a visualização das conexões e a densidade relativa de conexões existente em uma determinada área da placa. Além disso, interconexões ponto-a-ponto facilitam a análise do posicionamento dos componentes.

8. Atribuição de interconexões ponto-a-ponto a faces condutoras.

   A atribuição de interconexões ponto-a-ponto a faces condutoras visa diminuir as chances de interferência entre sinais durante a fase de traçado. Um dos critérios utilizados para distribuir as interconexões ponto-a-ponto é a sua orientação relativa.

[14] Uma interconexão ponto-a-ponto é uma reta conectando dois pontos pertencentes a um mesmo sinal.

9. Traçado de rotas.

10. Otimização do número de furos metalizados.

    Nesta fase tenta-se reduzir o número de furos metalizados através do retraçado parcial de alguns segmentos de conexões visto que a realização de um furo implica custo enquanto que os segmentos de conexão são processados em conjunto. Assim sendo, cada furo adicional encarece uma placa mas segmentos de conexão adicionais não.

11. Consistência das restrições de projeto.

12. Geração das artes-finais e interfaces para máquinas de controle numérico.

1.3 Fabricação de Circuitos Impressos

Os padrões de fabricação de circuitos impressos têm evoluído rapidamente. Atualmente, diversas tecnologias podem ser utilizadas na sua fabricação (placas multi-face, placas "multiwire", etc). O aumento do número de tecnologias disponíveis para construção de circuitos impressos é um dos fatores que aumenta a complexidade e sofisticação dos sistemas de projeto auxiliado por computador, pois a tendência observada é a de fornecer-se ao projetista um sistema que englobe, senão todas, a maioria das tecnologias existentes.

Nas próximas seções apresenta-se um resumo do processo de fabricação de circuitos impressos empregando-se laminados, uma das tecnologias mais utilizadas atualmente.

1.3.1 Laminados

Nesta tecnologia as faces isolantes do circuito impresso são normalmente fabricadas utilizando papel impregnado de resinas ou fibra-de-vidro prensados. Sobre a superfície isolante é aplicado o filme de material condutor e assim obtem-se um laminado composto por uma face isolante e uma face condutora. Circuitos multi-face são construídos pela superposição destes laminados básicos [Coo 67].

Tratamento do Laminado Base

A primeira operação na manufatura de laminados é demoninada tratamento e consiste na impregnação do material base, papel ou fibra-de-vidro, com resina. A resina é então tratada até estar pronta para armazenamento e

posterior prensagem. O papel ou fibra-de-vidro passa através de um tanque de resina e depois por uma sequência de cilindros e finalmente por um forno de secagem. No forno, muitos dos resíduos voláteis da resina são retirados deixando-a num estado de tratamento intermediário.

Um processo rígido de controle é aplicado para controlar a quantidade de resina aplicada ao material base, a espessura final do laminado tratado e o estágio de tratamento da resina após o término do processo. Sensores comparam a espessura do material da entrada com a do material já processado e automaticamente ajustam o espaço entre os cilindros de modo a manter correta a quantidade de resina adicionada ao material base. O grau de tratamento da resina é controlado pela velocidade do material dentro do forno e pela umidade e temperatura do ar circulante em seu interior. Após este pré-processamento o material é armazenado em locais com umidade e temperatura controladas até ser utilizado na confecção da placa. As diversas combinações de resina-material base empregadas na confecção do laminado base determinarão as características finais da placa de circuito impresso. Por exemplo, a maior ou menor inflamabilidade das placas de circuito impresso é um dos fatores determinados.

Inspeção do Filme Condutor

O outro componente fundamental no preparo do laminado para a fabricação da placa de circuito impresso é o filme de material condutor. Atualmente o filme é fabricado pela eletrodeposição de metal sobre cilindros de aço de grande diâmetro. Cada rolo de filme metálico é examinado pelo laminador para verificação da qualidade de sua superfície. Nesta fase uma amostra é colhida para testes de prensagem. Estes testes de prensagem permitem avaliar a força a ser utilizada na perfuração e o efeito de uma utilização excessiva dos filmes de óxido. Estes filmes de óxido são aplicados de forma controlada sobre o filme condutor para facilitar a sua fixação sobre o laminado base (isolante).

Fabricação da Placa

O filme condutor e o laminado base são prensados juntos para a fabricação do laminado final. Primeiro o filme condutor é depositado sobre a base inferior da prensa, em seguida um determinado número de laminados base é depositado sobre o filme condutor. O número de laminados base utilizado depende da espessura final desejada e de características inerentes ao próprio laminado. Um novo filme condutor é adicionado ao topo da pilha se um laminado de dupla-face deve ser fabricado, caso contrário a base superior da prensa será

recoberta com um demoldante. O conjunto obtido é levado para a prensa onde sob condições controladas de temperatura e pressão o material é tratado até apresentar as características desejadas (maturação da resina, espessura, resistência mecânica). A placa obtida tem suas bordas aparadas e é levada para o polimento em escovas rotativas, ficando pronta para ser utilizada na fabricação do circuito impresso. A Tabela 1.1 lista os diversos tipos de placas disponíveis e suas principais características. As placas de tipo "FR-4" e "G-10" são as mais empregadas pela indústria eletro-eletrônica [Lin 79].

1.3.2 Confecção das Máscaras

O passo inicial na fabricação de uma placa de circuito impresso é a impressão de um padrão do circuito sobre a face condutora do laminado a partir da arte-final ou de uma máscara de filme fotográfico. Os filmes fotográficos positivos ou negativos obtidos a partir da arte-final podem ser utilizados nos seguintes processos de impressão: foto-impressão, "silkscreen" e litográficos ("offset") [Coo 67]. A imagem produzida no filme fotográfico é transferida integralmente para a face condutora do laminado. Assim, os aspectos mais importantes na confecção da máscara de filme fotográfico são o tipo e qualidade do equipamento fotográfico utilizado e as condições de trabalho. Por exemplo, a câmera deve estar fixa numa base livre de vibrações e deve ter lentes com pequena distorção nas bordas. A utilização de iluminação posterior à arte-final é importante para que se consiga bom contraste na imagem.

O uso de escalas permite reduções fotográficas que elevam a definição dos traços que compõem o layout do circuito. A maioria das artes-finais para circuito impresso são traçadas com um aumento não superior a oito vezes o tamanho do circuito original.

Limpeza pré-impressão

A maior fonte de falhas nos processos de impressão, remoção do filme condutor e metalização é uma limpeza imprópria da face condutora. A baixa aderência dos protetores anti-corrosivos, a presença de furos, a remoção incompleta de filme condutor e outras falhas são geralmente relacionadas a práticas impróprias de limpeza. A limpeza pré-impressão inicia-se por um banho com jatos de solventes para remover óleos e sujeiras. A seguir a superfície metálica passa pela lixação e limpeza em escovas rotativas com pastas abrasivas. A retirada dos resíduos de pasta é feita com água. Uma limpeza química é realizada com uma seqüência de banhos ácidos e neutralizadores.

Tipo	Composição		Características
	material base	resina	
XXXPC	papel	fenólica	Resistente a umidade.
FR-2	papel	fenólica	Similar a XXXPC, menos inflamável.
XXXP	papel	fenólica	Papel empregado aumenta resistência mecânica.
FR-3	papel	epoxy	Alta resistência mecânica.
FR-4	fibra de vidro	epoxy	Pouco inflamável, resistente a produtos químicos e a umidade.
G-3	fibra de vidro	fenólica	Alta resistência a torção e estabilidade dimensional.
G-5	fibra de vidro	melamina	Alta resistência mecânica a impactos.
G-9	papel	melamina	Similar a G-5, melhor característica elétrica.
G-10	fibra de vidro	epoxy	Similar a FR-4, mais inflamável.
G-30	fibra de vidro	epoxy	Alta estabilidade dimensional sob altas temperaturas.
FR-5	fibra de vidro	epoxy	Similar a G-11, pouco inflamável.
GPO-1	fibra de vidro	poliéster	Uso geral, baixo custo.
GPO-2	fibra de vidro	poliéster	Idem GPO-1, pouco inflamável.

Tabela 1.1: Tipos de Placas de Circuito Impresso.

1.3.3 Impressão das Máscaras

A escolha do método de impressão das máscaras depende da qualidade e definição do traço de condutor requeridos, do tamanho do lote a ser produzido, da taxa de produção e dos custos envolvidos. Existem três métodos de impressão, utilizando:

- foto-protetores: são capazes da melhor definição de traço e são indicados para a produção de protótipos ou produção de pequenas quantidades (10 a 15 unidades). No processo de "impressão-remoção" a definição e resolução depende da espessura da face condutora. Tipicamente linhas de 5 mil[15] e espaços de 5 mil são possíveis em faces condutoras espessas. Uma resolução de traço variando entre 1 e 3 mil é possível se filmes metálicos finos são empregados.

- "silk screen": para altas e médias taxas de produção. Limitam as larguras de traço entre 10 e 15 mil. É considerado economicamente mais viável que os outros dois por envolver equipamentos menos dispendiosos e um processo mais simples de impressão [Coo 67].

- offset[16]: é apropriado para produção de placas de baixo custo em lotes grandes pois permite uma velocidade de impressão maior que a dos outros dois. A resolução de traço permitida é similar à obtida no "silk-screen".

Foto-protetores

Foto-protetores são revestimentos produzidos a partir de soluções orgânicas que expostas a luz de determinado comprimento de onda alteram suas propriedades químicas, tornando-se solúveis somente sob a ação de solventes químicos específicos.

Existem dois tipos de foto-protetores:

- negativos

- positivos

O foto-protetor negativo é inicialmente solúvel, se exposto ao seu solvente específico, mas polimeriza-se e torna-se insolúvel após exposição à luz; realizada através da máscara. O foto-protetor não exposto é dissolvido e lavado

[15] $1\ mil = \frac{1}{1000}\ de\ polegada.$

[16] Em Português: ofset. Preferi manter a grafia da palavra como escrita em Inglês.

da superfície da placa deixando o padrão do circuito impresso. O positivo comporta-se de modo oposto ao negativo, a exposição à luz faz com que ele se torne solúvel.

O padrão impresso é normalmente colorido para facilitar operações de inspeção e retoque. O padrão, que permanece após a dissolução e aquecimento, é insolúvel e quimicamente resistente à limpeza e aos processos de remoção do filme condutor e de metalização. Existem diversos métodos de aplicação, mas o mais comum é a aplicação através de filmes ou spray, principalmente para foto-protetores de ação negativa. Coombs [Coo 67] relaciona um número razoável de marcas de foto-protetores e seus respectivos modos de aplicação. O mais importante é que a aplicação esteja livre de furos, uniforme e muito aderente ao material base. Uma fase de pré-aquecimento é normalmente incluída quando o foto-protetor for aplicado sobre superfície de cobre. Esta fase serve para acentuar a aderência do foto-protetor sobre a superfície condutora. Após a aplicação do filme foto-protetor a placa é impressa utilizando-se a máscara do circuito. Isto é realizado expondo-se o foto-protetor à luz através da máscara do circuito. O tempo de exposição à luz depende do tipo de luz utilizada (vapor de mercúrio, incandescente ou de tungstênio), da distância da fonte de luz à superfície e do tipo e espessura de foto-protetor utilizado.

Uma falha comum nesta fase é o que se denomina "coving". Os foto-protetores podem agir de modo desigual nas fronteiras delimitadas pela máscara produzindo traços com borda arredondada e de espessura errada.

1.3.4 Perfuração

Existem dois processos de perfuração:

- por punção[17],
- por perfuração[18].

Se o método de punção é utilizado haverá uma tendência de fechamento dos furos após a operação, isto deve ser levado em conta ao se projetar o punção e sua respectiva forma para que o furo resultante tenha o diâmetro desejado. A forma e o punção devem ter um diâmetro ligeiramente maior que o desejado; o acréscimo no diâmetro é calculado em função da elasticidade do material que compõe a placa. O método de punção pode trazer problemas se o circuito impresso tem furos adjacentes muito próximos ou muito próximos da borda

[17] Em Inglês: punching.
[18] Em Inglês: drilling.

da placa. Este tipo de problema normalmente é previsto e evitado na fase de projeto do circuito impresso. As informações necessárias para a produção da forma, localização dos furos e diâmetro, são fornecidas pelo sistema de projeto auxiliado por computador.

Quando o lote produzido é pequeno torna-se inviável economicamente a utilização do sistema de punção pois o custo da confecção da forma seria pouco amortizado pelo lote. A perfuração por punção também é insatisfatória quando a precisão dos furos deve ser alta. Nesses casos, a melhor maneira de perfurar a placa é através de perfuradoras. O trabalho executado com perfuradoras é de qualidade superior. Perfuradoras automáticas de controle numérico permitem realizar o trabalho de perfuração num curto espaço de tempo sem prejuízo da precisão. Por este motivo e por sua grande flexibilidade, são programáveis em função da localização e diâmetro dos furos, as perfuradoras automáticas têm sido muito utilizadas na fabricação de circuitos impressos.

1.3.5 Metalização dos Circuitos

O objetivo fundamental da metalização da placa de circuito impresso é a produção dos furos metalizados. Para que cumpram suas funções adequadamente os furos metalizados devem atender às seguintes exigências: o revestimento da parede interna do furo deve ser completo e ter espessura uniforme, livre de grânulos ou fraturas. A espessura do metal depositado varia entre 1.5 e 3.0 mil. O processo de metalização ocorre quase sempre em seguida à perfuração.

Na realidade o processo de produção dos circuitos impressos depende do número de faces do circuito impresso. No caso de um circuito impresso de face simples a fabricação tem menos fases. Recobre-se a face condutora da placa com o foto-protetor e sobrepõe-se a máscara negativa previamente obtida. Em seguida, o conjunto é exposto, por um tempo determinado, à luz. Dessa forma, expõe-se à luz somente as regiões do foto-protetor que estão sob as partes transparentes da máscara, correspondentes às trilhas do circuito impresso.

Depois, inicia-se um processo de revelação. Nele é eliminada a parte do foto-protetor não exposta à luz, deixando cobertas as regiões que formam as trilhas condutoras.

Terminado o processo fotográfico, expõe-se a placa à corrosão do filme condutor, isto é, à ação de uma solução química de percloreto ferroso e água, por exemplo. Essa operação remove completamente o filme condutor não protegido pelo foto-protetor.

Uma vez impresso o layout do circuito sobre a placa, graças à retirada do filme condutor exposto, remove-se com outro solvente o foto-protetor que recobre as trilhas, lava-se e enxuga-se o circuito. O próximo passo é a perfuração, já descrita anteriormente. A operação pode ser concluída neste ponto. Porém, no caso de circuitos de alta qualidade, realiza-se um processo químico de metalização no qual é depositada uma camada de uma liga de estanho e chumbo sobre a superfície das trilhas. Essa camada serve para evitar a oxidação e proteger o filme condutor, facilitando com isso a operação de soldagem dos componentes. No fim do processo a liga de estanho-chumbo é refundida para tornar-se uniforme. Para isso o circuito é mergulhado num banho de óleo fervente por alguns minutos. Depois de resfriado e limpo o circuito apresenta as trilhas brilhantes e em condições favoráveis para a correta execução de cada tipo de soldagem e para as sucessivas operações de montagem.

Nos circuitos impressos de dupla-face, a primeira etapa de fabricação consiste em furar todos os pontos e, em seguida, metalizar esses furos. Para esse fim, expõe-se a placa a um procedimento químico, durante o qual uma película de metal é depositada sobre as paredes do furo e faces condutoras. Depois vem o processo fotográfico, idêntico ao já descrito, mas sobre ambas as faces da placa. É preciso, nesse momento, muito cuidado para fazer coincidir perfeitamente as máscaras de cada face com os furos já existentes.

O processo químico de depósito da liga de estanho e chumbo é idêntico ao seguido para circuitos de uma face, com uma única diferença: a liga recobre também as paredes dos furos, que, assim, oferecem melhores condições para a soldagem dos terminais de componentes.

Os circuitos multi-face são construídos pela superposição de circuitos de duas faces separados por faces isolantes. As ligações entre as diversas camadas são realizadas por furos metalizados. O conjunto de circuitos simples é empilhado na ordem correta e prensado, tendo ao final uma aparência semelhante à do circuito de dupla-face, entretanto, consegue-se divisar as diversas camadas existentes olhando-se para as bordas do circuito que apresentam uma estrutura laminar característica.

No caso de circuitos dupla-face nota-se a existência de duas fases de metalização. A primeira metalização é responsável pelo depósito de metal que garante a boa interligação elétrica entre as trilhas das duas faces do circuito, facilitando a fixação da liga de estanho-chumbo que ocorre na segunda metalização, a metalização final.

Existem diversos métodos de metalização para se conseguir o depósito metálico nos furos, mas o melhor deles é o de deposição química (oxidação) da face condutora [Coo 67]. Este método consiste na sensibilização e deposição

de metal sobre o furo e face condutora. A metalização pode ser realizada com vários metais ou ligas e cada um deles exige técnicas diferentes. Ligas de cobre, ouro, soldas especiais e níquel são as mais utilizadas [Coo 67]. Para se fazer a metalização final, com a liga de estanho-chumbo existem dois métodos, a metalização negativa[19] e a metalização positiva[20].

A metalização negativa é o processo onde a superfície toda da placa e os furos são metalizados. As áreas onde não deve ocorrer a metalização são protegidas por duas camadas protetoras. Um filme protetor é aplicado primeiro, e em seguida, sobre este é depositada uma camada metálica para evitar a remoção do material condutor original. A metalização positiva difere da metalização negativa somente na área sobre a qual o processo acima descrito se aplica. Aqui, somente o layout do circuito e os furos recebem as camadas protetoras.

A metalização consiste na produção de depósitos de metal sobre as superfícies condutoras realizada pela passagem de corrente elétrica através da solução de metalização. A taxa de metalização é determinada pela intensidade de corrente aplicada à superfície que está sendo metalizada.

1.3.6 Montagem Automática

As vantagens técnicas mais significativas da montagem semi-automática ou automática de componentes estão relacionadas à confiabilidade exigida dos circuitos impressos atualmente fabricados. Não é desejável que um circuito deixe de operar numa situação crítica devido a uma falha de inserção ou solda. Com o processo automático de montagem é possível conseguir-se um alto grau de uniformidade e maior densidade.

As variáveis que afetam a factibilidade da montagem automática de componentes são: a uniformidade do projeto do circuito, variabilidade do espaçamento deixado para a inserção, a localização dos furos e seu diâmetro, a variedade dos componentes a serem montados. Componentes axiais (resistores, capacitores, diodos, etc.) são montados em fitas formando longas tiras onde os componentes são dispostos segundo o espaçamento requerido pela máquina de inserção. Onde a produção de uma mesma placa se dá em grandes volumes é viável a implementação de uma linha de montagem com insersores controlados numericamente.

Como normalmente há componentes com variação de valor, polaridade e características físicas em cada placa, muitas tentativas têm sido realizadas

[19] Em Inglês, panel plating.
[20] Em Inglês, pattern plating.

para seqüenciar a inserção destes componentes utilizando um único mecanismo insersor. Existem basicamente três tipos de sequências possíveis. Para componentes que têm a mesma dimensão física mas características elétricas diferentes, por exemplo, resistores de 20Ω e de 40Ω. Nesses casos a inserção e dobradura é simples, sendo necessário somente determinar a sua sequência na fita. Um problema um pouco mais complicado existe quando os componentes têm as mesmas características físicas mas são diferentes, por exemplo, resistores e diodos. Nesses casos a orientação dos componentes deve ser levada em conta. O problema mais complexo existe quando os componentes têm características físicas diferentes. Os mecanismos de inserção e dobradura têm de ser flexíveis para permitir a inserção automática [Coo 67].

1.3.7 Solda

Entre a fase de inserção de componentes e a de soldagem a placa pode sofrer novos tratamentos. Ela pode, por exemplo, ser recoberta com elementos protetores que garantam resistência contra a fadiga imposta pelas condições de alta temperatura existentes na estação de solda. As regiões a serem soldadas recebem um tratamento químico que favorece o processo de solda, melhorando sua qualidade. Em geral, é usado um tratamento a base de ácidos para retirar resíduos óxidos.

A primeira etapa do processo de solda envolve uma fase de limpeza para retirar qualquer partícula ou resíduo ainda existente na placa. A seguir esta passa pela aplicação de um preparado químico para auxiliar no processo de solda. Este preparado é muito fusível, tendo papel determinante na fusão da solda. Por este motivo, após a aplicação, ele deve ser mantido numa temperatura elevada e constante para que permaneça ativo, isto é, em condições de prontamente reagir com a solda. Esta fase que precede a solda é denominada pré-aquecimento.

Após a solda a placa é novamente limpa para remover restos de fluxo de solda e preparada para a fase de inspeção e teste do circuito.

1.3.8 Testes

Existem sistemas de teste capazes de fornecer para o circuito a correta alimentação, sinais de entrada, circuitos de teste e testes funcionais para avaliar seu correto funcionamento. A grande maioria das estações de teste utilizadas atualmente estão interligadas ao sistema de projeto auxiliado por computador onde a placa foi projetada. Isto facilita a geração de dados de teste para o circuito. Se necessário, as correções podem ser facilmente incorporadas às

Largura do Traço (mil)	Máxima Corrente (Ampéres)
20	1.5
25	2.3
31	2.8
62	5.0
125	10.0
250	15.0

Espessura = 1.35 mil.

Tabela 1.2: Largura de Traços.

bases de dados do sistema diminuindo significativamente o tempo de projeto e fabricação do circuito impresso.

1.4 Tecnologia e Restrições de Projeto

Há um vínculo direto entre as tecnologias de fabricação de circuitos impressos e as restrições de projeto a serem seguidas pelos traçadores de rotas e outros subsistemas existentes nos sistemas de projeto auxiliado por computador.

Um exemplo disso são os padrões que ditam as diferentes seções retas (largura x espessura) dos condutores em função da corrente que os atravessa durante sua operação no circuito [Lin 79], [Bre 72] (Tabela 1.2). Estes padrões ditam também as regras a serem seguidas para o posicionamento entre condutores paralelos adjacentes e, entre coroas metalizadas e traços.

Em geral, recomenda-se que a distância entre traços paralelos adjacentes seja no mínimo igual a maior das larguras dos traços a serem posicionados. Larguras diferentes de traço são utilizadas para sinais como *Vcc* e *Terra*, porque por eles passa uma corrente elétrica maior. Outra recomendação diz respeito a distância entre os traços e a borda da placa de circuito impresso que não deve ser inferior a 50 mil; o ideal é manter uma distância de 100 mil entre os traços e a borda.

Outras restrições podem existir, e.g., alguns processos de fabricação podem formar pontes de solda entre traços vizinhos se a distância entre os traços for muito pequena e isto pode obrigar o uso de parâmetros diferentes durante o projeto. Os sistemas de projeto de circuitos impressos auxiliados por computador devem ser flexíveis o bastante para permitir o projeto dentro das restrições impostas pela maioria das tecnologias existentes.

Capítulo 2

Aspectos Teóricos

O problema do projeto de circuitos impressos pode ser decomposto em subproblemas que executam funções relacionadas mas suficientemente auto-contidas para serem tratadas de modo independente. Os subproblemas normalmente encontrados no desenvolvimento de sistemas automatizados de projeto de circuitos [Sah 80] são:

- Síntese: trata da conversão de uma representação de um sistema digital para uma outra, restrito a que as duas representações sejam equivalentes funcionalmente. Em sistemas para projeto de circuitos impressos este problema aparece de forma atenuada, em geral relacionada com a fase de aquisição de dados onde o esquema do circuito é convertido para uma representação interna utilizando listas de sinais e de componentes.

- Construção de uma Biblioteca de Módulos Lógicos[1] Intercambiáveis: novamente temos um problema que ocorre com maior frequência no projeto de circuitos integrados. Como projetar-se os módulos lógicos de modo a manter seu número na biblioteca dentro de limites razoáveis. No caso de sistemas de projeto de circuitos impressos este problema fica reduzido à manutenção de uma base de dados de componentes.

- Particionamento: obter um particionamento de um circuito em módulos de modo a otimizar:

  - o espaço ocupado pelos circuitos;
  - o número de conexões externas necessárias entre módulos;
  - o tempo de propagação dos sinais;
  - a facilidade de teste e manutenção do circuito final.

- Posicionamento: como posicionar objetos inter-relacionados em posições fixas dentro de uma área especificada. A definição precisa do que são os objetos e de quais são as relações entre eles depende da tecnologia e do nível hierárquico utilizado para representá-los. Pode-se falar do posicionamento de circuitos lógicos num componente, de componentes numa placa de circuito impresso, etc. Alguns critérios de otimização associados ao problema de posicionamento são:

  - minimização de interferências elétricas entre sinais;

[1] Módulo Lógico: conjunto de funções lógicas integradas e que desempenham alguma função básica necessária à construção de circuitos lógicos mais complexos.

  - controle de dissipação de calor;
  - minimização do número de cruzamentos entre sinais, etc.

- Traçado: interconexão e traçado de sinais. As restrições normalmente impostas são [Fen 73]:
  - número de faces disponíveis para traçado;
  - existência de furos metalizados entre faces condutoras;
  - comprimento das rotas;
  - largura das rotas;
  - densidade das rotas;
  - atribuição de sinais a faces, etc.

- Deteção de Falhas: geração de dados para teste e deteção de falhas nos circuitos.

  Um circuito pode ser definido como uma função de n argumentos que deve ser satisfeita. Verificar se uma dada função (circuito) produz o resultado esperado para todos os valores de cada um dos n argumentos de entrada é o problema que se pretende resolver.

Os textos de Breuer [Bre 72] e Hightower [Hig 80] são boas compilações a respeito dos problemas e técnicas de solução empregadas no desenvolvimento de sistemas auxiliares de projeto de circuitos. Todos os problemas acima têm em comum o fato de não se saber se é possível ou não encontrar um algoritmo eficiente que os resolva [Sah 80], [Don 80]. Faremos agora uma análise um pouco mais detalhada do problema de traçado de rotas. Podemos reformulá-lo como segue: "O objetivo principal de um traçador de rotas é interligar todos os sinais do circuito por rotas de comprimento mínimo".

Nesta reformulação tornamos clara a restrição quanto ao comprimento da rota. Rotas de comprimento mínimo garantem um pequeno tempo de propagação de sinal aumentando a eficiência do circuito.

A próxima Seção introduz conceitos teóricos importantes para a compreensão das estruturas e algoritmos relacionados ao problema formulado acima.

2.1 Grafos

Um grafo G consiste de um conjunto VG de vértices, um conjunto aG de arestas e de uma função ψ_G que associa a cada aresta α de G um par não ordenado de vértices de G, denominados extremos de α.

Grafos podem ser representados por diagramas, onde cada vértice é representado por um ponto e cada aresta por uma linha ligando os pontos que são seus extremos.

Os extremos de uma aresta são *adjacentes*; são *adjacentes* também arestas com pelo menos um extremo em comum. Uma aresta é um *laço* se seus extremos coincidem, uma *ligação* caso contrário.

Um grafo G é *finito* se VG e aG forem ambos finitos. A partir de agora o termo grafo designa sempre um grafo finito. O *tamanho* de um grafo G é o inteiro $|VG| + |aG|$. O grafo vazio é o grafo de tamanho zero, sem arestas nem vértices.

Um grafo G é *simples* se não tem laços nem duas ligações distintas entre o mesmo par de extremos. Um grafo *completo* é um grafo simples cujos vértices são dois a dois adjacentes.

O grau $g(v)$ de um vértice v em G é o número de arestas incidentes a v, cada laço sendo contado duas vezes. Um grafo é *regular* se todos os seus vértices têm o mesmo grau.

Um grafo H é um *subgrafo de* G (denotado por $H \subseteq G$) se $VH \subseteq VG$, $aH \subseteq aG$ e ψ_H é a restrição de ψ_G a aH. Quando $H \subseteq G$ mas $H \neq G$ escrevemos $H \subset G$ e chamamos H um *subgrafo próprio* de G. Um *subgrafo gerador* de G é um subgrafo H com $VH = VG$.

Removendo-se de G todos os laços e, para todo par de vértices adjacentes, todas menos uma ligação obtem-se um *subgrafo gerador simples* de G.

Suponha que V' é um subconjunto não vazio de V. O subgrafo de G cujo conjunto de vértices é V' e cujo conjunto de arestas é o conjunto daquelas arestas que têm ambos os extremos em V' é denominado subgrafo de G gerado por V' e é denotado por $G[V']$; dizemos que $G[V']$ é um *subgrafo gerado* de G.

Suponha que a' é um subconjunto não vazio de a. O subgrafo de G cujo conjunto de vértices é o conjunto dos extremos das arestas em a' e cujo conjunto de arestas é a' é denominado subgrafo de G gerado por a', denotado por $G[a']$.

2.1.1 Passeios

Um passeio P em G é uma seqüência finita e não vazia $(v_0, \alpha_1, v_1, \ldots, \alpha_n, v_n)$, cujos termos são alternadamente vértices v_i e arestas α_j, e tal que, para todo $i, 1 \leq i \leq n$, v_{i-1} e v_i são extremos de α_i.

Dizemos que P é um passeio de v_0 a v_n; os vértices v_0 e v_n são a *origem* e o *término* de P, respectivamente; os vértices $v_1, \ldots, v_{n-1}$ são vértices internos de P; VP e aP denotam os conjuntos $\{v_0, \ldots, v_n\}$ e $\{\alpha_1, \ldots, \alpha_n\}$ respectiva-

mente; o passeio P *passa* por α_i e por v_j ($\alpha_i \in aP$ e $v_j \in VP$). O comprimento de P é o inteiro n.

Se as arestas $\alpha_1, \ldots, \alpha_n$ de P forem duas a duas distintas entre si então P é uma *trilha*. Se os vértices $v_0, \ldots, v_n$ forem dois a dois distintos entre si então P é um *caminho*. Se a origem e o término de P coincidirem e $n > 0$ então P é *fechado*. Se P for uma trilha fechada e se $v_1, \ldots, v_n$ forem dois a dois distintos então P é um *circuito*.

A *distância* de um vértice u a outro v em G é o mínimo dos comprimentos dos passeios de u a v em G. (Se não existir nenhum passeio de u a v então dizemos que a distância é infinita).

Um vértice u de G é dito *ligado* ao vértice v de G se existe um passeio de u a v em G sendo a relação de ligação uma relação de equivalência. Portanto, exite uma partição de V em subconjuntos não vazios $V_1, V_2, \ldots, V_n$, tal que dois vértices u e v pertencem ao mesmo conjunto V_i se, e somente se, ambos são ligados. Os subgrafos $G[V_1], G[V_2], \ldots, G[V_n]$ são denominados componentes de G. Se G tem exatamente uma componente, G é *conexo*.

Se, associado a cada aresta α de um grafo G, temos um número real p(α), chamado *peso* de α, então temos um *grafo com peso nas arestas*.

2.1.2 Árvores

Um grafo acíclico é um grafo que não contém circuitos. Uma *árvore* é um grafo conexo acíclico.

Uma *árvore de custo mínimo* de um grafo com pesos nas arestas é a árvore gerada por VG com $a'G \subseteq aG$ tal que a soma dos pesos das arestas em $a'G$ é no mínimo tão pequena quanto a soma dos pesos das arestas de qualquer outro conjunto de arestas que gera árvores em G. Uma árvore de custo mínimo de um grafo com pesos nas arestas é uma *árvore ótima*.

2.1.3 Grafos e Circuitos

Pretende-se gradativamente relacionar grafos e circuitos elétricos de modo a caracterizar o problema de traçado como tendo complexidade exponencial.

Grafos com pesos nas arestas podem ser utilizados para representar circuitos elétricos: os vértices representam os pontos eletricamente equivalentes a serem interligados, as arestas representam as ligações entre esses pontos e seus pesos a distância geométrica entre cada par de pontos (vértices do grafo).

A definição precisa do termo distância geométrica depende da particular métrica utilizada para descrever o circuito, em nosso caso utilizaremos a seguinte definição: *Distância geométrica retilínea ou de Manhatan* entre dois

pontos p_1 e p_2 de coordenadas (x_1, y_1) e (x_2, y_2) respectivamente é designada por $d(p_1, p_2)$ e definida como:

$$d(p_1, p_2) = |x_1 - x_2| + |y_1 - y_2|$$

Atribuindo significado geométrico aos vértices de G obteremos a definição de *Árvore de Steiner Ótima.*

Uma *Árvore de Steiner Ótima* para os pontos $p_1, p_2, \ldots, p_n$ (vértices) no plano é uma árvore que interconecta estes pontos usando retas tal que o comprimento total seja o mínimo possível. A fim de conseguir o comprimento mínimo, a árvore de Steiner ótima poderá conter outros vértices (pontos de Steiner) além de $p_1, p_2, \ldots, p_n$. Supõe-se que os pontos de Steiner não introduzem custo adicional.

Se, finalmente, restringimos as retas a serem somente retas verticais e horizontais obtemos uma *Árvore Retilínea de Steiner*: Seja A um conjunto finito de pontos no plano. Uma *árvore retilínea de Steiner* para A é uma árvore, composta somente por segmentos de reta verticais e horizontais, que interconecta todos os pontos em A. Uma *árvore retilínea de Steiner ótima* é aquela que satisfaz a condição acima e na qual os segmentos de reta utilizados para interligar os pontos têm o menor comprimento possível. Este problema é NP-completo [Gar 77] e isso evidencia que soluções heurísticas deverão ser encontradas para resolver qualquer problema de interesse prático que possa ser reduzido ao problema acima colocado.

2.1.4 Algoritmos Relacionados ao Problema

Um outro modo de caracterizar o problema é especificar que os pontos deverão ser ligados em seqüência, de modo a obter-se o caminho (circuito) de menor comprimento total.

Esta última formulação relaciona o problema do traçado de rotas ao problema do caixeiro viajante[2] [Bon 76]. Em 1956, Kruskal [Kru 56] relaciona o problema de se encontrar uma árvore ótima ao problema do caixeiro viajante. Neste texto, Kruskal apresenta um primeiro algoritmo para resolver estes problemas. Logo a seguir, Loberman e Weinberger [Lob 57] descrevem um outro conjunto de procedimentos para resolver os mesmos problemas, já pensando em aplicações na área de circuitos. Entre 1957 e 1959, Prim [Pri 57] e Dijkstra [Dij 59] descrevem, de modo independente, novos algoritmos mais eficientes para resolvê-los.

[2]Em Inglês: The traveling salesman problem.

Os trabalhos citados acima, juntamente com outros trabalhos devidos a Hanan [Han 66], Gilbert & Pollak [Gil 68], Hwang [Hwa 76], Held & Karp [Hel 70],[Hel 71] e Bellmore [Bel 68] trazem a quase totalidade da teoria sobre árvores de Steiner, árvores ótimas e o problema do caixeiro viajante necessária como fundamento teórico para o problema de traçado de rotas em circuitos impressos.

Passamos agora a cuidar da descrição um pouco mais detalhada de dois desses algoritmos para determinar árvores ótimas.

2.2 Estruturas de Dados

As principais estruturas de dados utilizadas pelos algoritmos descritos nas próximas Seções estão reunidas aqui.

2.2.1 Conjuntos e Seqüências

Seja S um conjunto, representado por uma estrutura de dados adequada, e seja u um elemento qualquer de um conjunto universo U tal que $S \subseteq U$. As operações fundamentais sobre o conjunto S são:

- membro(u,S) : $u \in S$?;
- inserção(u,S) : inclui u em S;
- remoção(u,S) : remove u de S.

Suponha agora que $\{S_1, S_2, \ldots, S_k\}$ seja uma coleção de conjuntos tal que $S_i \cap S_j = \emptyset$ para qualquer $1 \leq i, j \leq k$. Algumas operações úteis sobre esta coleção são:

- busca(u) : retorna j, se $u \in S_j$;
- união(S_i, S_j, S_k) : retorna $S_k = S_i \cup S_j$.

Quando o conjunto universo U admite uma relação de ordem entre seus elementos as seguintes operações são importantes:

- min(S) : retorna o elemento mínimo de S;
- partição(u,S) : particiona S em $\{S_1, S_2\}$, tal que $S_1 = \{v | v \in S \wedge v \leq u\}$ e $S_2 = S - S_1$;

- concatenação(S_1, S_2) : supondo que para algum $u' \in S_1$, $u'' \in S_2$ vale $u' \leq u''$, retorna o conjunto ordenado $S = S_1 \cup S_2$.

Definidas as operações básicas podemos especificar as estruturas de dados que utilizam estas operações.

Para conjuntos ordenados temos (não nos atendo a detalhes de implementação):

Estruturas de Dados	Operações
Dicionário	membro, inserção, remoção
Fila de Prioridade	min, inserção, remoção
Fila Concatenável	inserção, remoção, partição, concatenação

Normalmente, cada uma destas estruturas é implementada como uma árvore balanceada. Com este tipo de implementação cada uma das operações acima é executada em tempo logarítmico e a memória usada é proporcional ao tamanho do conjunto.

Conjuntos não ordenados podem ser manipulados como conjuntos ordenados se impusermos, artificialmente, uma ordem aos elementos. Uma estrutura de dados típica para manipular conjuntos assim formados é a seguinte:

Estruturas de Dados	Operações
Heap	inserção, remoção, busca, união, (min)

2.2.2 Árvore de Intervalos

A *árvore de intervalos* [Pre 85], é uma estrutura de dados utilizada para manipular intervalos sobre a reta real, cujos extremos pertencem a um conjunto fixo de N coordenadas. Como o conjunto de coordenadas é fixo, a árvore de intervalos, como originalmente proposta, é uma estrutura estática em relação às coordenadas, ou seja, ela não suporta operações de inserção e remoção de coordenadas. As coordenadas dos intervalos podem ser normalizadas. Basta trocar cada uma delas pelo inteiro que representa a posição que ela ocupa no conjunto, após a sua ordenação em ordem crescente. Assim, podemos considerar o conjunto de N coordenadas como sendo os inteiros no intervalo $[1, N]$.

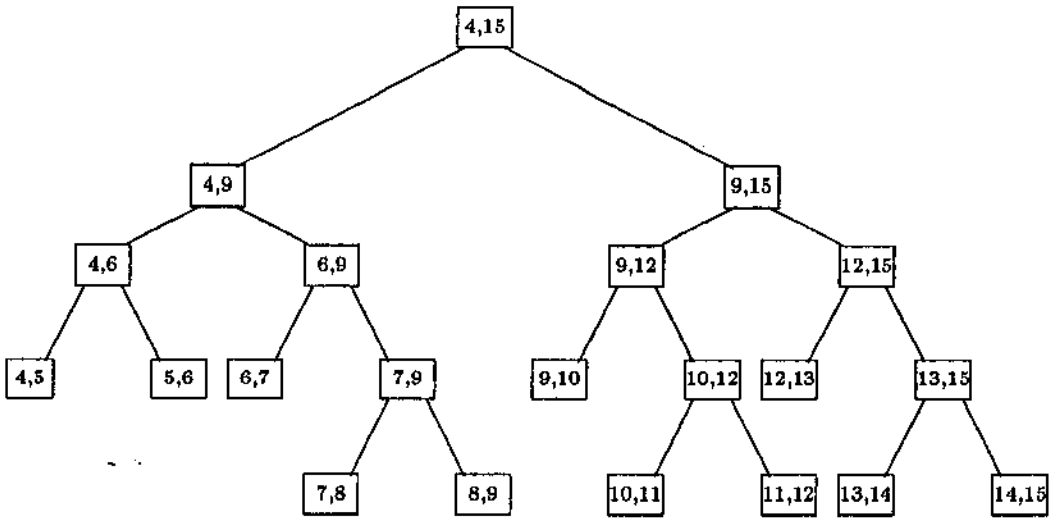


Figura 2.1: Árvore de Intervalos $\mathcal{T}(4,15)$.

A árvore de intervalos pode ser representada por uma árvore binária. Dados os inteiros e e d, com $e < d$, a árvore de intervalos $\mathcal{T}(e,d)$ é definida recursivamente como segue:

1. Ela tem uma raiz r, com parâmetros $I[r] = e$ e $F[r] = d$[3],
2. e, se $e-d > 1$, uma subárvore esquerda $\mathcal{T}(e, \left\lfloor \frac{I[r]+F[r]}{2} \right\rfloor)$, e uma subárvore direita $\mathcal{T}(\left\lfloor \frac{I[r]+F[r]}{2} \right\rfloor, d)$.

Os parâmetros $I[r]$ e $F[r]$ definem o intervalo $[I[r], F[r]] \subseteq [e,d]$ associado ao nó r. A árvore de intervalos $\mathcal{T}(4,15)$ está representada na Figura 2.1. O conjunto de intervalos $\{[I[r], F[r]] | r$ é raiz de $\mathcal{T}(e,r)\}$ são os *intervalos padrão* de $\mathcal{T}(a,b)$. Os intervalos padrão localizados nas folhas de $\mathcal{T}(e,r)$ são denominados *intervalos elementares*[4].

A árvore de intervalos $\mathcal{T}(e,d)$, quando representada como uma árvore binária, é adequada para armazenar intervalos cujos extremos pertencem ao conjunto $e, e+1, \ldots, d$ de modo dinâmico, isto é, incluindo operações de inserção e remoção. Especificamente, para $e - d > 3$, um intervalo qualquer

[3] I e F são mnemônicos para, respectivamente, "*I*nício" e "*F*im" de intervalo.

[4] Estritamente falando, o intervalo associado a r é o intervalo semi-aberto à direita $[I[r], F[r])$, exceto para os nós do caminho mais à direita de $\mathcal{T}(a,b)$, cujos intervalos são fechados.

$[m,n]$, com inteiros $m < n$, será dividido em uma coleção de, no máximo, $\lceil \log_2(d-e) \rceil + \lfloor \log_2(d-e) \rfloor - 2$ intervalos padrão de $\mathcal{T}(e,d)$. A segmentação do intervalo $[m,n]$ é completamente especificada pela operação que insere $[m,n]$ na árvore de intervalos $\mathcal{T}$, esboçada como:

```
{ Procedimento de Inserção }
procedimento insere(m,n,r);
{ [m,n] : intervalo, r : raiz da árvore de intervalos. }
início
   se (m ≤ I[r]) ∧ (F[r] ≤ n)
      então associe [m,n] a r
      senão
         início
            se (m < ⌊(I[r] + F[r])/2⌋) então
               insere(m,n,r ↑ .esq);
            se (⌊(I[r] + F[r])/2⌋ < n) então
               insere(m,n,r ↑ .dir)
         fim;
fim.
```

A ação da chamada *insere*$(m,n,raiz(\mathcal{T}))$ corresponde a um percurso em $\mathcal{T}$. Quando o nó que mantém o intervalo dentro do qual $[m,n]$ se encontra é encontrado então ele é atualizado para incluir o intervalo $[m,n]$.

A inclusão de um novo intervalo em um nó r qualquer de $\mathcal{T}$ pode ser implementada de diversas formas. Frequentemente, tudo o que se necessita é saber a cardinalidade do conjunto de intervalos existente em um dado nó r; isto pode ser realizado pela simples manutenção de um contador inteiro, digamos $C[r]$, em cada nó. Neste caso a operação *associe* $[m,n]$ *a* r realizada no procedimento acima torna-se:

$$C[r] \leftarrow C[r] + 1$$

Em outras aplicações, pode ser necessário preservar a identidade dos intervalos associados a um dado nó. Neste caso, basta associarmos a cada nó r de $\mathcal{T}$ uma lista ligada onde os intervalos serão mantidos.

De modo simétrico, pode-se definir o procedimento de remoção de um intervalo da árvore de intervalos (supõe-se que estamos interessados somente na cardinalidade do conjunto de intervalos associado a cada nó de $\mathcal{T}$):

```
{ Procedimento de Remoção }
procedimento remove(m,n,r);
{ [m,n] : intervalo, r : raiz da árvore de intervalos. }
início
   se (m ≤ I[r]) ∧ (F[r] ≤ n)
      então C[r] ← C[r] − 1
      senão
         início
            se (m < ⌊(I[r] + F[r])/2⌋) então
               remove(m,n,r ↑ .esq);
            se (⌊(I[r] + F[r])/2⌋ < n) então
               remove(m,n,r ↑ .dir)
         fim;
fim.
```

A árvore de intervalos aqui definida é útil ferramenta na solução de problemas de interseção geométrica e será usada pelo algoritmo de traçado de rotas proposto adiante. Convém notar ainda que, dada uma árvore de intervalos T, uma busca binária em T é suficiente para determinar o número de intervalos que contém um dado ponto x.

2.3 Algoritmos para Determinar Árvores Ótimas

2.3.1 Algoritmo de Dijkstra

Seja G um grafo conexo com pesos nas arestas que representam o custo ou a distância de se passar de um de seus extremos a outro através desta aresta. O custo é sempre um real não negativo. O algoritmo de Dijkstra encontra o caminho de custo mínimo entre um vértice origem, digamos v_i, e todos os outros vértices de G. O algoritmo trabalha mantendo um conjunto S de vértices cujo menor custo até v é conhecido. Inicialmente, S contém somente o vértice origem. A cada passo, adiciona-se a S um outro vértice v_j de modo a manter-se o comprimento entre v_i e v_j o menor possível. Como todas as arestas têm custos não negativos, pode-se sempre encontrar um caminho de comprimento mínimo, digamos P, entre o vértice origem v_i e o vértice v_j em questão. A cada passo do algoritmo usamos um vetor D para registrar o comprimento do menor caminho P entre o vértice origem e o vértice v_j corrente (último vértice incorporado a S). Como S inclui todos os vértices de G obteremos todos os caminhos, acima denominados P, que representam o caminho de menor comprimento entre v_i e cada outro vértice de G.

Abaixo temos uma possível codificação para o algoritmo de Dijkstra. Ela supõe que são fornecidos: o grafo G onde $V = \{1, 2, \ldots, n\}$ e que $v_i = 1$. C é uma matriz de pesos, onde $C[i, j]$ é o peso para se ir de v_i a v_j através da aresta (i, j). Se não existe ligação entre os vértices i e j então $C[i, j] = \infty$. A cada passo $D[i]$ contém o menor comprimento computado até aquele instante do caminho P entre o vértice origem e o vértice corrente.

Para calcular todos os caminhos de comprimento mínimo entre todos os vértices de G teríamos de executar o algoritmo de Dijkstra n vezes. O algoritmo de Floyd, uma variação do algoritmo de Dijkstra, resolve este problema de modo mais eficiente e torna-se assim uma das possíveis opções para a solução do problema da determinação de árvores ótimas.

Algoritmo de Dijkstra:

```
procedimento Dijkstra;
   início
(1)    S := {1};
(2)    para i := 2 até n faça
(3)    D[i] = C[1,i];
(4)    para i := 1 até n − 1 faça
          início
(5)          escolha um vértice w ∈ V − S tal que D[w] seja mínimo;
(6)          inserção(w, S);
(7)          para todo v ∈ V − S faça
(8)             D[v] := min({D[v], D[w] + C[w,v]})
          fim
   fim;
```

Complexidade:

Suponha que o algoritmo de Dijkstra é executado sobre um grafo com n vértices e e arestas. Se usarmos uma matriz de adjacência para representar o grafo, então os comandos iterativos das linhas (7) e (8) implicam um tempo $O(n)$, e são executados $n-1$ vezes, o que nos fornece um tempo total de $O(n^2)$. É fácil verificar que o restante do algoritmo não consome um tempo maior que este.

Se e é muito menor que n^2, é melhor utilizar-se uma lista de adjacência para representar o grafo. Neste caso, os comandos iterativos das linhas (7) e (8) podem ser implementados percorrendo-se as listas de adjacência usando o vértice corrente w como chave, esta operação é realizada em tempo proporcional ao número de arestas incidentes em w. Quando somamos esta quantidade

para todo w em G temos o número de arestas e, assim o tempo total gasto pelas linhas (7) e (8) é $O(e)$ ao invés de $O(n^2)$.

As linhas (1)-(3) consomem tempo $O(n)$, e também as linhas (4) e (6). Podemos limitar o tempo gasto na linha (5) a $O(\log n)$ se organizarmos os vértices de $V - S$ em uma fila de prioridade, operação $\min(\{V - S\})$. Neste caso cada uma das $n - 1$ buscas requer $O(\log n)$. Como resultado, o tempo total gasto pela versão com filas de prioridade do algoritmo de Dijkstra é da ordem de $O(e + n \log n)$. Este tempo é consideravelmente melhor que $O(n^2)$, se e for muito menor que n^2.

2.3.2 Algoritmo de Kruskal

Suponha, novamente, que são dados um grafo conexo com peso nas arestas G, com $V = \{1, 2, \ldots, n\}$. Uma outra maneira de se obter uma árvore ótima é iniciar com um grafo T, onde $VT = VG$ e $aT = \emptyset$. Cada vértice é por si só uma componente conexa. Na medida em que o algoritmo progride, teremos sempre uma coleção de componentes conexas, e para cada componente, teremos selecionado conjuntos de arestas que formam árvores ótimas.

Para construir componentes conexas maiores, examina-se as arestas em aG, em ordem crescente de peso. Se a aresta liga vértices em componentes conexas diferentes, é adicionada a T. Se a aresta liga vértices na mesma componente ela é descartada, pois formaria um circuito. Quando todos os vértices de G compuserem uma única componente, T será uma árvore ótima gerada por VG.

Algoritmo de Kruskal [1]:

procedimento Kruskal;
início
a) Escolha uma aresta e_1 tal que $p(e_1)$ seja tão pequeno quanto possível.
b) Se arestas $e_1, e_2, \ldots, e_i$ foram escolhidas então escolha uma aresta $e_{i+1} \in E - \{e_1, e_2, \ldots, e_i\}$ de modo que:
1. $G[\{e_1, e_2, \ldots, e_{i+1}\}]$ seja acíclico;
2. $p(e_{i+1})$ seja tão pequeno quanto possível.

c) Pare quando o passo **b** não puder mais ser executado.
fim;

Codificando o algoritmo de Kruskal utilizando as operações definidas na Seção anterior temos:

Algoritmo de Kruskal [2]:

```
procedimento Kruskal;
   início
      { G = (VG,aG) }
      { P = fila de prioridade (custos das arestas). }
      { C1,C2,...,C|VG| = componentes de G. }
      { T = conjunto de arestas. }
      T = ∅; c = 0; n = |VG|;
(1)   para v ∈ VG faça
         início
(2)         c = c + 1;
(3)         inserção(v,Cc);
         fim;
(4)   para e ∈ aG faça inserção(e,P);
(5)   enquanto n > 1 faça
         início
(6)         remoção(e,P);
            { seja e = (u,v) }
(7)         se busca(u) ≠ busca(v) então
               início
            { e liga componentes distintas. }
(8)               união(Cu, Cv, Cv);
(9)               n = n − 1;
(10)              inserção(e,T);
               fim
   fim;
```

Complexidade:

Utilizando árvores binárias para implementar as operações empregadas obtem-se tempos de processamento da ordem de $\log N$, onde N é o número de elementos na árvore.

Se há e arestas em G, gastamos um tempo de $O(e \log e)$ inserindo as arestas na fila de prioridade, linha (4). Em cada iteração do *enquanto*, linhas (5)-(10), gasta-se um tempo proporcional a $O(\log e)$ para encontrar a aresta de custo mínimo, pois, as operações sobre a fila de prioridade tomam $O(e \log e)$ no pior caso. O tempo total necessário para executar as operações de *união* (8) e

busca (7) é dependente de implementação, entretanto em [Sed 84] temos uma implementação que consome $O(e \log e)$, utilizando um heap. Assim, temos que o algoritmo de Kruskal pode ser implementado de modo a executar em tempo proporcional a $O(e \log e)$.

2.4 Algoritmos Geométricos

Um problema que ocorre frequentemente em aplicações envolvendo dados geométricos é: "Dado um conjunto de N objetos, existe interseção entre pelo menos dois deles?". Os "objetos" envolvidos podem ser segmentos de reta, retângulos, circulos, etc.

A solução óbvia para o problema da interseção é verificar cada par de objetos para saber se há ou não interseção entre eles. Temos $\frac{N^2}{2}$ pares de objetos e, portanto, o tempo necessário para calcular as interseções deve ser proporcional a N^2. Para muitas aplicações este é um tempo aceitável porque outros fatores podem limitar o número total de objetos a ser considerado. Quando se pensa em circuitos impressos isto não se aplica pois o número de objetos pode ser muito grande e conseqüentemente o algoritmo quadrático é inadequado.

Um método alternativo mais eficiente, utilizado pelo traçador de rotas, foi desenvolvido por M. Shamos [Sha 76] e tem complexidade logarítmica. Durante a execução do traçador estaremos interessados no cálculo da interseção entre conjuntos de segmentos de reta horizontais e verticais que formam polígonos, aqui denominados $polígonos_{x-y}$.

A estratégia usada é imaginar uma linha horizontal de varredura que move-se de baixo para cima no plano. Projetadas sobre esta linha horizontal de varredura, segmentos de reta verticais tornam-se pontos e segmentos de reta horizontais intervalos. Na medida em que a linha horizontal de varredura avança, pontos (representando as linhas verticais) aparecem e desaparecem e, ocasionalmente, segmentos de reta horizontais são encontradas. Uma interseção é encontrada quando um segmento horizontal de reta, representado por um intervalo, contém um ponto que representa a projeção de um segmento vertical de reta. Uma *árvore de intervalos* (Seção 2.2.2) é utilizada para implementar as operações de busca necessárias [Sed 84], [Sha 76] e [Pre 85].

Não é realmente necessário mover uma linha horizontal de varredura através de toda a extensão do plano uma vez que as ações para determinar as interseções somente acontecem quando os extremos de segmentos de reta são encontrados. Pode-se iniciar pela ordenação dos segmentos de reta de acordo

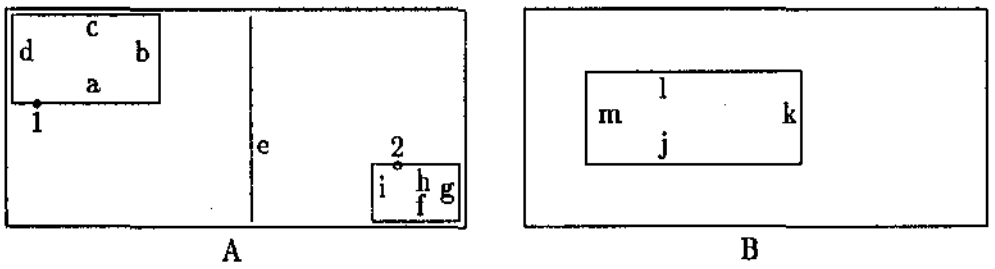


Figura 2.2: Planos de Traçado.

com suas coordenadas y, executando-se em seguida o algoritmo de interseção sobre os segmentos ordenados. Se um extremo inferior de um segmento de reta vertical é encontrado, adicionamos sua coordenada x à árvore de intervalos; se o extremo superior de um segmento de reta vertical é encontrado, retiramos o segmento da árvore de intervalos; e se um segmento de reta horizontal é encontrado procedemos a uma busca de intervalo utilizando suas coordenadas x como intervalo de busca. Um exemplo pode esclarecer o processo.

Para acompanharmos a execução do algoritmo sobre o conjunto de pontos que pertence aos segmentos de reta mostrados na Figura 2.2 (planos A e B), precisamos inicialmente, ordenar os extremos dos segmentos de reta em relação às suas coordenadas y. Cada segmento de reta vertical aparece duas vezes na lista, cada segmento de reta horizontal aparece somente uma vez (Tabela 2.1).

Para o algoritmo de interseção, esta lista ordenada de pontos pode ser imaginada como uma seqüência de comandos: *inserir* (quando o extremo inferior de um segmento de reta vertical é encontrado), *remover* (segmentos de reta verticais, quando seu extremo superior é encontrado) e *buscar* (quando extremos de segmentos de retas horizontais são encontrados) (Tabela 2.1). Todos estes comandos podem ser implementados como procedimentos de busca em árvores binárias.

extremo	e	i	g	f	g	h	i	m	j	k	d	a	b	k	l	m	b	c	d	e
comando	i	i	i	b	r	b	r	i	b	i	i	b	i	r	b	r	r	b	r	r

legenda para os comandos:
i = insira extremo na árvore
b = realize busca de intervalo
r = remova extremo da árvore

Tabela 2.1: Extremos Ordenados de Segmentos de Reta.

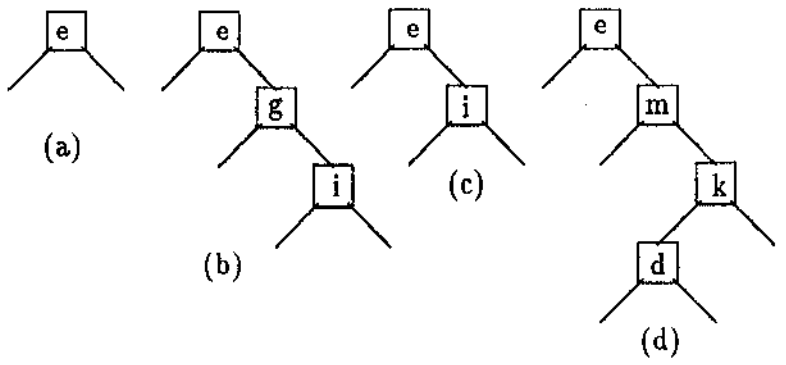


Figura 2.3: Árvores de Intervalo.

Primeiro, *e* é inserido na árvore de intervalos (Figura 2.3a), que se encontrava vazia até então, a seguir são inseridos *i* and *g* (Figura 2.3b). Agora, *f* é encontrado e uma busca de intervalo é realizada usando o intervalo definido pelas coordenadas x de *f*. Esta busca descobre a interseção entre os segmentos *f*, *i* e *g*. Para efeito do algoritmo de traçado estas interseções não tem valor pois ocorrem entre segmentos (arestas) pertencentes ao mesmo $polígono_{x-y}$. Determinar a ocorrência deste tipo de interseção é fácil uma vez que cada ponto armazenado na árvore de busca guarda informação à respeito de seu $polígono_{x-y}$ de origem.

E ainda, se a ordenação dos extremos inferiores dos segmentos de reta *i*, *f* e *g* na Tabela 2.1 fosse *i*, *f* e *g* ao invés de *i*, *g* e *f*, a interseção entre *g* e *f* não seria verificada. Na verdade, o algoritmo verifica, com certeza, todas as interseções onde há um *cruzamento* efetivo entre segmentos de reta que compõem os $polígonos_{x-y}$ e é isto que interessa ao traçador de rotas.

A seguir, o extremo superior de *g* é encontrado e removido (Figura 2.3c); então *h* é encontrado e outra busca é realizada. O processo continua até chegarmos à árvore representada na Figura 2.3d; *j* é encontrado. A interseção encontrada entre *j* e *e* será útil para compor os novos obstáculos. Em seguida, as interseções entre *l* & *e*; *b* & *l* e *m* & *a* são descobertas. Finalmente, os extremos superiores de *d* e *e* (últimos dois pontos na Tabela 2.1) são encontrados e retornamos à árvore vazia.

Complexidade:

O tempo gasto por este algoritmo para determinar as interseções depende do número de interseções encontradas e também do número de segmentos de reta armazenados. As funções de manipulação da árvore de intervalos

tomam tempo proporcional a $\log N$, no caso médio; se árvores balanceadas são utilizadas, então um pior caso de $\log N$ pode ser garantido. O tempo gasto na busca binária depende também do número de interseções encontradas. Assim, o tempo total gasto pelos procedimentos para encontrar interseções é proporcional a $N \log N + I$, onde I é o número de interseções.

Este algoritmo de cálculo de interseções é o algoritmo utilizado para o cálculo dos hiperplanos de traçado, como veremos no Capítulo 4.

Capítulo 3

Algoritmos para Traçado de Rotas

O problema do traçado de rotas é geralmente decomposto nas seguintes fases:

- Determinação da Lista de Conexões;
- Atribuição de Conexões às Faces Condutoras;
- Ordenação de Conexões;
- Traçado propriamente dito.

As três primeiras fases enunciadas acima são executadas para tentar facilitar o posterior traçado das rotas. A união destas três fases é, em geral, denominada pré-processamento.

3.1 Pré-processamento

3.1.1 Determinação da Lista de Conexões

A definição inicial do problema de traçado pode ser feita através de uma lista de sinais e de uma descrição geométrica da placa, dos componentes e de seu posicionamento sobre a placa do circuito. Em sua forma mais geral, a lista de sinais trará uma especificação, para cada sinal, dos possíveis componentes aos quais ele pode ser associado e para cada componente os possíveis terminais que poderão ser atribuídos ao sinal. Poderá haver mais de um componente de mesmo tipo e num mesmo componente poderá haver vários terminais de mesma função. A lista de sinais inicial não especifica, por exemplo, qual dentre 10 terminais logicamente equivalentes foi atribuído ao sinal 1 ou ao sinal 15 de um dado circuito. A idéia é determinar a partir desta lista genérica uma nova lista onde cada sinal está especificado univocamente em função dos componentes e terminais existentes na descrição do circuito.

Para resolver este problema Koren [Kor 72] apresenta um algoritmo que tenta atribuir terminais de componentes ao sinal que está processando de forma a otimizar a direção de traçado do sinal. Suponha que ele esteja processando um sinal que deva ser mantido na horizontal. Dentre os terminais disponíveis atribuirá ao sinal aquele que tiver menor Δy em relação às coordenadas do último terminal processado. Oestreicher [Oes 72] desenvolveu um algoritmo onde as três fases (atribuição, ordenação e traçado) são executadas concomintantemente. Uma discussão simplificada a respeito destes algoritmos pode ser encontrada em [Hig 74].

Após a execução desta fase obtém-se uma lista de conexões que especifica univocamente as conexões a serem feitas. Esta nova lista pode ser especificada como um conjunto de coordenadas (x,y). O próximo problema é interligar os pontos pertencentes a cada sinal.

No atual projeto do traçador o problema da determinação da lista de conexões inexiste pois o ambiente computacional onde este será instalado fornece como entrada uma lista das conexões ponto-a-ponto a serem feitas. Para maiores detalhes sobre o ambiente computacional hospedeiro veja [Int 84a] e [Int 84b].

3.1.2 Atribuição de Conexões às Faces Condutoras

Nesta fase quer-se minimizar o número de conflitos entre as rotas durante a fase de traçado. Pinter [Pin 82] descreve um algoritmo interessante de atribuição de conexões em placas dupla-face. O algoritmo recebe como entrada o resultado do traçado das rotas em uma única face. Neste traçado são permitidos cruzamentos e sobreposições de rotas. Estas inconsistências são denominadas conflitos. Constroe-se então um grafo que descreve estes conflitos e reduz-se o problema de atribuição a um de coloração de vértices em grafos [Luc 79], [Bon 76]. Usa-se uma cor para cada face. Nem sempre é possível colorir o grafo obtido. Neste caso, o traçado de rotas em uma face deve ser simplificado pela remoção de alguns conflitos. Na realidade, determinar se um grafo é ou não 2-colorável é simples. Determinar se um grafo é k-colorável (k fixo, $k \geq 3$) é um problema NP-completo [Luc 79], [Bon 76]. Ciesielski [Cie 81] resolve o mesmo problema através de programação linear inteira.

No presente projeto, optou-se por uma técnica simples de distribuição baseada na inclinação relativa das conexões. Várias observações do comportamento de sistemas comerciais como o [Int 86a,Int 86b,Int 86c] permitem deduzir que estes também devem empregar técnica semelhante de distribuição. O número de faces condutoras disponíveis para traçado determina o número de setores angulares utilizados para classificação e distribuição das conexões. Segue um exemplo. Para uma placa dupla-face seriam utilizados dois setores angulares tomados a 0^o e a 90^o. Assim, as ligações seriam classificadas como horizontais e verticais respectivamente. Seriam classificadas como verticais conexões que tivessem inclinação dentro do setor angular $(90 \pm \Delta)^o$, com $\Delta = 45^o$. Seriam consideradas horizontais as que estivessem no setor angular $(0^o \pm \Delta)^o$. Depois de classificadas as horizontais seriam atribuídas a uma face e as verticais à outra. Se tivéssemos uma placa multi-face com quatro faces condutoras poderíamos utilizar setores angulares com direções-base em torno de 0^o, 45^o, 90^o

e 135° associadas ao Δ adequado.

3.1.3 Ordenação de Conexões

O problema da ordenação de conexões está muito relacionado ao problema da atribuição. Primeiro decide-se onde traçar os conjuntos de conexões e em seguida em que ordem traçá-las, tem-se dois problemas de ordenação vistos por ângulos diferentes. O objetivo dos algoritmos desenvolvidos para solucionar estes problemas é minimizar o número de insucessos durante o traçado das rotas.

Abel [Abe 72] utiliza o algoritmo de Lee [Lee 61] como base para o estudo de vários algoritmos de ordenação de conexões. O algoritmo de Lee foi escolhido por dois motivos. Primeiro, a grande maioria dos sistemas de traçado utiliza o algoritmo de Lee como algoritmo base. Segundo, o algoritmo de Lee é facilmente modificado para incluir os diferentes algoritmos de ordenação. Uma vez traçada uma conexão não é removida, mesmo que posteriormente se descubra que a sua localização é um obstáculo para que outras rotas sejam completadas com sucesso. Assim, a ordem em que elas são traçadas deve ser importante para minimizar conflitos. Abel acumula evidências experimentais que permitem concluir que, no caso genérico, a ordenação das conexões é irrelevante no desempenho do traçador se o critério de avaliação é o número de conexões completadas de comprimento mínimo, no contexto em que foram traçadas.

Os principais métodos de ordenação são [Bre 72], [Hig 74]:

- Ordenação pelo Comprimento
- Ordenação pela Densidade de Conexões
- Ordenação Adaptativa

Ordenação pelo Comprimento

Muitos dos processos de ordenação incorporam o comprimento da conexão no cálculo da ordem e, geralmente, muitos dos procedimentos de traçado encorajam o traçado das conexões mais curtas primeiro (entretanto há situações em o inverso fornece melhores resultados). Hightower [Hig 74] determinou que o comprimento e inclinação são critérios importantes para ordenação. Esta conclusão é confirmada por Ginsberg [Gin 69]. Numa série de experimentos, os melhores resultados foram encontrados traçando-se as conexões em ordem crescente de $v = \Delta x + 10\Delta y$. Esta ordenação encoraja a ligação de conexões

curtas primeiro e penaliza mudanças significantes na direção y, de modo que dentre as conexões curtas as mais horizontais irão primeiro. Por outro lado, uma conexão vertical curta irá antes de uma horizontal longa.

Ordenação pela Densidade de Conexões

Aqui determina-se a densidade de conexões nas áreas próximas dos pontos a serem interligados. A área é obtida traçando-se um retângulo de dimensões $\Delta x \times \Delta y$, denominado retângulo mínimo, que tem os pontos a serem interligados como extremos de uma de suas diagonais. Em seguida, as conexões recebem um peso em função da densidade de rotas existente na vizinhança dos pontos a serem interligados. Em [Hig 74] encontra-se um exemplo de função para calcular a densidade. Baseado nos pesos atribuídos às conexões internas ao retângulo mínimo o algoritmo traçará primeiro as conexões curtas de regiões densas, seguidas por conexões curtas em regiões pouco densas e conexões longas em regiões densas, seguidas por conexões longas em regiões pouco densas. A idéia básica deste método é traçar a rota que une os pontos da conexão antes que um deles se torne inacessível por estar circundado por outras conexões.

Uma maneira mais simples para avaliar-se a densidade relativa de conexões dentro de um retângulo mínimo consiste em contar-se o número de extremos de conexões existente dentro do retângulo. Este valor é associado à conexão analisada, determinando a ordem em que ela é traçada (Tabela 3.1 e Figura 3.1).

Ordenação Adaptativa

Existe um denominador comum entre os esquemas de ordenação apresentados até aqui: ordenam as conexões sem levar em conta os efeitos provocados pela resolução das rotas durante o processo de traçado. Claramente, a melhor ordem de traçado muda - possivelmente muito - durante a fase de traçado. Uma idéia, talvez melhor, é considerar qual deve ser a próxima conexão a ser tentada na medida em que as conexões estão sendo realizadas. Uma maneira de implementar esta técnica de ordenação é atribuir inicialmente um valor v_i a cada conexão i e então escolher a melhor ligação a fazer em função deste valor. Após a realização de um determinado número de conexões os valores v_i para as conexões serão recalculados determinando uma nova ordenação que será utilizada a partir deste ponto. Um bom conjunto de valores para os v_i é a distância do ponto a ser traçado ao trecho de sinal já traçado (Tabela 3.2 e Figura 3.2).

ordenação		
conexão	número de pontos internos ao retângulo mínimo	ordem
cc	0	(1)
bb	1	(2)
dd	2	(3)
aa	5	(4)

Tabela 3.1: Ordenação por Retângulos Mínimos.

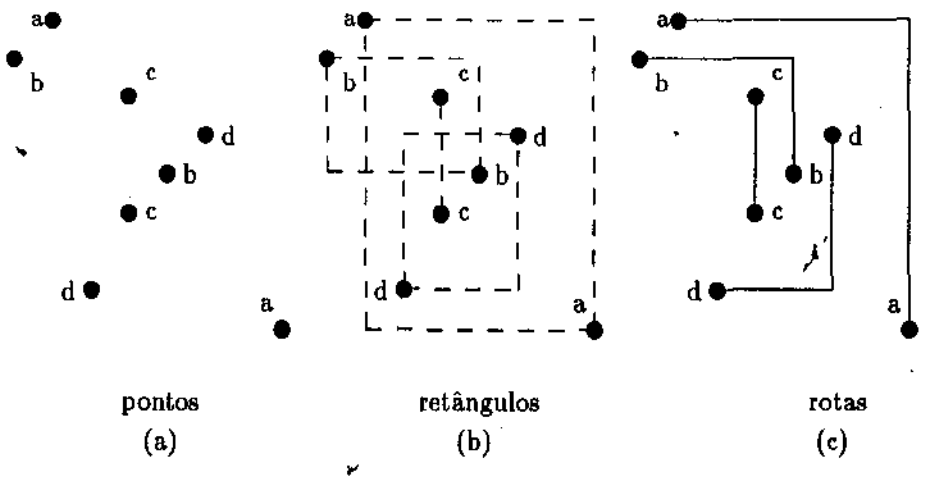


Figura 3.1: Ordenação Utilizando Retângulos Mínimos.

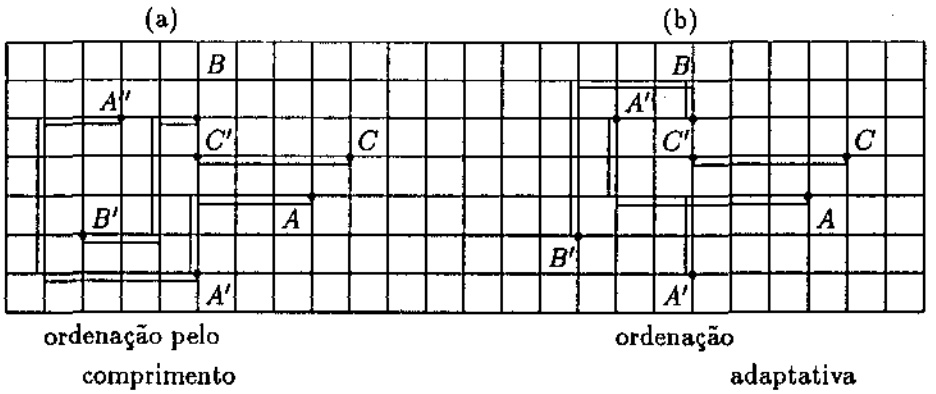


Figura 3.2: Ordenação Adaptativa de Conexões.

ordenação inicial			ordenação após o traçado de CC' e AA'		
conexão	valor	ordem	conexão	valor	ordem
CC'	4	(1)	AA''	4	(1)
AA'	5	(2)	BB'	6	(2)
BB'	6	(3)			
AA''	6	(4)			

Tabela 3.2: Ordenação Adaptativa das Conexões.

3.2 Traçado

É interessante notar que as descrições dos algoritmos para traçado de rotas normalmente encontrados na literatura acadêmica não são diretamente aplicáveis a situações práticas. As descrições das implementações são propositadamente superficiais devido ao seu valor econômico.

Os algoritmos para traçado de rotas podem ser classificados da seguinte forma:

- labirinto[1]
- propagação de linha[2]
- confinados em canais[3]

3.2.1 Algoritmos de Labirinto

Algoritmos de labirinto representam a face condutora através de um reticulado, isto é, a face é dividida por um conjunto de retas verticais e horizontais com distância constante entre retas paralelas adjacentes. Denomina-se passo a esta distância constante e célula ao menor quadrado definido pelo reticulado. O lado deste quadrado mede um passo e é definido em função das regras de projeto decorrentes da tecnologia de fabricação adotada. Um caminho entre duas células dadas, denominadas extremos, é um conjunto de células adjacentes, ainda não bloqueadas por outras ligações, que inclue os extremos, de modo que cada célula não extremo é adjacente a no máximo outras duas células não extremo.

O algoritmo de labirinto desenvolvido por Lee [Lee 61] é considerado um marco na área de traçado de rotas; foi inicialmente aplicado em placas de uma face e para interligar pinos em uma ordem estabelecida previamente. Esse algoritmo é descrito a seguir. Considera-se um dos extremos como a origem e o outro como o destino. À célula origem é assinalado o rótulo zero. Inicia-se então o processo de propagação de uma onda que circunda obstáculos. No primeiro movimento da onda todas as células adjacentes a origem, não bloqueadas e não rotuladas, são rotuladas com o valor um. Nos movimentos subseqüentes da onda continua-se o processo de rotulação de modo semelhante, com valores de rótulos crescentes, até que não seja mais possível expandir a onda ou até que a célula destino seja rotulada (Figura 3.3a).

[1] Em inglês: Maze Routers.
[2] Em inglês: Line Propagator Routers.
[3] Em inglês: Channel Routers.

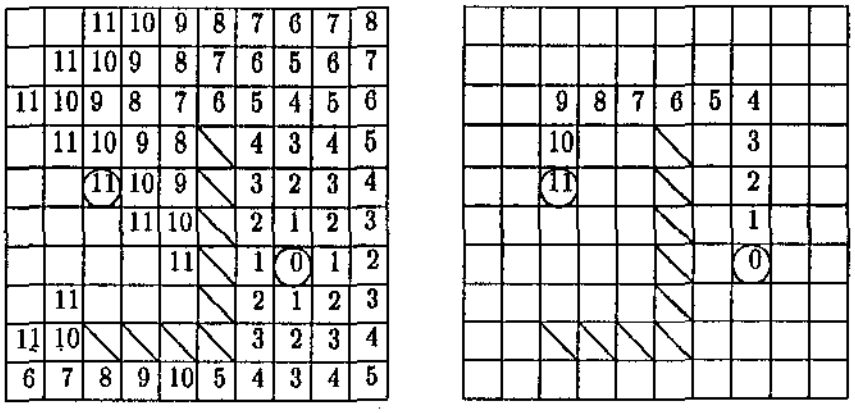
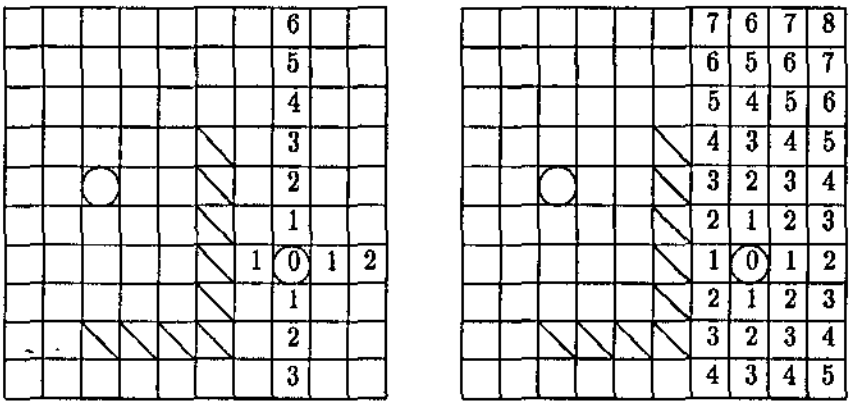


Figura 3.3: Algoritmo de Lee.

Caso a célula destino tenha sido rotulada então existe um caminho entre a origem e o destino. O algoritmo de Lee executa então um retrocesso[4], marcando a célula destino, de rótulo d, como usada e escolhendo em seguida uma de suas células adjacentes, de rótulo $d-1$, marcando-a também como usada. Esta nova célula passa a ser a última do caminho. Repete-se o processo para a última do caminho até que se encontre a célula origem (Figura 3.3b).

Os problemas relacionados a esta classe de algoritmos já são bem conhecidos [Hig 74] e estão centrados na estrutura de dados utilizada para representar o reticulado que é custosa, e no procedimento utilizado para resolver as conexões. Já se provou [Hig 83] que a fase de expansão de conexões do algoritmo de Lee consome, em média, em torno de 90% do tempo de execução. Este tempo pode ser reduzido a aproximadamente a metade se a expansão for feita somente na vertical e na horizontal (a partir da célula origem) [Hig 74]. A partir da célula origem expande-se a onda ao longo de verticais e horizontais até que se encontre algum obstáculo. Em seguida, escolhe-se o mais longo dos eixos determinados na expansão e para cada célula ao longo dele executa-se uma nova expansão, na perpendicular (Figura 3.4).

[4] Em Inglês: backtracking.



(a) primeira expansão

(b) segunda expansão

Figura 3.4: Exemplo de Modificação para o Algoritmo de Lee.

As propriedades importantes do algoritmo de Lee (mantidas por algumas das modificações propostas) são as seguintes:

1. se existem caminhos entre duas células do reticulado para uma dada configuração de conexões na face, então um destes caminhos é encontrado;
2. o caminho encontrado tem comprimento mínimo no contexto em que foi determinado.

Hightower [Hig 83] desenvolveu uma versão do algoritmo para traçado de "gate arrays" com bons resultados. Seu algoritmo associa custos não unitários a cada célula do reticulado de modo a encorajar ligações em algumas direções e em outras não. Abordagem semelhante já havia sido utilizada por Korn [Kor 82]. O seu algoritmo reduz significativamente o tempo gasto na expansão mas não garante a minimalidade do caminho encontrado. Na realidade, idéias sobre a utilização de custos não unitários para obter a melhor direção de traçado já haviam sido sugeridas pelo próprio Lee [Lee 61].

Aranoff [Ara 81] confina o processo de expansão dentro de uma sub-área do plano de traçado, denominada janela. Se um caminho não é encontrado dentro da janela então a conexão é considerada impossível. Isto implica o teste de um menor número de células e pode aumentar a eficiência do algoritmo. Uma abordagem análoga suscitou o desenvolvimento de traçadores confinados em canais, tais como [Las 69], [Sut 69] e [Tad 80].

3.2.2 Algoritmos de Propagação de Linhas

Para tentar solucionar as deficiências dos diversos algoritmos de labirinto alterou-se a abordagem do problema. Uma nova representação do plano de traçado foi introduzida. O plano passa a ser representado por um plano cartesiano, contínuo, em contraposição à representação discreta adotada pelos algoritmos de labirinto. As seguintes vantagens são obtidas: não existe, teoricamente, um limite para a precisão utilizada para descrever os pontos no plano de traçado. Na prática, o único fator restritivo é a precisão da aritmética de reais (inteiros) da máquina hospedeira do algoritmo. Os algoritmos de propagação de linha armazenam somente segmentos de reta. Uma conseqüência imediata disso pode ser avaliada pela comparação do número de células necessárias para representar-se uma placa de 9×9 polegadas com precisão de traço de 1 mil usando reticulados, cada face corresponderia a 81×10^6 células, enquanto que para um algoritmo de propagação de linhas a memória requerida é uma função do número de pontos representados. A quantidade de pontos usados pelo último é, em geral, muito inferior pois é proporcional ao número de conexões realizadas. A desvantagem principal desses algoritmos é a necessidade de se fazer a verificação dos parâmetros de projeto (Capítulo 1) durante a propagação. Nos algoritmos de labirinto esses parâmetros de projeto são garantidos através da especificação do reticulado.

Estes algoritmos representam os componentes e outros obstáculos, inclusive conexões já traçadas, como segmentos de reta. Em muitos casos, os segmentos de reta horizontais e verticais estão localizados em planos (faces) diferentes. Em todas as descrições supõe-se que os sinais são compostos por dois terminais somente; a extensão para sinais de múltiplos terminais não é difícil.

Hightower [Hig 69] propôs um dos dois algoritmos que apresentam as principais características dessa classe de algoritmos. Seu algoritmo, denominado algoritmo de busca de linha[5], não garante que uma solução seja encontrada caso exista, e não garante que seu comprimento é mínimo quando a encontra. Em contrapartida, o algoritmo tem desempenho superior, em espaço e tempo, ao apresentado pelos algoritmos de labirinto.

São traçados dois segmentos de reta, perpendiculares entre si, passando pelos pontos que representam os dois terminais a serem interligados. Seu comprimento deve ser o maior possível e não devem cruzar qualquer obstáculo (Figura 3.5a). Se estes segmentos se cruzam então uma solução foi encontrada. Se não, então os maiores segmentos de reta, perpendiculares aos últimos traçados, devem ser traçados (Figura 3.5b). Esse processo é repetido até que

[5]Em Inglês: Line Search Router.

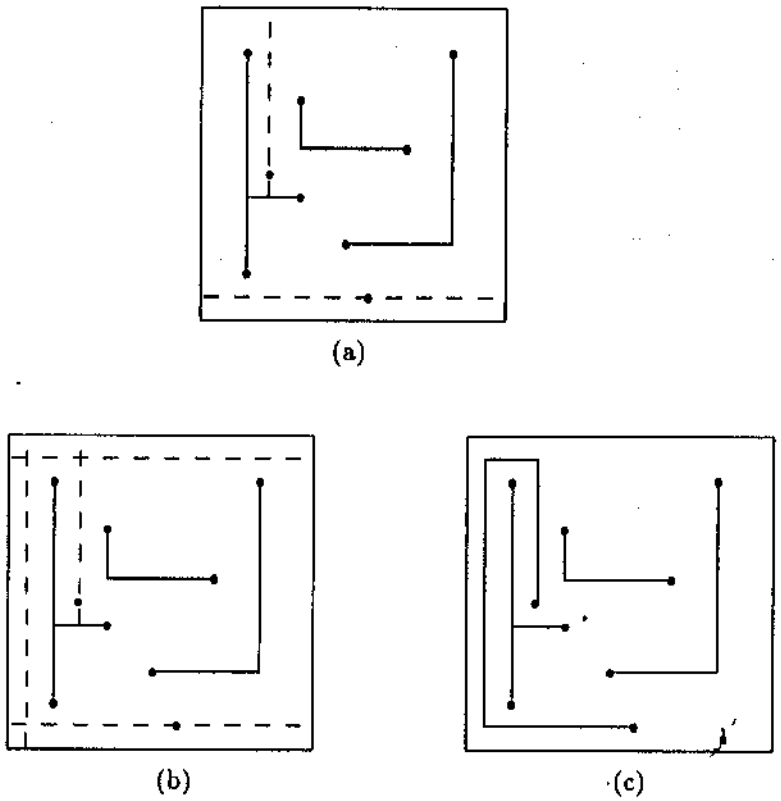


Figura 3.5: Algoritmo de Busca de Linhas.

dois segmentos se interceptem ou até que não se possa mais traçar novos segmentos. No último caso não se obteve uma solução. O algoritmo falhou. Nos casos de sucesso este algoritmo gera rotas com o menor número de quebras (Figura 3.5c). Higtower refere-se aos segmentos de reta traçados como *linhas de fuga*.

O outro representante dos algoritmos de propagação de linhas foi proposto por Heyns [Hey 80], um algoritmo de expansão de retas. Inicialmente, um segmento de reta é traçado pelo ponto que representa o terminal escolhido como origem do sinal (Figura 3.6a). Então, o contorno da região, determindado pelo feixe de retas de fuga (não por somente uma como no algoritmo anterior) que cruzam a reta original, é demarcado (Figura 3.6b). Segue-se a determinação de novos contornos (áreas) que podem ser alcançadas por todas as retas de fuga

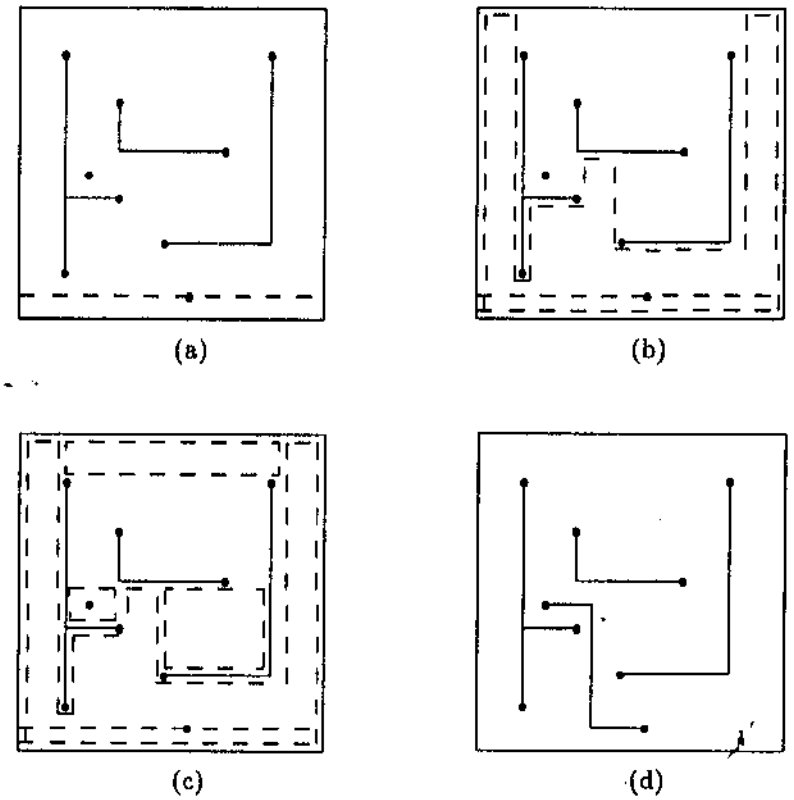


Figura 3.6: Algoritmo de Expansão de Retas.

perpendiculares às retas de contorno obtidas no passo anterior (Figura 3.6c). O processo é repetido até que o ponto que representa o terminal destino seja incorporado por uma região ou então, até que não seja mais possível demarcar novas regiões. Quando há sucesso o ponto destino foi incorporado e tem início um retrocesso que define a rota definitiva (Figura 3.6d). Este algoritmo sempre encontra uma solução, se ela existe, porque todas as alternativas são verificadas. Permanece o problema das interligações serem realizadas uma a uma e de, durante o traçado, não se utilizar um critério adaptativo para ordená-las. Uma conseqüência imediata desse fato é que rotas já traçadas podem isolar terminais ainda por conectar, impedindo totalmente sua ligação.

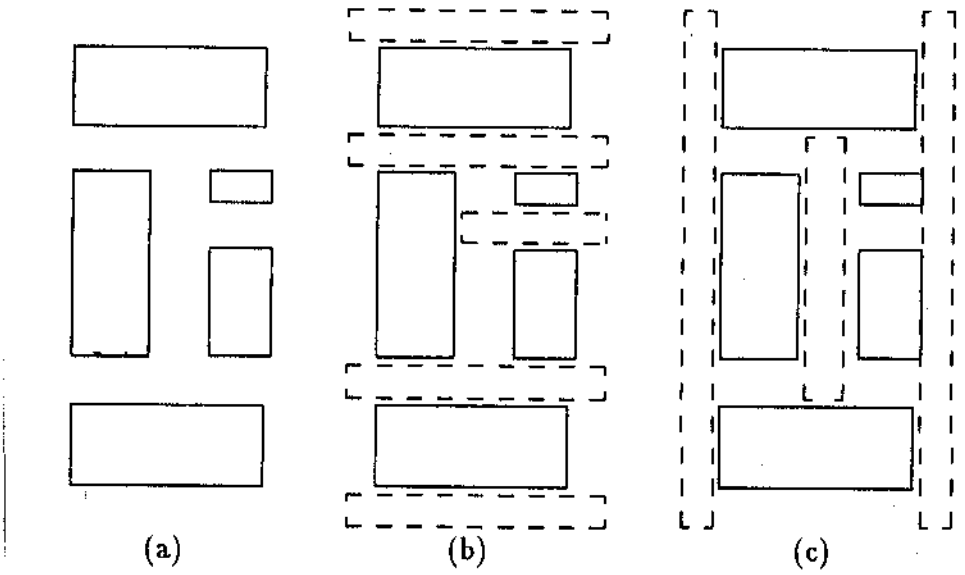


Figura 3.7: Canais. (a) Componentes. (b) Canais Horizontais. (c) Canais Verticais.

3.2.3 Algoritmos Confinados em Canais

Traçadores confinados em canal tratam de um caso especial do problema de traçado de rotas onde as interconexões devem ser executadas dentro de uma região restrita da área de traçado. Esta região, denominada *canal*, tem forma, em geral, retangular com os terminais de componentes localizados em lados opostos do retângulo.

Esta configuração geométrica particular surge frequentemente no projeto de circuitos integrados [Lie 86], onde as células que compõem o circuito são posicionadas em configurações regulares, em linhas e colunas, formando entre si canais abertos por onde as conexões devem ser traçadas (Figura 3.7).

Em 1983, Burstein e Pelavin [Bur 83] apresentaram um algoritmo hierárquico que ilustra as técnicas utilizadas para resolver o problema de traçado confinado em canais. A grande maioria dos algoritmos existentes utiliza o princípio de dividir-para-conquistar[6] para realizar as conexões.

[6]Em Inglês: divide and conquer.

A cada interação o canal é subdividido em dois outros subcanais e metade das conexões é atribuída a cada subcanal. Se mais de uma conexão é atribuída a cada subcanal o traçador é ativado novamente para resolver este subproblema. Quando somente uma conexão é atribuída a um subcanal então esta é resolvida, se possível. O algoritmo de Burstein e Pelavin supõe que a distribuição das conexões dentro dos canais é uniforme e trabalha somente em duas faces, uma é utilizada para traçar os segmentos horizontais e a outra os segmentos verticais.

Hightower e Boyd [Hig 80] subdividem o problema em duas fases: a de traçado global, onde conexões entre canais são avaliadas, e a de traçado confinado nos canais. O modelo utilizado para representar as conexões entre canais é um grafo onde os vértices são os canais e arestas com peso ligam vértices quando estes representam canais adjacentes. O peso atribuído às arestas é função da ocupação do canal e do comprimento médio esperado para as conexões que o utilizarem.

Um refinamento foi introduzido no modelo acima para aumentar a informação sobre os canais. Boyd sugeriu que cada canal fosse subdividido em subcanais definidos pelas dimensões dos lados das células que têm terminais a serem conectados através dele.

O grafo anterior é substituído por um grafo orientado com peso nas arestas. Os novos vértices são os subcanais e uma aresta liga dois vértices se eles representam canais adjacentes. Como pode haver conexões que atravessem as células o grafo pode, eventualmente, incluir arestas que representam estas conexões intra-celulares.

A partir do grafo árvores ótimas são obtidas para todos os sinais. A cada árvore obtida o grafo tem os pesos das arestas recalculados para refletir a nova ocupação dos canais. O traçado propriamente dito é realizado por um traçador semelhante ao desenvolvido por Hightower [Hig 80]. Várias otimizações são possíveis pelo fato do traçado estar confinado a um canal. Os parâmetros empregados para a verificação das regras de projeto são os mesmos utilizados em trabalhos pioneiros na área por Hashimoto e Stevens [Has 71] e Deutsch [Deu 76].

O trabalho de Hashimoto e Stevens [Has 71] é conhecido como o algoritmo do "lado esquerdo". Para um canal horizontal com pontos terminais colocados nos lados superior e inferior uma busca é iniciada pela esquerda. Tanto o lado superior quanto o inferior são "varridos" em busca de um primeiro terminal pertencente ao sinal que está sendo traçado. O primeiro terminal encontrado é interligado a todos os outros do mesmo sinal traçando-se a maior conexão horizontal possível no subcanal livre mais superior. As conexões horizontais

são traçadas em uma face e as verticais em outra. O processo se repete até que todos os sinais tenham sido traçados ou até que seja impossível interligar algum dos pontos terminais do sinal selecionado.

Deutsch [Deu 76] permite o traçado de sinais contendo mais de dois terminais. Os sinais são decompostos em sinais de dois terminais e supõe-se a existência de duas faces de traçado, uma para as conexões horizontais e a outra para as verticais. Quebrando-se os sinais originais em sinais de dois terminais é possível, no caso de um canal horizontal, traçar o sinal original em alturas diferentes, e não através da maior horizontal possível. Os segmentos horizontais são interligados por segmentos verticais traçados na outra face. Este traçado é denominado "dogleg". O algoritmo de Deutsch obtém canais com maior densidade de conexões, sua ineficiência está no fato de precisar de muitos furos de transpasse pois as conexões frequentemente mudam de face de traçado.

Capítulo 4

Algoritmo Proposto

Neste Capítulo descreve-se o algoritmo proposto para traçado de rotas em placas de circuitos impressos multi-face. Primeiro, o procedimento utilizado para atribuição de conexões às faces condutoras é detalhado. Em seguida, descreve-se o procedimento para traçado propriamente dito, partindo-se de uma descrição válida para o traçado em uma face. Esta descrição serve como base para entendimento do algoritmo generalizado, que utiliza conceitos normalmente aplicados em algoritmos geométricos.

4.1 Árvores Ótimas

O cálculo das árvores ótimas é realizado via algoritmo de Kruskal, como descrito no Capítulo 2. Após a obtenção das árvores ótimas para cada sinal a ser resolvido utiliza-se um critério angular para a atribuição de conexões às faces condutoras.

4.2 Atribuição de Conexões às Faces Condutoras

A atribuição de conexões às faces condutoras (planos de traçado) é realizada conforme algoritmo proposto na Seção 3.1.2.

Para realizar a classificação angular precisamos de uma função para o cálculo da inclinação da aresta (conexão ponto a ponto). Como os extremos das arestas estão relacionados a pontos no plano de traçado o ângulo de que necessitamos pode ser expresso por:

$$\tan^{-1}\left(\frac{dy}{dx}\right)$$

Para obter o arco-tangente pode-se pensar na utilização de funções trigonométricas pré-definidas nas bibliotecas de linguagens de programação mas isto traz algumas desvantagens: o cálculo pode ser muito lento, é necessário testar a condição $dx = 0$ (divisão por zero) e é necessário testar em que quadrante os pontos se localizam. Uma vez que, tudo o que se quer é um número que permita a classificação (ordenação) das inclinações das retas para posterior separação em setores angulares, pode-se utilizar uma função aproximada da função arco-tangente. No algoritmo proposto utiliza-se a seguinte função:

$$\Theta(dx, dy) = \frac{dy}{|\,dx\,| + |\,dy\,|}$$

O teste de condições de contorno ainda é necessário mas muito simples. Assim sendo, a função Θ abaixo retorna um valor real entre 0 e 360. Este valor não é um ângulo mas permite decidir em que setor angular uma dada aresta se encontra. A Tabela 4.1 mostra os valores retornados pela função para alguns ângulos facilmente identificáveis.

$$\Theta(dx,dy) = \begin{cases} 0 & \text{se } dx = 0 \wedge dy = 0 \\ 90\left(\frac{dy}{|dy|+|dx|}\right) & \text{se } dx > 0 \wedge dy > 0 \\ 90\left(2 - \frac{dy}{|dy|+|dx|}\right) & \text{se } dx < 0 \\ 90\left(4 + \frac{dy}{|dy|+|dx|}\right) & \text{se } dy < 0 \wedge dx > 0 \end{cases}$$

dx	dy	$\Theta(dx,dy)$
0.0	0.0	0.0
4.0	1.0	18.0
4.0	4.0	45.0
1.0	4.0	72.0
-2.0	3.0	126.0
-3.0	-3.0	225.0
1.0	-4.0	288.0
3.0	-3.0	315.0

Tabela 4.1: Exemplos de valores de $\Theta(dx,dy)$.

4.3 Traçado de Rotas

4.3.1 Traçado em uma Face

O algoritmo básico de expansão de retas, descrito a seguir, opera sobre uma única face de traçado. A aplicação deste algoritmo, associada a um outro para determinação de interseções entre áreas-de-expansão resulta na generalização necessária para o traçado de placas multi-face.

Estruturas de Dados

A placa será caracterizada, de modo abstrato, como sendo um conjunto de planos contendo obstruções, componentes e conexões. Neste contexto, componentes e obstruções são obstáculos retilíneos, isto é, polígonos contendo somente segmentos de reta verticais e horizontais como arestas. Normalmente, polígonos deste tipo são denominados *polígonos*$_{x-y}$. Conexões são simplesmente segmentos de reta verticais e/ou horizontais interligando componentes. Um *delta* (δ) especifica a distância mínima entre conexões adjacentes[1]. O traçado de conexões sob componentes não é permitido.

Os segmentos de reta horizontais (verticais) são armazenados em listas ligadas. Segmentos de reta horizontais (verticais) de mesma coordenada x (y) são armazenados em uma mesma lista ligada, ordenados segundo a primeira coordenada y (x). Os segmentos de reta que representam as bordas da placa também são armazenados nas estruturas [Buz 88].

O Algoritmo de Expansão de Retas

Este procedimento expande retas perpendiculares a uma *reta-base* colocada como reta referencial de expansão. A *área-de-expansão* pode ser descrita como a máxima área coberta pelas perpendiculares traçadas em relação a reta-base. Um ponto da área-de-expansão é dito *coberto* pela área se puder ser interceptado por alguma das perpendiculares à reta-base.

Após cada expansão, o perímetro da área-de-expansão é determinado e os segmentos de retas perimetrais não obstruídos serão as novas retas-base utilizadas para continuar o processo de expansão de retas. No algoritmo estes segmentos são armazenados no vetor **expansíveis**.

A expansão é realizada no algoritmo proposto a partir da reta com coordenadas (y_1, x_1, x_2) (Figura 4.1). Para simplificar, as coordenadas do plano de traçado são consideradas inteiras e também o valor de δ. O algoritmo sempre pára, pois as bordas do plano sempre impõem um fim à expansão, caso uma outra obstrução não o tenha feito antes.

Para descrever os algoritmos adota-se uma notação próxima à linguagem Pascal; as funções e procedimentos básicos utilizados na descrição não são detalhados pois seu significado é facilmente deduzido.

[1] Os *deltas* são definidos em função do projeto elétrico dos circuitos para evitar interferências entre sinais.

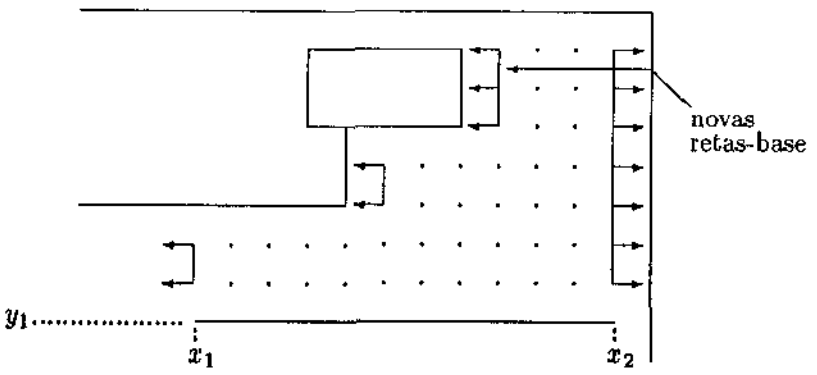


Figura 4.1: Expansão da reta-base (y_1, x_1, x_2) na direção "para cima".

```
{ Algoritmo de Expansão de Reta }
expansão_de_reta(reta_base,obstruções);
início
   { Atribuições iniciais. }
   i_obstruções ← 0; i_expansíveis ← 1;
   expansíveis[i_expansíveis] ← (x1, x2); y_aux ← y1;
   se direção_expansão = PARA_CIMA
      então delta ← δ
      senão delta ← −δ;
{ Recupera a próxima reta de coordenada y_aux > y1. }
   reta_recuperada ← prox_reta(y_aux,delta);
   repita
      enquanto ¬ bloqueio(reta_recuperada,expansíveis)
         faça reta_recuperada ← prox_reta(y_aux,delta);
{ Suponha que o bloqueio entre a
      reta_recuperada e expansíveis[i_expansíveis] = (x3, x4)
            ocorre na região de x5 a x6 (fig. 4.2). }
      i_obstruções ← i_obstruções + 1;
{ Armazene as coordenadas do obstáculo que determina o fim da expansão. }
      obstruções[i_obstruções] ← (y_aux,x5, x6);
{ Atualize expansíveis de modo a permanecerem somente os segmentos
      de retas ainda expansíveis. }
      se x3 ≠ x5 então
         início
            i_expansíveis ← i_expansíveis + 1;
            expansíveis[i_expansíveis] ← (x3, x5 − δ)
```

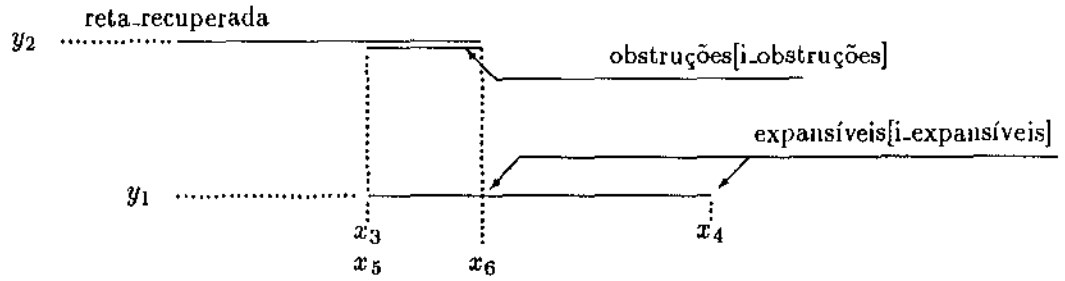


Figura 4.2: Determinação das obstruções.

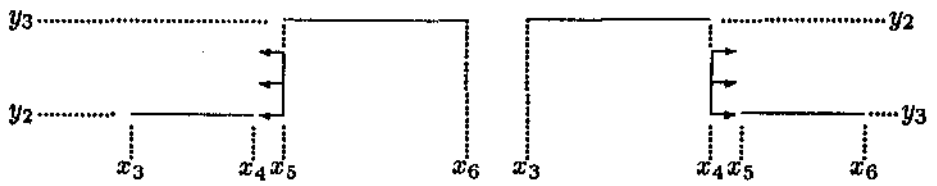
```
        fim;
    se x4 ≠ x6 então
        início
            i_expansíveis ← i_expansíveis + 1;
            expansíveis[i_expansíveis] ← (x6 + δ, x4)
        fim;
    { Retire de expansíveis os segmentos cuja interseção já foi verificada. }
    ajuste(i_expansíveis)
  até que i_expansíveis = 0
fim.
```

Utilizando a informação do vetor **obstruções**, podemos gerar as retas-base necessárias para continuar o processo de expansão. A direção de expansão também é determinada e armazenada junto a cada reta-base.

O algoritmo para geração das retas-base será descrito a seguir.

```
{ Algoritmo para geração de retas-base }
geração_de_retas_base(obstruções,retas_base,número_retas_base);
início
    { Atribuições iniciais. }
    i_retas_base ← 0;
    { Adição de pseudo obstruções ao vetor obstruções. }
    i_obstruções ← i_obstruções +1;
    obstruções[i_obstruções] ← (y1, x1 − δ,x1 − δ);
    i_obstruções ← i_obstruções +1;
    obstruções[i_obstruções] ← (y1, x2 + δ,x2 + δ);
    { Ordene o vetor obstruções, em ordem crescente, de acordo com as
      coordenadas x (inferiores) de cada segmento de reta. }
```

```
ordenação(obstruções);
{ Gere para cada par sucessivo de segmentos armazenados
em obstruções uma nova reta-base. }
enquanto recupera_par(obstruções) faça
{ Recupere pares de segmentos de reta da seguinte forma:
        primeiro par: obstruções[1], obstruções[2],
        segundo par: obstruções[2], obstruções[3],
        assim sucessivamente. }
   início
{ Suponha que o par recuperado contém segmentos de retas de coordenadas
        (y2, x3, x4) e (y3, x5, x6); nesta ordem. }
   dy1 ← y2 − y3;
   caso dy1 for
      negativo: { dy1 < 0 (fig. 4.3a). }
         { Geração de reta-base à esquerda. }
         início
            se y1 = y2 então dy2 ← δ
                  senão dy2 ← 0;
            se y3 ≠ y2+ dy2 então
               início
                  i_retas_base ← i_retas_base +1;
                  retas_base[i_retas_base] ← (x5, y2+ dy2, y3 − δ)
               fim
         fim;
      positivo: { dy1 > 0 (fig. 4.3b). }
         { Geração de reta-base à direita. }
         início
            se y1 = y3 então dy3 ← δ
               senão dy3 ← 0;
            se y2 ≠ y3 + dy3 então
               início
                  i_retas_base ← i_retas_base + 1;
                  retas_base[i_retas_base] ← (x4, y3+ dy3, y2 − δ)
               fim
         fim
   fim { do caso }
fim; { enquanto }
   número_retas_base ← número_retas_base + i_retas_base
fim.
```



(a) (b)

Figura 4.3: Geração de retas-base. (a) para a esquerda. (b) para a direita.

4.3.2 O Algoritmo Básico de Expansão de Retas

O objetivo principal do algoritmo básico de expansão de retas é interligar pontos eletricamente equivalentes de um mesmo plano. Uma pequena modificação pode habilitá-lo a encontrar áreas de expansão existentes num plano.

Inicialmente, busca-se uma solução que interligue os pontos desejados. Caso não seja possível a interligação, então outros planos serão investigados. Na Seção 4.3.4, as idéias centrais a respeito do algoritmo generalizado são discutidas. O algoritmo básico de expansão é apresentado a seguir.

```
{ Algoritmo Básico de Expansão de Retas }
algoritmo_básico(plano,ponto_origem,ponto_destino);
início
{ Geração das retas-base iniciais. }
   número_retas_base ← 1;
   retas_base[0] ← reta contendo o ponto_origem;
   enquanto número_retas_base > 0 ∧ ¬ solução faça
      início
         número_retas_base ← seleciona_reta_base(retas_base,reta_base);
         expansão_de_reta(reta_base,obstruções);
         geração_de_retas_base(obstruções,retas_base,número_retas_base);
         solução ← verifica_solução(reta_base,ponto_destino);
      fim
fim.
```

Foram desenvolvidos procedimentos heurísticos para impedir que uma área de expansão já investigada seja verificada novamente durante o processo de expansão. Estes procedimentos não são descritos porque se referem a uma otimização irrelevante ao entendimento do algoritmo generalizado.

4.3.3 Garantia de Solução

O processo de busca por uma solução obriga a expansão das retas base obtidas na última iteração se o ponto destino não está coberto pela área-de-expansão recém determinada. É possível verificar que o fato acima assegura que uma solução é encontrada, se existir.

Suponha que são dadas:

- uma rota interligando o ponto origem ao ponto destino,
- uma reta-base, que intercepta a rota perpendicularmente em algum de seus pontos.

Então temos duas possibilidades a considerar. Primeiro, suponha que o segmento terminal da rota está interno à área-de-expansão determinada pela reta-base dada. Neste caso, o ponto destino também está coberto pela área-de-expansão e, por definição, pode ser encontrado. Na segunda possibilidade, o ponto destino não está coberto pela área-de-expansão. Isto significa que a rota atravessa os limites da área-de-expansão por alguma das retas-base geradas. Somente o ponto de cruzamento, entre a rota dada e a reta-base interceptada a partir da reta-base dada é relevante. Este é o ponto em que a rota deixa a área-de-expansão, e portanto, pertence a uma reta-base interceptada pela rota; retornamos à situação inicial. Se persistirmos na execução do procedimento acima reconstruiremos a rota dada.

No entanto, no caso real, a rota ainda não é conhecida, mas a expansão da reta-base inicial cobre o primeiro segmento de reta pertencente a rota solução, se ela existir. As sucessivas expansões das retas-base geradas a partir da inicial acabam por cobrir os segmentos de rota que compõem a solução. Inicialmente, não se sabe qual das retas-base encontradas numa iteração deve ser expandida para que se encontre a solução, logo, o procedimento de expansão deve, a priori, investigar todas as retas-base obtidas em cada iteração.

Mostramos assim que o algoritmo básico de expansão de retas sempre encontra uma solução, se ela existir.

4.3.4 O Algoritmo Generalizado

A seguir esboçamos os princípios utilizados na generalização do algoritmo básico de expansão de retas [Buz 89].

Algoritmos Geométricos

O projetista de circuitos impressos tem uma intuição geométrica de como interligar os pontos pertencentes a um sinal pois pode avaliar melhor a sua topologia, a distribuição geométrica dos obstáculos e interligações já realizadas. Algoritmos geométricos lidam exatamente com esta classe de problemas, preocupando-se em resolvê-los de forma eficiente.

A generalização do algoritmo básico de expansão de retas para múltiplos planos é obtida pela utilização de um algoritmo que fornece as interseções entre áreas-de-expansão obtidas em planos de traçado diferentes. Como áreas-de-expansão são $polígonos_{x-y}$ o algoritmo descrito na Seção 2.4 pode ser utilizado para o cálculo das interseções.

A Generalização

A generalização baseia-se na aplicação integrada de ambos os algoritmos introduzidos até aqui; o algoritmo básico de expansão de retas e o algoritmo para cálculo de interseções. Inicialmente, busca-se uma solução que esteja restrita ao plano que contém os pontos a serem interligados. Caso esta busca falhe, são determinadas áreas-de-expansão para o plano corrente e para um outro adjacente a ele. Em seguida, são calculadas as interseções entre as áreas-de-expansão dos planos distintos e um novo plano, denominado *plano-solução*, é formado pela união das áreas-de-expansão com interseções não nulas obtidas.

Se for possível interligar os pontos origem e destino no plano-solução, então é possível encontrar uma rota, com trechos em planos diferentes, que conecta os pontos desejados. Um algoritmo de retrocesso[2] é utilizado na determinação da rota final. Se, por outro lado, a interligação dos pontos no plano-solução não for possível, então um novo plano é selecionado e o procedimento acima é executado novamente, levando em conta as áreas-de-expansão do plano-solução e as obtidas para o plano selecionado. O algoritmo pára quando se esgotarem todos os planos disponíveis para investigação.

Um exemplo ilustra os princípios discutidos acima. Suponha dois planos A e B (Figura 4.4). Claramente, não existe uma rota entre os pontos 1 e 2 do plano A. O algoritmo básico pára, devido a presença de bloqueios. Em seguida, áreas-de-expansão são determinadas (áreas tracejadas da Figura 4.4) para os planos A e B e o plano-solução é encontrado (Figura 4.5). É possível interligar os pontos 1 e 2 no plano-solução, portanto, é possível determinar uma rota, composta por trechos em planos distintos, que os conecta. A interligação entre

[2]Em Inglês: Backtracking.

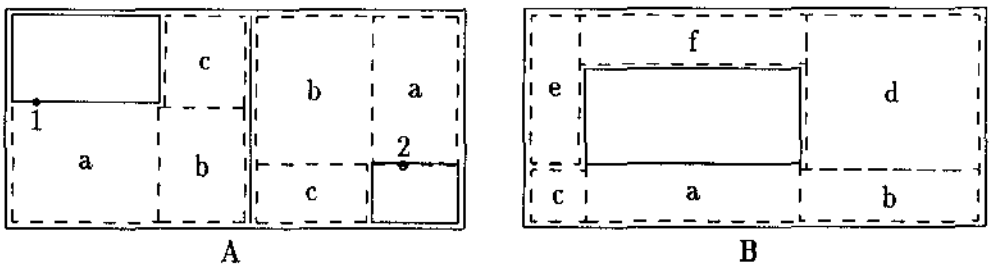


Figura 4.4: Planos de Traçado.

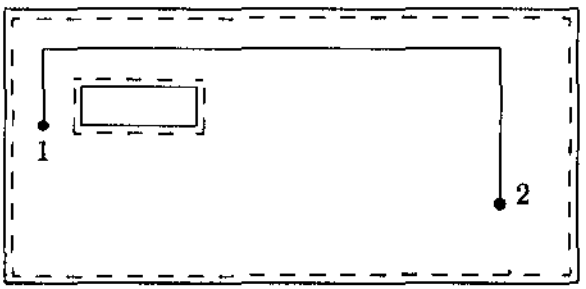


Figura 4.5: Traçado de rotas em mais de um plano (plano-solução).

trechos de rotas pertencentes a planos diferentes é realizada por segmentos de retas perpendiculares aos planos que interligam[3].

É importante notar que o cálculo de interseções não invalida a propriedade de garantia de solução exibida pelo algoritmo básico.

```
{ Algoritmo Generalizado de Expansão de Reta }
algoritmo_generalizado(plano, ponto_origem, ponto_destino);
início
    { No início, o plano-solução é o plano que contém
      os pontos a serem interligados. }
    plano-solução ← plano;
    repita
        solução ← algoritmo_básico(plano_solução,ponto_origem,ponto_destino);
        se ¬ solução então
```

[3] Na construção do circuito impresso os segmentos de retas perpendiculares que interligam os planos são substituídos por furos metalizados.

```
            início
                expansão_de_áreas(plano_solução);
                { Um plano adjacente ao plano-solução é selecionado. }
                plano_selecionado ← seleção_de_plano(plano_solução);
            { A função interseção fornece a união das áreas com interseção
              não nula entre os dois planos. }
                plano_solução ← interseção(plano_solução,plano_selecionado)
            fim
        até que solução ∨ planos esgotados;
        se solução então retrocesso;
    { Determina efetivamente a rota encontrada. }
    fim.
```

4.4 Complexidade do Algoritmo Proposto

Algumas questões a respeito da complexidade do algoritmo devem ser avaliadas. O algoritmo básico de expansão tem sua complexidade determinada pelo número de conexões a serem realizadas. Dados obtidos por [Hey 80] demonstram sua viabilidade para resolver o problema do traçado de rotas restritas a um plano. Quanto aos algoritmos geométricos, poucos algoritmos deste tipo foram analisados até hoje, de modo a se obterem medidas precisas de desempenho. O tempo de execução pode depender de muitas variáveis: da distribuição dos pontos em si, de sua ordenação na entrada, da necessidade ou não do cálculo de funções trigonométricas. Entretanto, existem evidências práticas de que é possível encontrar bons algoritmos para aplicações específicas [Sed 84].

Recentemente, [Wu 87] obteve um algoritmo bom para a determinação de rotas de comprimento mínimo em planos compostos por *polígonos*$_{x-y}$ adjacentes. Este algoritmo pode ser utilizado para melhorar o desempenho do algoritmo proposto, indicando que a abordagem adotada na generalização é promissora.

Capítulo 5

Implementação

A implementação adotada reflete diretamente o exposto no Capítulo anterior. Existe um módulo de implementação para cada uma das fases de traçado, ou auxiliares ao traçado, já descritas.

Para a realização das funções de entrada e saída de dados foram desenvolvidos dois programas:

- Controlador para a Mesa Digitalizadora: é responsável pela captura e formatação dos dados, de acordo com a *linguagem de descrição de dados* definida a seguir. Foi escrito em C e tem aproximadamente 500 linhas de código.
- Controlador para o Traçador de Gráficos ("plotter"): Sua entrada é um arquivo contendo a descrição da placa segundo a mesma *linguagem de descrição de dados*. Encarrega-se de controlar o traçado de cada plano, gerenciando as trocas de penas e possíveis falhas na transmissão dos dados. Foi escrito em Pascal e tem aproximadamente 700 linhas de código.

5.1 Linguagem de Descrição de Dados

A descrição é realizada em FNB[1] com adição de comentários quando considerado necessário.

1. Número de Faces

   <número_faces> ::= NFC = <inteiro> ;
   significado:
   identifica o número
   de faces utilizadas durante
   o traçado.

2. Dimensões da Área de Traçado

   <dimensão> ::= DIM = (<delta_x>,<delta_y>) ;
   significado:
   especifica a máxima área
   disponível para traçado.

[1] Forma Normal de Backus. Em Inglês: Backus Normal Form.

<delta_x> ::= <real>

<delta_y> ::= <real>

3. Passo (Reticulado)

<reticulado> ::= RET = <inteiro> ;
significado:
especifica a distância
a ser mantida entre conexões
paralelas adjacentes.

4. Restrições e Conexões Pré-traçadas

<restrição> ::= RES<tipo_restrição>,
<localização>,
<largura_traço>,
(<coordenada_x_origem>,<coordenada_y_origem>),
(<coordenada_x_destino>,<coordenada_y_destino>) ;

<tipo_restrição> ::= 0 | 1 | 2
significado:
0 = borda de componente
1 = área restrita
2 = conexão pré-traçada

<localização> ::= < inteiro >
significado:
índice identificando em que plano
a restrição deve ser introduzida.

<largura_traço> ::= <inteiro>

<coordenada_x_origem> ::= <coordenada>
<coordenada_y_origem> ::= <coordenada>
<coordenada_x_destino> ::= <coordenada>
<coordenada_y_destino> ::= <coordenada>

<coordenada> ::= <real>

5. Pinos de Componentes

<pino> ::= PIN<número_pino>,
<coroa_metalizada>,
<orientação_coroa>,
(<coordenada_x>,<coordenada_y>) ;

<número_pino> ::= <inteiro>

<coroa_metalizada> ::= <inteiro>
significado:
especifica o tipo de
coroa metalizada
usada no furo metalizado.

<orientação_coroa> ::= <orientação>

<orientação> ::= 0 | 1 | 2 | 3
siginificado:
0 : Norte
1 : Sul
2 : Leste
3 : Oeste

<coordenada_x> ::= <coordenada>
<coordenada_y> ::= <coordenada>

6. Furos Metalizados

<furo_metalizado> ::= FUR<número_furo>,
<coroa_metalizada>,
<orientação_coroa>,
<plano_inferior>,
<plano_superior>,
(<coordenada_x>,<coordenada_y>) ;

<número_furo> ::= <inteiro>

<coroa_metalizada> ::= <inteiro>

<orientação_coroa> ::= <orientação>

<plano_inferior> ::= <inteiro>

<plano_superior> ::= <inteiro>

<coordenada_x> ::= <coordenada>
<coordenada_y> ::= <coordenada>

7. Sinais

<sinal> ::= SIN<número_sinal> =
(<número_pino> | <lista_pinos>),
<largura_traço> ;

<número_sinal> ::= <inteiro>

<número_pino> ::= <inteiro>

<lista_pinos> ::= <número_pino> {, <número_pino> }

8. Inteiros e Reais

<inteiro> ::= <digito>{, <digito> }

<real> ::= <parte_inteira>.<fração>

<parte_inteira> ::= <digito> {, <digito> }

<fração> ::= <digito> {, <digito> }

<digito> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Segue um exemplo de arquivo onde estão descritos os planos de traçado da placa de circuito impresso utilizada como exemplo no Capítulo 4.

início do arquivo

```
NFC = 2;
DIM = (  75.00,  34.30);
RET = 2;
PIN 0, 1, 0, (   5.10,  19.50);
PIN 1, 1, 0, (  65.10,   9.60);
SIN 0 = ( 1 ,0 ), 2;
RES 0, 0, 0, (   1.20,  19.60), (  25.10,  33.30);
RES 0, 0, 0, (  60.00,   0.70), (  74.10,   9.50);
RES 2, 0, 1, (  40.00,   0.80), (  40.00,  33.00);
RES 1, -1, 0, (  94.90,   9.30), ( 129.80,  24.20);
FIM
```

fim do arquivo

O traçador foi programado em módulos independentes para facilitar sua modificação e manutenção, está codificado em Pascal e tem aproximadamente 1800 linhas. Esta modo de programar certamente deixa o traçador mais lento porque aumenta o número de procedimentos e chamadas entre procedimentos, entretanto, acredita-se que uma implementação inicial não deve se preocupar com velocidade, pelo menos não a nível de código objeto gerado.

5.2 Módulo de Entrada

É responsável pela entrada e consistência dos dados, inclui o analisador léxico e sintático para a linguagem de descrição de dados. Ao final de sua execução terá feito as atribuições inicias de dados a cada plano de traçado (Seção 4.3).

5.3 Módulo de Operações sobre Conjuntos

Implementa algoritmos para busca de interseções entre conjuntos[2]. Foi implementado para auxiliar na determinação da conexidade entre componentes de um grafo (Seção 2.2.1).

[2]Em Inglês: union-find.

5.4 Módulo de Filas de Prioridade

É fundamental para a implementação de algoritmos de busca em grafos (Seção 2.2.1). Os procedimentos deste módulo são utilizados também pelo módulo de ordenação.

5.5 Módulo de Ordenação

Implementa um "heap sort" genérico. Utiliza procedimentos implementados no módulo de filas de prioridade para contruir o heap (Seção 2.2.1).

5.6 Módulo de Busca de Árvores Ótimas

Implementa o algoritmo de Kruskal. A implementação segue o algoritmo descrito na Seção 2.3.2 para determinar as árvores de custo mínimo relacionadas a cada sinal a ser traçado.

5.7 Módulo de Ordenação Angular

Implementa o procedimento de ordenação angular descrito na Seção 4.2.

5.8 Módulo de Traçado

Após o término da fase de atribuição de conexões aos planos, realizada pelo módulo de ordenação angular, inicia-se o traçado propriamente dito (Seção 4.3).

5.9 Módulo de Determinação de Interseções

Contrói os hiperplanos de traçado a partir dos planos de traçado originais. O processo de decisão para saber que plano de traçado será incorporado ao hiperplano baseia-se na avaliação da densidade de conexões existente nos planos adjacentes ao hiperplano. Para cada plano, a densidade de conexões é expressa como o inverso do número de conexões realizadas até o momento (Seções 2.2.2 e 4.3.4).

5.10 Módulo de Saída

Percorre as estruturas de dados utilizadas pelo traçador de rotas para representar as informações sobre o traçado, produzindo um arquivo que descreve a placa no momento em que o processo de traçado foi interrompido. A linguagem de descrição de dados proposta é usada para formatar os dados.

Capítulo 6

Conclusão

6.1 Principais Características do Traçador de Rotas

O traçador de rotas tem como principal característica a utilização de conceitos de geometria computacional para obtenção de hiperplanos de traçado (também denominados planos-solução). Os hiperplanos são obtidos da união das interseções não vazias calculadas entre as áreas de expansão geradas durante o processo de expansão realizado nos planos de traçado considerados.

Sua implementação modular permite a substituição de algoritmos. Por exemplo, o algoritmo de Dijkstra pode ser utilizado em lugar do algoritmo de Kruskal para a determinação das árvore ótimas. Para isto, basta conhecer a interface entre este módulo e os demais módulos do traçador. Qualquer módulo pode ser substituido desde que o substituinte obedeça às interfaces definidas. Seu desenvolvimento foi orientado de modo que possa ser facilmente usado em máquinas de pequeno porte.

Sob o ponto de vista do traçado propriamente dito ele apresenta as seguintes características:

- possibilidade de especificação de rotas de largura variável;
- possibilidade de uso de diversos tipos de coroas metalizadas;
- definição de conexões pré-traçadas;
- definição de áreas restritas ao traçado de rotas;
- o traçado final apresenta um pequeno número de furos metalizados;
- os dados de entrada e saída são armazenados do mesmo modo e isto permite que o processo de traçado seja interrompido e reiniciado sem perda do processamento realizado até o momento da interrupção.

Suas principais falhas são:

- a interface homem-traçador não está suficientemente bem desenvolvida;
- conexões são sempre realizadas a 90°. Isto pode levar a um gasto desnecessário do espaço disponível para o traçado. Se as conexões fossem realizadas a 45° algum espaço poderia ser economizado;
- o traçado sob componentes é proibido. Entretanto, é possível alterar facilmente o traçador para que ele permita o traçado sob os componentes.

6.2 Desempenho do Traçador de Rotas

Utilizando um "profiler" serão obtidas medidas a respeito dos trechos mais executados do traçador e isto nos permitirá tecer considerações sobre as opções de implementação adotadas, permitindo indicar possíveis otimizações.

6.3 Sugestão para Futura Pesquisa

Durante o desenvolvimento do traçador de rotas as fases de determinação de árvores ótimas e de cálculo de hiperplanos foram as que mais atrairam minha atenção por apresentarem um apelo claramente geométrico. E é nestas áreas que acredito ser possível melhorar o desempenho do algoritmo proposto.

Em [Pre 85] encontram-se abordagens interessantes para a solução destes problemas. Em particular, acredito que o algoritmo para cálculo de contornos de $polígonos_{x-y}$ descrito naquele texto pode melhorar em muito o desempenho do traçador de rotas.

Outra sugestão é a modificação do módulo de traçado de rotas para que ele seja capaz de tratar o problema do traçado de estruturas regulares de sinais [Lie 82]. Por exemplo, o traçado de sinais de barramentos de memória, onde os diversos sinais, que carregam os endereços e dados, podem ser dispostos de forma regular e em conjunto.

Bibliografia

[Abe 72] Abel, L.C. *On the Ordering of Connections for Automatic Wire Routing, IEEE-Trans. on Computers*, Nov. 72, pp. 1227-33.

[Ake 67] Akers, S. *A Modification of Lee's Path Connection Algorithm,* IEEE Transactions on Electronic Computers, Vol. EC-16(1), Fev 67, pp. 97-98.

[Ara 81] Aranoff, S. & Abulafflo, Y. *Routing on Printed Circuit Boards, 18th IEEE Des. Aut. Conf.*, 1981, pp. 130-136.

[Bel 68] Bellmore, M. & Nemhauser, G.L. *The Traveling-salesman Problem: a Survey, Operations Res.*, Vol. 16, 1968, pp. 538-58.

[Bon 76] Bondy, J.A. & Murty, U.S.R. *Graph Theory with Applications, American Elsevier Publ. Co.*, 1976, pp. 264.

[Bre 72] Breuer, M.A. (editor) *Design Automation of Digital Systems - Theory and Techniques, Prentice-Hall, Inc.*, Englewood Clifs, New Jersey, 1972, pp. 419.

[Bur 83] Burstein, M. & Pelavin, R. *Hierarchical Channel Router, Integration*, North-Holland Publ. Co., Vol. 1(1), Abr. 83, pp. 21-38.

[Buz 88] Buzato, L.E. & Liesenberg, H.K.E. *Um Algoritmo Generalizado de Expansão de Retas com Solução Garantida, Anais do VII Congresso da SBC-XV SEMISH*, Jul. 1988, pp. 164-173.

[Buz 89] Buzato, L.E. & Liesenberg, H.K.E. *A Computational Geometry Based Routing System, Proc. of Int. Conf. on CAD & CG*, Ago. 1989, pp. 503-508.

[CDC 84] Control Data Corporation. *Integrated Computer-Aided Engineering and Manufacturing - TEKROUTE Reference Manual*, 1984, pp. 120.

[Cie 81] Ciesielski, M. J. & Kinnen, E. *An Optimum Layer Assignment for Routing ICs and PCBs, 18th IEEE Des. Aut. Conf.*, 1981, pp. 733-737.

[Coo 67] Coombs, Jr., C.F. *Printed Circuits Handbook, McGraw-Hill Book Company*, New York, 1967, pp. 539.

[Deu 76] Deutsch, D. N. *A "Dogleg" Channel Router, 13th ACM-IEEE Des. Aut. Conf.*, 1976, pp. 425-533.

[Dij 59] Dijkstra, E.W. *A Note on Two Problems in Connexion with Graphs, Numerische Mathematik*, Vol. 1, 1959, pp. 269-271.

[Don 80] Donath, W.E. *Complexity Theory and Design Automation, Comm. of the ACM*, 1980, pp. 412-419.

[Fen 73] Fencl, Z. *Routing Problem, Comm. of The ACM*, Vol. 16(9), Sep. 73, pp. 572-574.

[Gar 77] Garey, M.R. & Johnson, D.S. *The Rectilinear Steiner Tree Problem is NP-complete, SIAM J. Appl. Math.*, Vol. 32(4), Jun. 77, pp. 826-859.

[Gin 69] Ginsberg, G. et al *An Updated Multilayer Printed Wiring CAD Capability, Proc. 6th Des. Aut. Workshop*, 1969, pp. 1-29.

[Gil 68] Gilbert, E.N. & Pollak, H.O. *Steiner Minimal Trees, SIAM J. Appl. Math.*, Vol. 16(1), 1968, pp. 1-20.

[Han 66] Hanan, M. *On Steiner's Problem with Rectilinear Distance, J. SIAM Appl. Math.*, Vol. 14(2), Mar. 66, pp. 255-265.

[Has 71] Hashimoto, T. & Stevens, T. *Wire-Routing by Optimizing Channel-Assignment within Large Apertures, 8th Design Automation Workshop*, Jun. 1971, pp. 155-169.

[Hey 80] Heyns, W. Sansen, W. & Beke, H. *A Line-Expansion Algorithm for the General Routing Problem with a Guaranteed Solution, 17th Design Automation Conference*, Jun 80, pp. 243-249.

[Hel 70] Held, M. & Karp, R. M. *The Traveling-salesman Problem and Minimum Spannig Trees, Operations Res.*, Vol. 18, 1970, pp. 1138-62.

[Hel 71] Held, M. & Karp, R. M. *The Traveling-salesman Problem and Minimum Spannig Trees: part II, Math. Programming*, Vol. 1, 1971, pp. 6-25.

[Hig 69] Hightower, D.W. *A Solution to Line-routing Problems on the Continuous Plane, 6th Des. Aut. Workshop*, Jun. 69, pp. 1-23.

[Hig 74] Hightower, D.W. *The Interconnection Problem: a Tutorial, IEEE Computer*, Vol. 7(4), Abr. 74, pp. 18-32.

[Hig 80] Hightower, D.W. & Boyd, R.L. *A Generalized Channel Router, 17th ACM-IEEE Des. Aut. Conf.*, Jun. 80, pp. 12-21.

[Hig 83] Hightower, D. *The Lee Router Revisited, IEEE Int. Conf. of Computer Design VLSI*, Nov. 83, pp. 136-139.

[Hwa 76] Hwang, F.K. *On Steiner Minimal Trees with Rectilinear Distance, SIAM J. Appl. Math.*, Vol. 30(1), 1976, pp. 104-114.

[Int 86a] Intergraph. *Tiger Router Manual for Beta Sites, Intergraph Co.*, Nov. 86, pp. 26.

[Int 86b] Intergraph. *Tiger Router — Pre-certification Guide, Intergraph Co.*, Dez. 86, pp. 25.

[Int 86c] Intergraph. *Hardware Accelerated Routing Environment Users Guide, Intergraph Co.*, Jan. 86, pp. 35.

[Int 84a] Intergraph. *Vax Interactive Eletronic Design Software (IEDS) Application Guide, Intergraph Co.*, Mai. 84, pp. 96.

[Int 84b] Intergraph. *Vax IEDS User's Guide (8.7), Intergraph Co.*, Nov. 84, pp. 301.

[Kor 72] Koren, N. *Pin Assignment in Automated Printed Circuit Board Design, Proc. 9th Des. Aut. Workshop*, 1972, pp. 72-79.

[Kor 82] Korn, R. K. *An Efficient Variable-cost Maze Router, 19th ACM-IEEE Des. Aut. Conf.*, Jun. 82, pp. 425-431.

[Kru 56] Kruskal, J.B. *On the Shortest Spanning Subtree of a Graph and The Traveling Salesman Problem, Proceedings of AMS*, Vol. 7(1), 1956, pp. 48-50.

[Las 69] Lass, S.E. *Automated Printed Circuit Routing with Stepping Aperture Comm. of the ACM*, Vol. 12(5), Mai. 69, pp. 262-265.

[Lee 61] Lee, C. Y. *An Algorithm for Path Connections and Its Appllications, IRE Trans. on Elec. Comp.*, Vol. EC-10(3), Sep. 61, pp. 346-365.

[Lie 80] Liesenberg, H.K.E. *Traçado Completo de Rotas em Placas de Dimensões não Pré fixadas, UNICAMP - tese de mestrado*, Out. 80, pp. 144.

[Lie 86] Liesenberg, H.K.E. *Projetos VLSI e seu Suporte Computacional, V Escola de Computação*, Ed. Gráfica Formato S.A., 1986, pp. 146.

[Lie 82] Lie, M. & Horng, C. *A Bus Router for IC Layout, 19th ACM-IEEE Des. Aut. Workshop*, Jun. 82, pp. 129-132.

[Lin 79] Lindsey, D. *The Design and Drafting of Printed Circuits, Bishop Graphics, Inc.*, 1979, pp. 196.

[Lob 57] Loberman, H. & Weinberger, A. *Formal Procedures for Connecting Terminals with a Minimum Total Wire Length, Journal of ACM*, Vol. 4, Oct. 57, pp. 428-437.

[Luc 79] Lucchesi, C.L., Simon, I., Simon, I., Simon, J. & Kowaltowski, T. *Aspectos Teóricos da Computação Impa - Cnpq (Projeto Euclides)*, 1979, pp. 292.

[Oes 72] Oestreicher, D. *Automatic Printed Circuit Board Design, University of Utah Computer Science Division, UTEC-CSc*, Jun 1972, pp.72-119.

[Pin 82] Pinter, R. Y. *Optimal Layer Assignment for Interconnect, IEEE Int. Conf. on Circuits and Computers*, Oct. 82, pp. 398-401.

[Pre 85] Preparata, F.P. & Shamos, I.M. *Computational Geometry - an Introduction, Springer-Verlag*, New York, 1985, pp. 390.

[Pri 57] Prim, R.C. *Shortest Connection Networks and some Generalizations, Bell System Tech. Journal*, Vol. 36, 1957.

[Sah 80] Sahni, S. & Bhatt, A. *The Complexity of Design Automation Problems, ACM IEEE 17th Design Automation Conference*, 1980, pp. 402-411.

[Sed 84] Sedgewick, R. *Algorithms. Addison-Wesley Publishing Company*, 1984, pp. 551.

[Sha 76] Shamos, M.I. & Hoey, D. *Geometric Intersection Problems, Seventeenth Annual IEEE Symposium on Foundations of Computer Science*, Out. 1976, pp. 208-215.

[Sut 69] Sutherland, I.E. *A Method for Solving Arbitrary-Wall Mazes by Computer, IEEE-Trans. on Comp.*, Vol. C-18(12), Dez. 69, pp. 1092-1097.

[Tad 80] Tada, F. & Kagata, T. & Yoshimura, K. & Shirakawa, T. *A Fast Maze Router with Iterative use of Variable Search Space Restriction, 17th IEEE Des. Aut. Conf.*, Jun. 80, pp. 250-254.

[Wu 87] Wu, Y., Widmayer, P., Schlag, M.D. & Wong, C.K. *Rectilinear Shortest Paths and Minimum Spanning Trees in the Presence of Rectilinear Obstacles, IEEE Transactions on Computers*, Vol. C-36(3), Mar 87, pp. 321-331.