UM ALGORITMO EFICIENTE DE ROTEAMENTO GLOBAL PARA FERRAMENTAS CAD

STÈPHANO GONÇALVES; LEOMAR DA ROSA JR; FELIPE MARQUES

Universidade Federal de Pelotas – {smmgoncalves, leomarjr, felipem}@inf.ufpel.edu.br

1. INTRODUÇÃO

Com o desenvolvimento das tecnologias, os circuitos integrados passaram a se tornar cada vez mais complexos, demandando o uso de métodos automatizados para sua concepção. Com isso, foram desenvolvidas ferramentas CAD (*Computer Aided Design*), cujo principal objetivo é a automação do processo de síntese lógica de circuitos. No entanto, elas também podem oferecer suporte à visualização e manipulação de circuitos. Esta funcionalidade traz diversos benefícios ao desenvolvedor, pois possibilita uma maior compreensão do funcionamento do circuito, facilita a detecção de erros lógicos, e pode ser usada para simular o processo de síntese física do circuito.

Devido ao grande número de componentes e de conexões em um circuito, é necessário que existam algoritmos que organizem a disposição dos elementos do circuito. Os algoritmos de posicionamento são responsáveis por determinar as posições das células de forma que o comprimento total de fio do circuito seja minimizado. Uma vez que os componentes estejam posicionados, é necessário fazer as ligações entre esses componentes, por meio de algoritmos de roteamento. O roteamento visa encontrar um caminho entre dois ou mais componentes, desviando de obstáculos e buscando minimizar o comprimento do caminho. Uma abordagem clássica para o problema do roteamento é a utilização do algoritmo A* (HART, 1968).

Como a visualização de circuitos em ferramentas CAD oferece a possibilidade de manipulação dos componentes do circuito, o cálculo de roteamento deve ser feito cada vez que os componentes selecionados são movidos na tela, o que pode ser um processo lento, dependendo do tamanho do circuito e do número de elementos selecionados. Com isso, é necessário que o roteamento utilize algoritmos que minimizem o tempo de processamento.

Este trabalho propõe um algoritmo de roteamento com o objetivo de otimizar o tempo de processamento. O método guarda informação de segmentose realiza cálculos de intersecção de segmentos para impedir a sobreposição dos fios, diferentemente da abordagem utilizada pelo algoritmo A*, que guarda a informação de cada ponto traçado em uma grade virtual. O trabalho apresenta o algoritmo proposto e mostra uma análise comparativa do tempo de processamento utilizado pelo método proposto e pelo algoritmo A*.

2. METODOLOGIA

O algoritmo proposto realiza a conexão entre dois pontos por meio de segmentos de reta ortogonais entre si. Os segmentos são definidos pelos dois valores que representam o intervalo em um determinado eixo (x ou y), um valor que representa a posição do segmento no outro eixo, e o identificador do eixo. O algoritmo possui uma abordagem de pesquisa por profundidade, baseada na técnica de *backtracking* (EITAN, 1999). Para impedir a sobreposição de fios com elementos do circuito, são utilizadas duas estruturas de dados que representam a

grade de posições ocupadas, sendo uma para o eixo x e outra para o eixo y. Essas posições são descritas através de segmentos, ao invés de pontos, como na abordagem do algoritmo A*.

O algoritmo proposto parte de um ponto inicial e tenta traçar uma série de segmentos ortogonais até o destino. O ponto de destino de um segmento é o ponto de origem do próximo. Quando o algoritmo encontra alguma intersecção, é necessário descobrir de que forma o segmento foi interceptado. Um segmento pode ser interceptado pelo seu ponto de origem ou pelo ponto de destino. Quando um segmento é interceptado pelo ponto de destino, um novo ponto de destino é gerado. Quando é interceptado pelo ponto de origem, o segmento não pode ser criado na determinada linha. Neste caso, o algoritmo descarta o segmento atual e calcula um novo ponto de destino no segmento anterior, visando prosseguir a busca através desse ponto. A figura 1 mostra o resultado final do procedimento de criação de segmentos em um caso com duas intersecções. O objetivo é traçar um fio partindo do ponto de origem O ao ponto de destino D. Primeiramente, o algoritmo criará o segmento [O, P1]. Após, será criado o segmento [P1, P2]. Porém, este segmento é interceptado em P1 (ponto de origem do segmento) por outro segmento. Com isso, o segundo segmento não poderá ser criado nesta origem. O algoritmo então corrige o primeiro segmento criado para [O, P3]. Em seguida, o segmento [P3, P4] é criado, mas este intercepta um componente do circuito pela sua extremidade de destino. O segmento pode ser criado nesta origem mas deve então ser corrigido para [P3, P5]. Após, os segmentos [P5, P6], [P6, P7] e [P7, D] são criados sem intersecções.

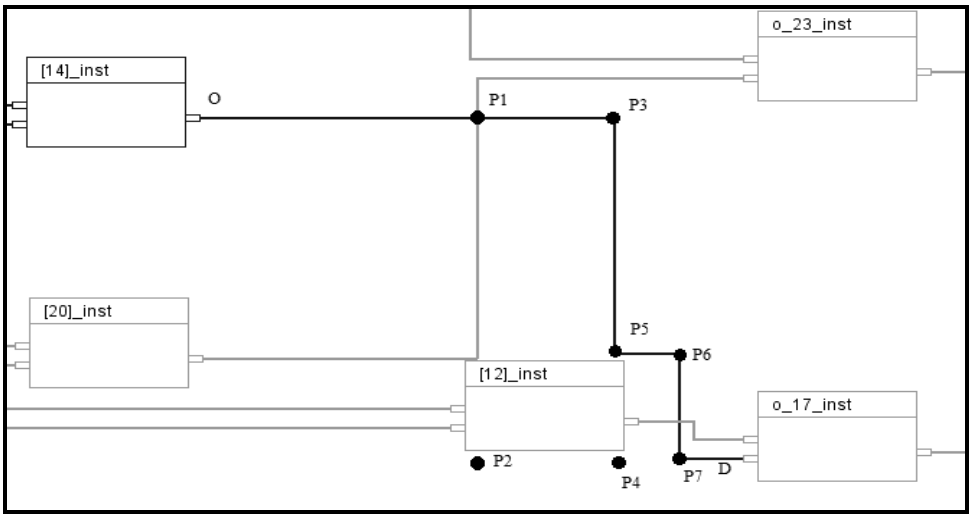

Figura 1. Fio resultante após enfrentar duas intersecções

Quando um segmento intercepta um componente, é necessário verificar se o componente bloqueia o caminho ótimo entre o ponto inicial e o destino. Se o caminho é bloqueado é necessário que o fio contorne o componente para prosseguir. Quando mais de um componente bloqueia o caminho ótimo, o algoritmo não garante que o menor caminho seja encontrado. Caso contrário, o caminho ótimo é sempre encontrado.

3. RESULTADOS E DISCUSSÃO

O algoritmo proposto e o algoritmo A* foram implementados na linguagem de programação JAVA (2013). Os algoritmos foram integrados a uma ferramenta de visualização de circuitos implementada em nosso grupo de pesquisa. Os

experimentos foram realizados em uma máquina com um processador Core 2 Duo E8500 3.16 Ghz e com 4 Gb de memória RAM. Ambos algoritmos de roteamento foram executados para o roteamento de fios de 5 circuitos do conjunto de *benchmarks* MCNC (LISANKE, 1988). Para cada benchmark o processo de roteamento foi executado 10 vezes. Dessa forma, foi calculado o tempo médio de criação de caminhos, medido em nanosegundos.

A tabela 1 apresenta o tempo médio de criação de fios para cada circuito. A última coluna representa o percentual de melhoria do algoritmo proposto em relação ao algoritmo A*. O algoritmo proposto mostrou-se superior em todos os casos. Para interpretar os resultados apresentados pela tabela 1, é necessário tomar conhecimento de algumas características dos circuitos utilizados. O número de componentes, o número de fios e o tamanho dos fios são fatores que influenciam no desempenho do roteamento.

Tabela 1. Tempo médio de roteamento entre o algoritmo proposto e o A* e comprimento médio de fio para cada circuito

circuitos	Comprimento (estimado) médio de fio (pixels)	Tempo médio (nanosegundos)		Percentual de Melhoria(%)
		Algoritmo Proposto	A*	
cm163a	2395	21.478	5.924.826	27.585,56
c17	900	33.145	708.494	2.137,56
cm152a	1465	33.236	1.882.834	5.665,04
cm82a	1011	39.185	861.004	2.197,28
i2	23091	172.936	1.256.508.736	726.574,42

Analisando a tabela 1, pode-se observar que, para o algoritmo A*, o comprimento médio de fio teve um grande impacto no desempenho. Como este algoritmo expande sua busca de ponto a ponto, é esperado que, quanto maior a distância entre a origem e o destino, maior será o número de expansões e, consequentemente, maior será o custo computacional. O algoritmo também é influenciado pelo número componentes, pois estes formam obstáculos na busca pelo caminho. Ao encontrar obstáculos, o algoritmo precisa expandir a busca em todas as direções até que os obstáculos sejam superados, o que acarreta em um aumento no número de expansões.

O algoritmo proposto não é afetado pelo tamanho médio de fio. Apenas no circuito i2 o desempenho do algoritmo proposto foi consideravelmente inferior em relação aos outros circuitos. Porém, isto se deve ao elevado número de componentes e de fios, e não ao tamanho médio de fio. A complexidade do algoritmo proposto é proporcional ao número de segmentos criados, e não ao tamanho dos segmentos. Dessa forma, um circuito com um número elevado de componentes e fios apresentará um grande número de obstáculos no roteamento, acarretando em um maior número de intersecções e correções de segmentos. Com isso, grandes circuitos exigirão um grande número de tentativas de criação de segmentos, até que o caminho seja encontrado.

No que diz respeito ao comprimento do caminho, o algoritmo A* é superior ao algoritmo proposto, pois sempre encontra o menor caminho. Em geral, o algoritmo proposto também encontra o menor caminho, no entanto, quando o caminho ótimo é bloqueado por mais de um obstáculo, o algoritmo não garante que o menor caminho seja encontrado.

Para a visualização de circuitos, o comprimento de fio não é um fator prioritário, apesar de ser desejável. A prioridade é o fator tempo, pois é desejável visualizar e manipular o circuito sem esperar o processo de roteamento ser realizado. Com isso, o algoritmo proposto representa uma melhor escolha para o roteamento em um visualizador de circuitos.

4. CONCLUSÕES

Este trabalho apresentou a proposta de um algoritmo de roteamento para ferramentas CAD. O algoritmo possui uma abordagem de busca por profundidade. A busca é feita através da criação de segmentos ao invés da tradicional busca feita de ponto em ponto.

O algoritmo proposto foi comparado com o algoritmo A*. Ambos foram implementados e executados em uma ferramenta de visualização de circuitos. O tempo médio de criação dos fios foi computado para ambos algoritmos. Em todos os casos o algoritmo proposto mostrou-se superior com taxas de melhora de desempenho que variaram de 2.137% à 726.574%. De acordo com os resultados experimentais, pode-se concluir que o fator que torna o algoritmo proposto superior, em termos de desempenho, em relação ao algoritmo A* é a utilização de segmentos para formar o caminho. Com essa abordagem, a distancia entre os pontos de origem e de destino não influencia no desempenho do algoritmo. Em contrapartida, o algoritmo A* garante que o menor caminho seja sempre encontrado, o que não é válido para o algoritmo proposto. No entanto, considerando que as ferramentas com suporte a visualização de circuitos, no que diz respeito ao roteamento, possuem a prioridade de minimizar o tempo de processamento, o algoritmo proposto representa uma melhor escolha para executar o roteamento.

5. REFERÊNCIAS BIBLIOGRÁFICAS

HART, P. E.; NILSSON, N. J.; RAPHAEL, B. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. **Transactions on Systems Science and Cybernetics**, p. 100–107, 1968.

EITAN, G. **Backtracking Algorithms**. CIS 680: Data Structures, 1999. Acessado em 3 out. 2013. Online. Disponível em: http://www.cse.ohio-state.edu/~gurari/course/cis680/cis680Ch19.html.

Java: Make the Future Java, 2013. Acessado em 3 out. 2013. Disponível em: http://www.oracle.com/us/technologies/java/ overview/index.html.

LISANKE , R. Logic synthesis and optimization benchmarks. **Microelectron. Center North Carolina**, Research Triangle Park. 1988