UNIVERSIDADE FEDERAL DA PARAÍBA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

GUSTAVO REZENDE CARVALHO

Heurísticas para a Fase de Roteamento de Circuitos Integrados Baseados em FPGAs

JOÃO PESSOA-PB
Março-2009

GUSTAVO REZENDE CARVALHO

Heurísticas para a Fase de Roteameneto de Circuitos Integrados Baseados em FPGAs

DISSERTAÇÃO APRESENTADA AO CENTRO DE CIÊNCIAS EXATAS E DA NATUREZA DA UNIVERSIDADE FEDERAL DA PARAÍBA, COMO REQUISITO PARCIAL PARA OBTENÇÃO DO TÍTULO DE MESTRE EM INFORMÁTICA (SISTEMAS DE COMPUTAÇÃO).

Orientador: Antônio Carlos Cavalcanti
Orientador: Lucídio dos Anjos Formiga Cabral

JOÃO PESSOA-PB
Março-2009

C331h Carvalho, Gustavo Rezende.
Heurísticas para a fase de roteameneto de circuitos integrados baseados em FPGAs / Gustavo Rezende Carvalho. -- João Pessoa: [s.n.], 2009.
89f. :il.
Orientador: Antônio Carlos Cavalcanti.
Co-Orientador: Lucídio dos Anjos Formiga Cabral.
Dissertação (Mestrado) – UFPb – CCEN.

1.Informática. 2. Roteamento. 3.VPR. 4.GRASP. 5. ILS.

UFPb/BC CDU:(043)

Ata da Sessão Pública de Defesa de Dissertação de Mestrado de **Gustavo Resende de Carvalho**, candidato ao Título de Mestre em Informática na Área de Sistemas de Computação, realizada em 05 de março de 2010.

Aos cinco dias do mês de março do ano dois mil e dez, às 14 horas e 30 minutos, no Auditório do Centro de Ciências Exatas e da Natureza da Universidade Federal da Paraíba, reuniram-se os membros da Banca Examinadora constituída para examinar o candidato ao grau de Mestre em Informática, na área de "*Sistemas de Computação*", na linha de pesquisa "*Processamento de Sinais e Sistemas Gráficos*", o Sr. Gustavo Resende de Carvalho. A comissão examinadora foi composta pelos professores doutores: Lucidio dos Anjos Formiga Cabral (DI-UFPB), Primeiro Orientador e Presidente da Banca Examinadora, Antonio Carlos Cavalcanti (DI-UFPB), Segundo Orientador, José Antonio Gomes de Lima (DI-UFPB), examinador interno e Rômulo Pires Coelho Ferreira (Instituto Federal de Educação, Ciência e Tecnologia de Alagoas-IFAL), examinador externo. Dando início aos trabalhos, o professor Lucidio dos Anjos Formiga Cabral, cumprimentou os presentes, comunicou aos mesmos a finalidade da reunião e passou a palavra ao candidato para que o mesmo fizesse, oralmente, a exposição do trabalho de dissertação intitulado "*Heurísticas para a Fase de Roteamento de Circuitos Integrados Baseados em FPGA*" . Concluída a exposição, o candidato foi argüido pela Banca Examinadora que emitiu o seguinte parecer: "*aprovado*". Assim sendo, deve a Universidade Federal da Paraíba expedir o respectivo diploma de Mestre em Informática na forma da lei e, para constar, eu, professora Tatiana Aires Tavares, Coordenadora deste Programa, servindo de secretária lavrei a presente ata que vai assinada por mim mesmo e pelos membros da Banca Examinadora. João Pessoa, 05 de março de 2010.

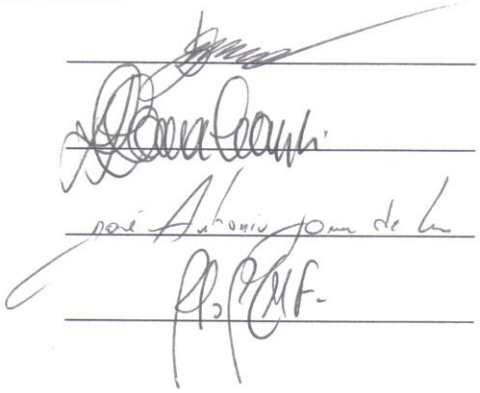
Tatiana Aires Tavares

Prof. Dr. Lucidio dos Anjos Formiga Cabral
Primeiro Orientador (DI-UFPB)

Prof. Dr. Antonio Carlos Cavalcanti
Segundo Orientador (DI-UFPB)

Prof. Dr. José Antonio Gomes de Lima
Examinador Interno (DI-UFPB)

Prof. Dr. Rômulo Pires Coelho Ferreira
Examinador Externo (IFAL)



Agradecimentos

Ao Prof. Dr. Antônio Carlos Cavalcanti, pela atenção e apoio durante o processo de definição e orientação.

Ao Prof. Dr. Lucídio dos Anjos Formiga Cabral, pela atenção e apoio durante o processo de definição e orientação.

Aos meus pais, Prof. Leandro Baptista Carvalho Filho e Profa. Vania Rezende Carvalho, pela atenção e apoio durante o processo de definição e orientação.

Ao Programa de Pós-Graduação Graduação em Informática da Universidade Federal da Paraíba, pela oportunidade de realização do curso de mestrado.

Ao Conselho Nacional de Desenvolvimento Científico e Tecnológico, pela concessão da bolsa de mestrado e pelo apoio financeiro para realização desta pesquisa através do Programa Nacional de Microeletrônica.

Ao Doutorando Anand Subramanian, pela atenção e apoio durante o processo de orientação.

Ao Prof. Gilberto Farias de Souza Filho, pela atenção e apoio durante o processo de orientação.

Aos amigos, pelo apoio durante a realização do curso de mestrado.

RESUMO

A presente dissertação trata do problema de roteamento de circuitos para Field Programable Gate Arrays (FPGAs). Em função da natureza combinatória do problema, métodos heurísticos são comumente utilizados para gerar soluções de boa qualidade em um tempo computacionalmente aceitável. Neste contexto, um procedimento baseado na metaheurística GRASP (*Greedy Randomized Adaptive Search Procedure*) com um procedimento de busca local baseado em ILS (*Iterated Local Search)* é proposto. O algoritmo foi testado em instâncias encontradas na literatura, *benchmark* MCNC, explorando os critérios de tempo crítico e números de trilhas, onde mostrou-se capaz de melhorar 55% das instancias no critério de tempo crítico e 5,3% quanto ao número de trilhas.

Palavras chave: Roteamento, FPGA, VPR, GRASP, ILS.

Abstract

The present dissertation deals with the Routing Circuits for Field Programable Gate Arrays (FPGAs). Due to the combinatorial nature of the problem, heuristics methods are commonly used to generate good quality solutions in an acceptable computationally time. In this context, a procedure based on GRASP (Greedy Randomized Adaptive Search Procedure) with a procedure of local search based on ILS (Iterated Local Search) is proposed. The algorithm has been tested in benchmark problems found in the literature, MCNC, exploring timing-driven and channel-width criteria, being able to improve 55% of the benchmarks on timing drive criteria and improve 5,3% of the benchmarks on channel width criteria.

Key words: Routing, FPGA, VPR, GRASP, ILS.

Sumário

Lista de Figuras

Lista de Tabelas

Introdução

Com o surgimento em 1985 da tecnologia *Field-Programmable Gate Arrays* (FPGA), criada pelos co-fundadores da Xilinx, Ross Freeman e Bernard Vonderschmitt, o paradigma de hardware programável é criado. Projetada para competir com a *Complex Programmable Logic Device* (CPLDs), suas principais vantagens sobre suas concorrentes é ser reprogramavel através de uma linguagem de descrição de hardware (*hardware description language* – HDL) ou um design esquemático do circuito e esta programação não é permanente na placa. Isto se torna possível pela utilização, em sua maioria, de tecnologia *Synchronous* RAM (SRAM) para configurar os blocos lógicos.

Para o primeiro FPGA comercial, a XC2064 da Xilinx, com apenas 64 blocos lógicos e LUTs de 3 entradas, não era uma tarefa difícil definir manualmente a programação do FPGA para implementar o circuito desejado, mas nem por isso deixava de ser trabalhoso. Para programar um FPGA, no mínimo é necessário ter o circuito desejado em mãos, mapeado logicamente, posicionar os pinos I/O do circuito e as funções lógicas nos blocos lógicos do FPGA e roteá-la, ou seja, configurar os *switchs* presentes no FPGA para definir as ligações do circuito.

Tendo em vista que automatizando este processo uma grande quantidade de tempo poderia ser poupada, conseqüentemente, ferramentas de CAD para FPGAs foram criadas. Um sistema de projeto típico deverá incluir suporte para os seguintes passos: entrada do projeto inicial, otimização lógica, simulação, mapeamento lógico, posicionamento e roteamento. Essas etapas interferem diretamente no desempenho do circuito implementado no FPGA e o tempo gasto em cada uma destas etapas é determinante no *time to market* dos projetos.

Sendo as trilhas o número de segmentos de fios de cada canal do FPGA, em 1997, Vaughn Betz propõe um desafio chamado de "*FPGA Place and Route Challenge*" (Desafio de posicionamento e roteamento de FPGA), no qual pagaria 1,00 US$ para cada trilha que utilizassem a menos que o programa VPR, para posicionar e rotear os circuitos da MCNC. Em função da natureza combinatória do problema de posicionar e rotear, NP-Difícil, podendo ser provado através de coloração de grafos, vários trabalhos foram publicados propondo métodos

heurísticos para gerar soluções de boa qualidade em um tempo computacional aceitável.

A principal contribuição deste trabalho reside em propor um algoritmo de roteamento baseada nas metaheurísticas GRASP e ILS com o objetivo primordial de diminuir o tempo do caminho crítico e a quantidade de trilhas para rotear a FPGA. Outra questão abordada neste trabalho foi quanto ao *rip-up*, é que a seleção das *nets* a serem “ripadas” ou re-roteadas não foi bem explorada, pois a ordem em que as *nets* são roteadas interfere diretamente nos resultados e este estudo foi deixado em aberto por Keith So (SO, 2007).

O trabalho está organizado da forma que se segue: o capítulo 1 apresenta os objetivos gerais e específicos da proposta; o capítulo 2 descreve os fundamentos da FPGA, assim como arquiteturas e suas peculiaridades; o capítulo 3 faz uma revisão dos trabalhos com os melhores resultados atingidos na literatura nos últimos anos; capítulo 4 trata do conjunto de ferramentas VPR 5.0; capítulo 5 demonstra o algoritmo proposto; capítulo 6 encontra-se os resultados obtidos; e no capítulo 7 as conclusões.

Capítulo 1 – Delimitação do Objeto

O presente trabalho trata sobre o estudo do processo automático de roteamento em circuitos baseados em FPGAs, com enfoque no melhoramento do algoritmo de roteamento em relação a área do circuito.

1.1 Objetivo Geral

Implementar um algoritmo de roteamento de *Field Programable Gate Arrays* (FPGAs) que produza melhorias no desempenho do circuito através da redução do tamanho do caminho crítico e possivel redução da área utilizada. Ou seja, reduzir o número o tamanho das *nets* que compõem o caminho crítico e reduzir as trilhas necessárias para rotear um circuito em um FPGA a partir do levantamento dos algoritmos presentes na literatura.

1.2 Objetivos específicos

Levantar o estudo da arte sobre os processos de roteamento FPGAs, arquiteturas e instâncias utilizadas, na perspectiva de efetuar o estudo dos algoritmos e seus resultados, observando vantagens e desvantagens, tal qual a modelagem necessária para cada tipo de arquitetura e instâncias necessárias para os testes.

Analisar as estruturas de dados e rotinas internas da ferramenta VPR 5.0 relativas ao roteamento de circuitos a fim de:

- Aproveitar o mecanismo de posicionamento presente na ferramenta.
- Utilizar os mecanismos de verificação presentes na ferramenta.
- Realizar testes comparativos com o VPR 5.0 capazes de comprovar a eficácia do algoritmo, em estudo, através dos *benchmarks* disponíveis.

Desenvolver um algoritmo baseado na metaheurística GRASP com a fase de busca local baseado na metaheurística ILS.

Estudar a influência do critério do ordenamento das *nets* a serem roteadas com o objetivo de identificar critérios que possibilitem um melhor roteamento através de testes.

Capítulo 2 – Fundamentos do FPGA

Para melhor compreensão da modelagem do roteamento de FPGAs faz-se necessário o detalhamento de suas arquiteturas e peculiaridades descritas a partir de estudos realizados.

2.1 Arquiteturas de FPGAs

Todos os FPGAs são compostos de três componentes fundamentais: blocos lógicos (L), blocos de entrada/saída (E/S), e roteamento programável. Um circuito é implementado em um FPGA programando-se cada bloco lógico para gerar uma partes da lógica requerida pelo sistema, cada bloco (E/S) para agir como um pino (*pad*) de entrada ou de saída e o roteamento programável é configurado para fazer todas as conexões necessárias entre blocos lógicos e blocos E/S (CABRAL,2001).

2.1.1 Arquitetura de FPGA 2-D Array

A arquitetura FPGA 2-D Array (*island style)* consiste em uma matriz bidimensional de células lógicas e recursos de roteamento, na qual cada bloco (célula lógica) é marcado com L e pode ser configurado para ser uma *look-up-table* (LUT), um *flip-flop*, uma memória, etc. Entre os blocos lógicos existem canais verticais e horizontais onde passam segmentos de fios com a funcionalidade de conectar pinos de blocos, estes alinhados em trilhas (*tracks*). Portanto, um canal vertical ou horizontal é um conjunto de trilhas entre duas colunas ou linhas (no caso do canal vertical) consecutivas de células. Como se pode observar na Figura 1, existem quatro trilhas por canal e cada bloco lógico tem quatro pinos sobre cada um dos seus lados.

Como segmentos de fios são pré-fabricados, a personalização dos recursos de roteamento é alcançada através da programação adequada de *switches*. Estes últimos são agrupados dentro de blocos *switches* (S) e de conexões (C) (Figuras 1(a) e 1(b)). Os blocos C contêm *switches* de roteamento que podem

ser programados para conectar pinos de células lógicas a segmentos de fio. Os pontos terminais de segmentos de fio estão dentro dos limites de blocos S e estes contêm *switches* que permitem a conexão de um segmento a outro (CABRAL, 2001). A arquitetura 2-D *array* utiliza apenas segmentos de fios de tamanho simples (ou seja, todo segmento de fio cobre somente um bloco C, ou ainda, que seus pontos extremos estejam em blocos S vizinhos).

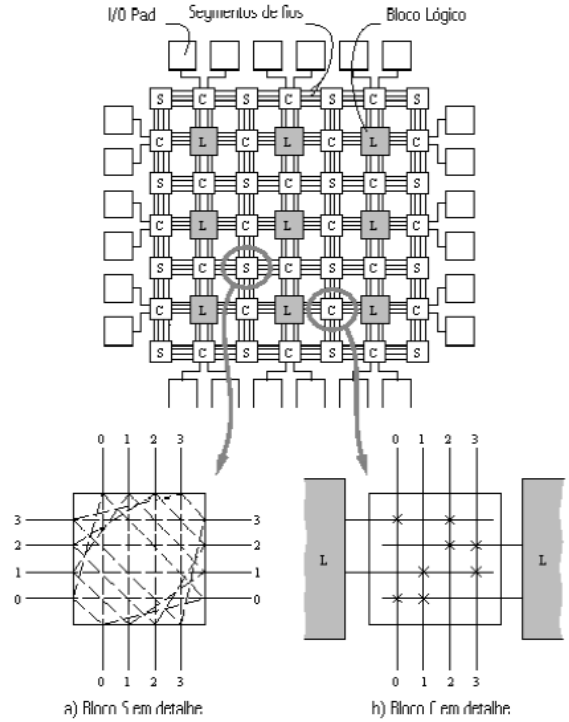


Figura 1: Modelo Geral de uma FPGA baseada em Array

2.1.2 Arquitetura de FPGA 3-D

A topologia da arquitetura 3-D para FPGA é análoga a um FPGA 2-D *Array* com várias camadas sobrepostas e interligadas por vias no terceiro eixo da dimensão, em conseqüência, passa-se a ter mais possibilidades de alocação e roteamento de circuitos neste tipo de topologia, conforme pode ser observado na Figura 2.

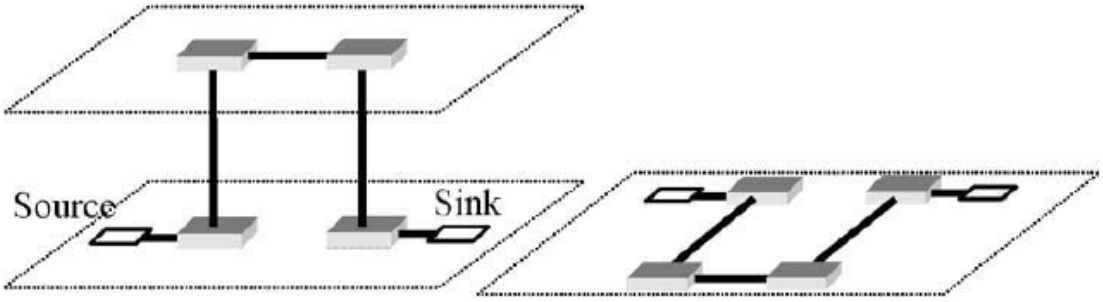


Figura 2: Opções de posicionamento e roteamento em uma FPGA 3-D.

Um maior número de possibilidades necessita de um maior número de transistores para implementação das conexões acarretando no aumento do consumo de energia. Outro problema é a seleção de um conector apropriado entre as camadas do dispositivo 3-D. Além disso, um número grande de vias ocupa uma grande porção de área SI, fazendo com que circuitos ativos e interconexões devam ser excluídos. Em conseqüência, o efeito da distribuição e extensão dessas vias sobre o desempenho e consumo de energia de FPGAs 3D precisa ser focada (SIOZIOS, 2007).

Fazendo um comparativo entre um *switch box 2-D com um switch box 3-D* (Figura 3), observa-se que, com a mesma quantidade de trilhas, neste caso três, o *switch box 2-D* necessita ter a opção de conectar a três trilhas contra cinco da *switch box 3-D* (SIOZIOS, 2007).

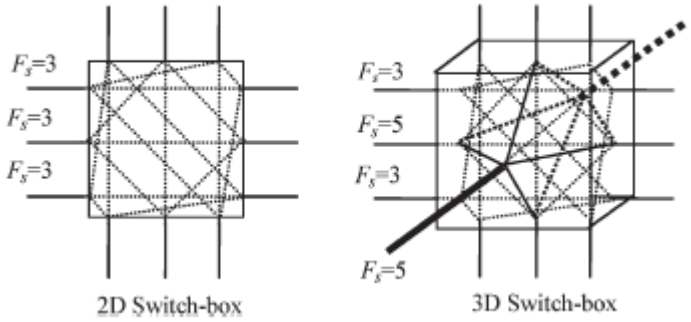


Figura 3: Opções de conexão de uma *Switch Box* 2D e 3D.

2.1.3 Arquitetura de FPGA Heterogênia

Nos FPGAs atuais é comum ter um conjunto heterogêneo de blocos lógicos. Além do básico *soft logic cluster,* os blocos adicionais são normalmente

circuitos de estruturas complexas que são projetados para realizar uma operação específica e, por esta razão, esses blocos heterogêneos são comumente chamados de *hard blocks*. Os blocos diferenciados poderiam ser também um tipo diferente de *soft logic cluster* que fornece melhor desempenho ou economia de espaço (CONG, 1998 e HE,1993).

Os FPGAs comerciais comumente incluem memórias e multiplicadores, entre outras possibilidades que poderiam ser consideradas na inclusão de *hard blocks*, assim como crossbars (JAMIESON, 2007) e multiplicadores de ponto flutuante (BEAUCHAMP, 2006). A seleção específica de cada bloco aumenta a complexidade da implementação dos circuitos e incluí-los em um FPGA é uma das questões centrais das arquiteturas FPGAs. Esses blocos podem oferecer significantes benefícios quando usados, caso contrário, são desperdiçadas (LUU, 2009). Na Figura 4, podemos visualizar os *hard blocks* (coloridos) diferenciados das CLBs (cinza).

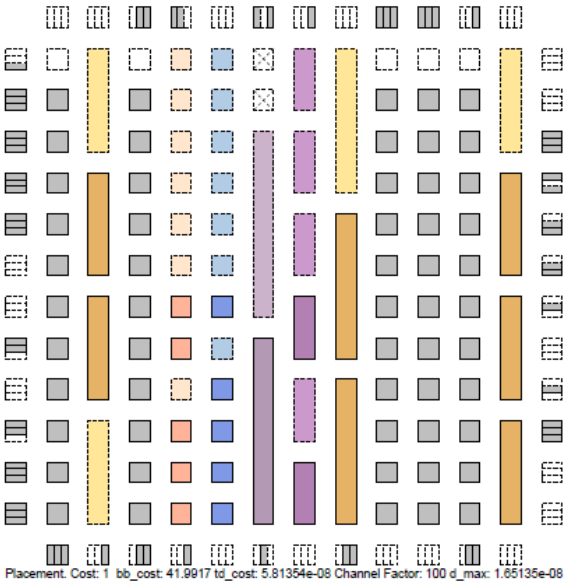


Figura 4: Representação de um FPGA heterogênea no VPR 5.0 (LUU, 2009).

2.1.4 Arquitetura de FPGA *Pipe-routed*

Devido a necessidade de implementar projetos *pipelineds em FPGAs,* um outro tipo de arquitetura foi proposta por pesquisadores. A primeira delas foi a *High-Speed, Hierarchical Synchronous Reconfigurable Array* (HSRA) que é um

exemplo de arquitetura de FPGA que tem hierarquia - *pipelined interconnect structure* (TSU, 1999). Na arquitetura HSRA, uma fração dos *switch boxes* é preenchida com registradores *switches* para atingir o período de relógio (*clock* do *pipeline*) desejado. Além disso, em vez de ter um único registrador na saída da LUT (que geralmente é o caso nas arquiteturas de FPGA existentes), um banco de registradores é conectado em cada entrada da LUT, o que ajuda a balancear os atrasos dos caminhos introduzidos pela interconexão *pipelined*. Aplicações do usuário são mapeados para o HSRA por dados integrados re-temporizados com um convencional fluxo CAD para FPGA.

Um segundo exemplo para arquiteturas de FPGAs *pipelined* é proposto em Singh et al (2001) no qual a arquitetura de roteamento é hierárquica, e o maior nível de roteamento consiste nas linhas longas verticais e horizontais que cercam os blocos lógicos. Cada linha longa é *pipelined* usando um banco de registradores *switch-points*, e todo *switch-point* pode ser usado para atrasar uma linha longa de 0–4 ciclos de relógio (*clock*).

2.2 Tecnologia de Programação de FPGAs

Existem três tipos de técnicas que permitem fabricar FPGAs: LUT, baseada em SRAM (FPGAs da fabricante Xilinx), multiplexador, baseada em anti-fusíveis (FPGAs da Actel) e PLD, baseada em EPROM (FPGAs da Altera).

Atualmente, a tecnologia mais popular usa células SRAM para controlar os transistores de passagem, multiplexadores e *buffer tri-state* permitindo assim configurar todo o roteamento programável e blocos lógicos, segundo a necessidade de cada circuito. Muitos FPGAs da Xilinx, as maiores pastilhas da Altera, os FPGAs da Actel, os FPGAs da *Lucent Tecnologies* e os VFI FPGAs da Vantis são baseados em células SRAM (CABRAL, 2001). A Figura 5 mostra esses três tipos de *switches* baseados em células SRAM.

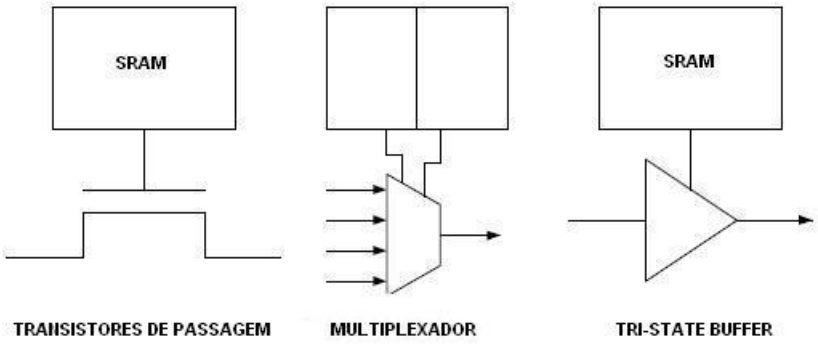


Figura 5: Três tipos de *switches* utilzado em FPGAs SRAM.

2.3 Arquitetura dos Blocos Lógicos

O tipo de bloco lógico usado em um FPGA influencia fortemente a velocidade e a eficiência em sua área. Por este motivo, diferentes tipos de blocos lógicos têm sido usados (BROWN, 1992), porém a maioria dos FPGAs comercialmente disponíveis utilizam blocos baseado em *look-up-tables* (LUTs).

A Figura 6 mostra como uma LUT de três entradas pode ser implementada baseada em células SRAM – uma LUT de k-entradas requer 2^k células SRAM e um multiplexador com 2^k-entradas. Uma LUT de k-entradas pode implementar qualquer função de k-entradas simplesmente programando as 2^k células SRAM de modo a obter-se a tabela verdade da função desejada (CABRAL, 2001).

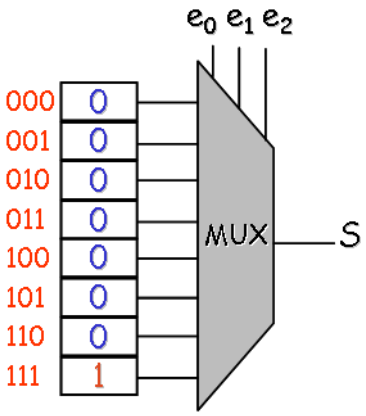


Figura 6: Uma LUT 3-entradas

Nos FPGAs disponíveis comercialmente como, por exemplo, da empresa Altera, os blocos lógicos LUTs possuem geralmente quatro ou cinco entradas, o que

permite endereçar 16 ou 32 células. Muitos FPGAs modernas têm seus blocos lógicos compostos não apenas por uma única LUT, mas sim por um grupo de LUTs e registradores com alguma interconexão local entre eles. A Figura 7 mostra vários tipos de blocos lógicos composto por uma, duas ou mais LUTs.

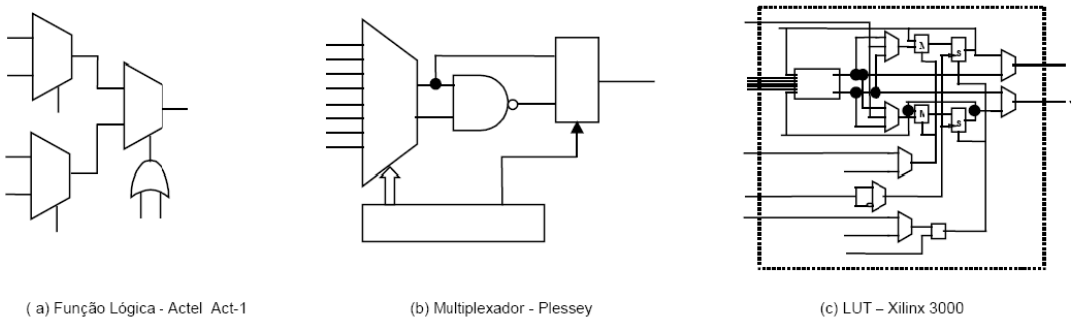


Figura 7: Exemplo de estruturas de blocos lógicos
(TOROK, 2001).

2.4 Arquiteturas de Roteamento de FPGAs

A arquitetura de roteamento de um FPGA é a maneira através da qual são configuradas as chaves e segmentos de fios para realizar as interconexões dos blocos lógicos. Nesta seção apresenta-se uma arquitetura utilizada comercialmente como referência para o estudo das arquiteturas de roteamento de forma genérica, observando-se que cada fabricante possui um modelo particular (TOROK, 2001). Os FPGAs comerciais podem ser classificadas em três grandes grupos, baseados em sua arquitetura de roteamento. Os FPGAs da XILINs, Lucent e Vanlis são baseadas em matrizes (*island-style*) enquanto os FPGAs da Actel são baseados em linhas e os da ALTERA são hierárquicas.

A Figura 8 define um modelo genérico de arquitetura de roteamento de FPGAs comerciais. Neste modelo, segundo Torok (2001), podemos identificar várias características:

- Um segmento de fio é contínuo.
- Uma ou mais chaves podem ser ligadas a segmentos de fio.
- Cada final de um segmento de fio está tipicamente ligado a uma chave programável.
- Uma trilha é uma sucessão em linha de um ou mais segmentos de fio.

- Um canal de roteamento é um grupo de trilhas paralelas.

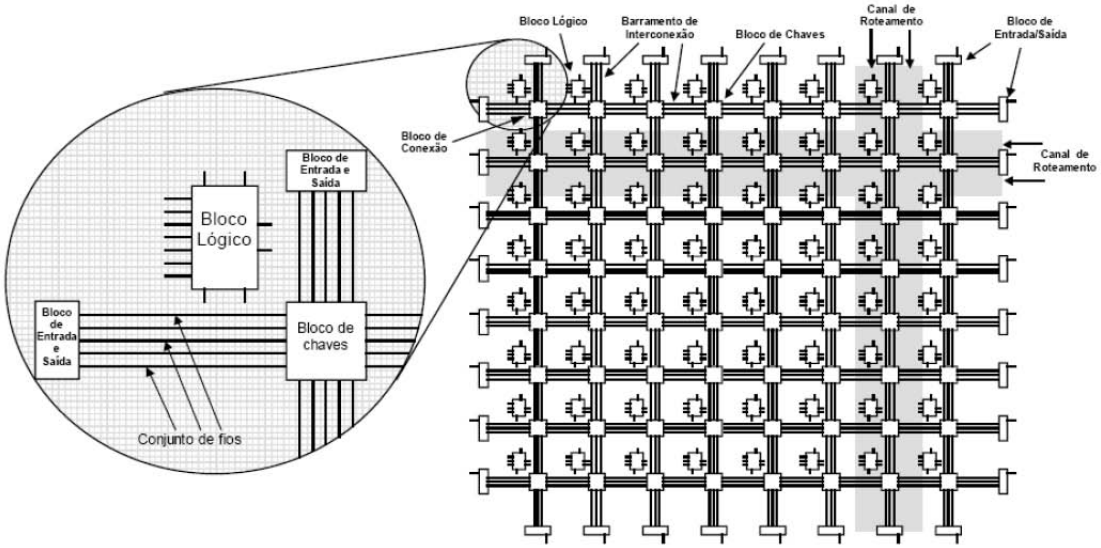


Figura 8: Arquitetura de roteamento genérica
(TOROK, 2001).

Levando as características anteriores em consideração, será admitido que todos os canais tenham o mesmo número de trilhas. O número de trilhas em cada canal é denotado por *W*. As trilhas em cada canal horizontal são numeradas de baixo para cima e, em cada canal vertical, da esquerda para a direita. Um número associado a uma trilha corresponde ao seu identificador. A flexibilidade do bloco *C*, denominado por *Fc*, é definida como o número de trilhas ao qual um pino lógico pode se conectar. A Figura 9 (b) ilustra um bloco *C*, onde observa-se as trilhas que passam através dele e podem ser conectadas aos pinos de blocos lógicos via um conjunto de *switches*. Na figura 9 (b), uma opção de roteamento é representado por um *X*, de modo que cada pino pode ser conectado para duas trilhas verticais, ou seja, *Fc = 2*. Já na figura 9 (c), existem três pinos lógicos no bloco *C* e cada um deles pode se conectar a qualquer uma das quatro trilhas, logo, conclui-se que a flexibilidade deste bloco é quatro.

Um bloco *S* é uma caixa retangular de *switches* (chaves programáveis) que conecta segmentos de fios em dois canais distintos. Dependendo da topologia, cada segmento de fio sobre um lado do bloco S pode ser conectado para um ou para todos, ou ainda para alguma fração dos segmentos de fios sobre cada um dos outros lados de um bloco *S*. A flexibilidade do bloco *S* é dada pelo parâmetro *Fs*,

que define o número de outros segmentos de fios aos quais um segmento em um bloco *S* pode se conectar.

Um exemplo de bloco *S* aparece na Figura 9 (c), onde cada linha tracejada representa um *switch* de roteamento programável – nesta figura, *Fs = 3*. A flexibilidade e a topologia adotada nos blocos *S* tem considerável impacto na roteabilidade das redes (*nets*). A flexibilidade do bloco S, denominada por *Fs*, é definida como o número de segmentos de fios ao qual cada segmento de fio pode conectar-se. Por exemplo, a Figura 9 mostra todas as possibilidades de conexões para o segmento de fio do lado direito, que está sobre a trilha 1. Ele pode se conectar ao segmento de fio na parte superior sobre a trilha 4, ou na parte de baixo sobre a trilha 1, ou ainda no lado esquerdo sobre a trilha 3. Supõe-se que todos os outros segmentos de fios adjacentes a este bloco *S* possam somente conectar-se a outros três outros segmentos de fios. Assim, sua flexibilidade é 3.

A figura 9 (c), ilustra um exemplo de um bloco *S* com flexibilidade básica. Um outro interessante bloco *S* é o bloco *S* identidade, que permite que todos os segmentos com o mesmo identificador de trilha sejam conectados e possui flexibilidade básica. O bloco *S* diagonal (Figura 1 (a)) usado no FPGA XC4000 é deste tipo.

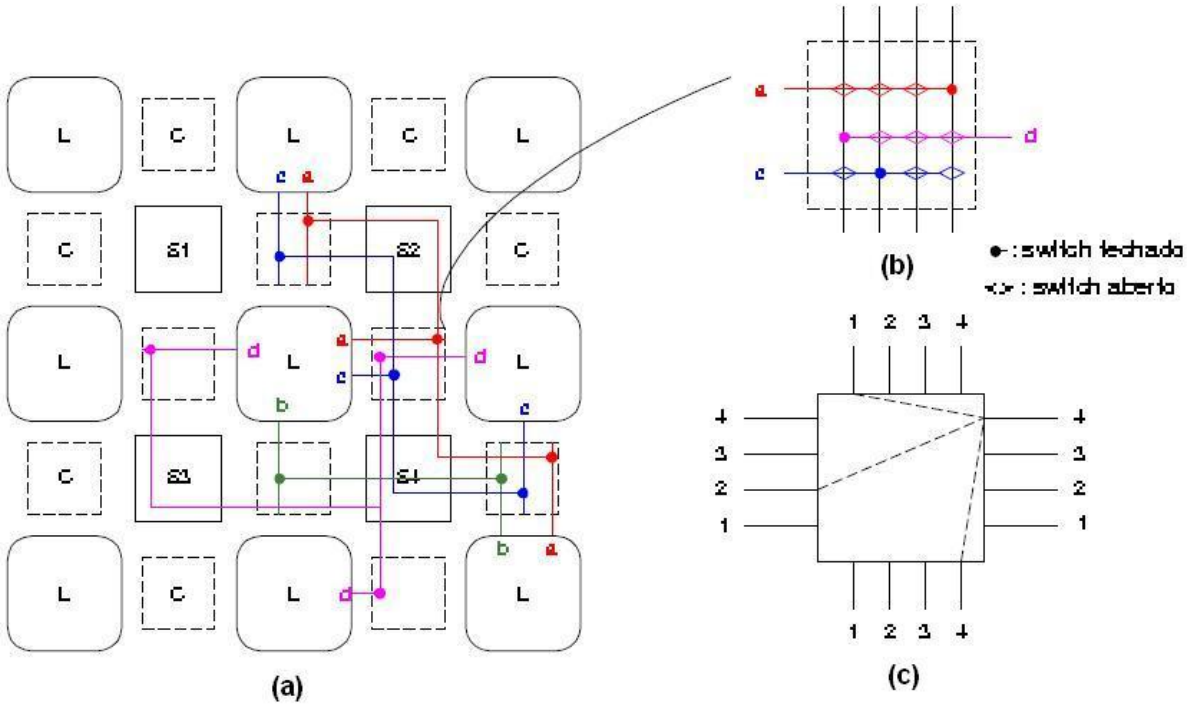


Figura 9: (a) Exemplo de roteamento. (b) Um bloco C. (c) Um bloco S.

2.5 CAD para FPGAs

Implementar um circuito em um FPGA moderno requer que centenas de milhares ou mesmo milhões de *switches* programáveis e bits de configuração sejam configurados adequadamente (ligados ou desligados). Pode-se observar que esta tarefa somente é possível com o uso de ferramentas CAD. Usuários de FPGA descrevem um circuito em alto nível de abstração, tipicamente usando uma linguagem de descrição de hardware (VHDL, System Verilog, System C, etc) ou entrada esquemática. Ferramentas de CAD convertem esta descrição de alto nível em um arquivo de programação especificando o estado (ligado ou desligado) de cada *switch* programável dentro do FPGA. Devido à complexidade desta tarefa, ela é dividida em vários passos, mostrados na Figura 10 (CABRAL, 2001).

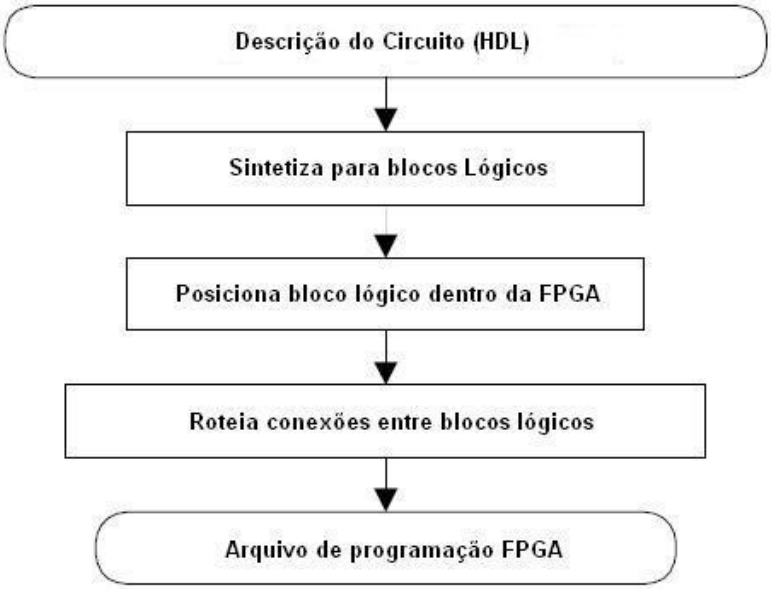


Figura 10: Fluxo de CAD para FPGA.

2.5.1 Síntese e Empacotamento de Blocos Lógicos

O primeiro estágio converte a descrição inicial de circuito, em geral feita através de uma linguagem de descrição de hardware ou esquemático, numa *netlist* de portas básicas. Em seguida, o processo de síntese converte esta *netlist* de portas básicas em uma *netlist* de blocos lógicos da FPGA de tal maneira que o

número de blocos lógicos necessários seja minimizado e/ou a velocidade do circuito seja maximizada. A tarefa de síntese lógica é suficientemente complexa, tanto que é geralmente dividida em dois ou mais subproblemas. A figura 11 mostra um exemplo de decomposição do procedimento de síntese em três subproblemas, os quais são descritos sucintamente a seguir (CABRAL, 2001).

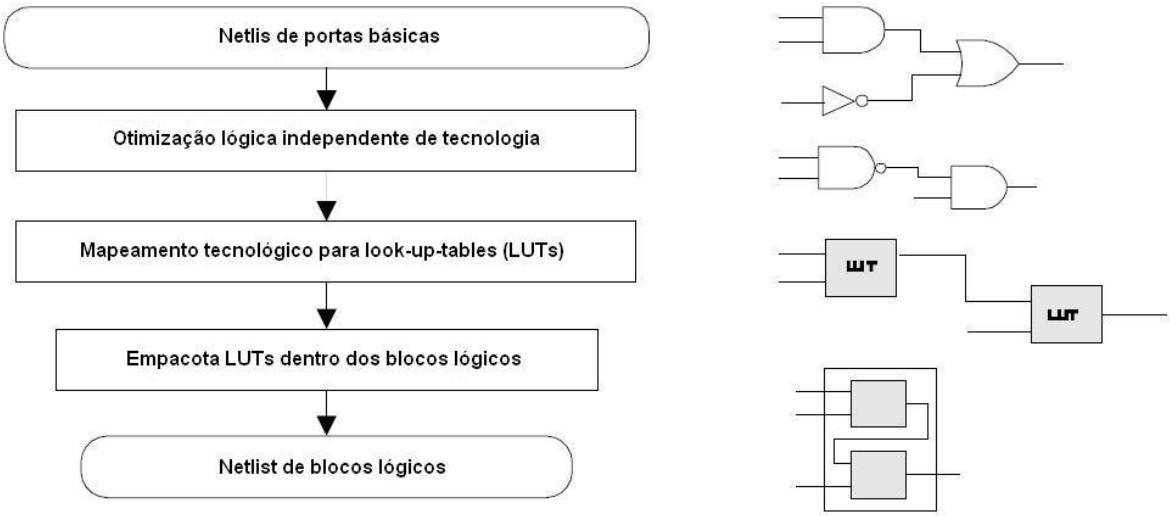


Figura 11: Detalhes do procedimento de síntese.

A otimização lógica, independente da tecnologia, remove lógica redundante e efetua simplificações lógicas sempre que possíveis. A *netlist* otimizada de portas lógicas é então mapeada para *look-up-tables*. O terceiro passo do processo de síntese é necessário sempre que um bloco lógico do FPGA contenha mais de uma LUT. Este passo agrupa diversas LUTs e registradores dentro de um bloco lógico, respeitando limitações, tais como: o número de LUTs, o número de sinais de entrada distintos e *clocks* que um bloco lógico pode conter. As metas de otimização, nessa fase, são empacotar tão próximo quanto possível LUTs conectadas, de modo a minimizar o números de sinais a serem roteados entre blocos lógicos e tentar preencher cada bloco lógico em toda a sua capacidade, de forma a minimizar o número de blocos lógicos usados (CABRAL,2001).

2.5.2 Posicionamento

Algoritmos de posicionamento determinam qual bloco lógico dentro de um FPGA deverá implementar cada uma das funções lógicas requeridas pelo circuito. As metas de otimização são:

- Posicionar, de forma aglutinada, blocos lógicos conectados minimizando assim o comprimento total de fio requerido (*wirelength-driven placement*).
- Posicionar blocos lógicos de modo a balancear a densidade de fiação ao longo dos canais do FPGA (*routability-driven placement*).
- Maximizar a velocidade do circuito (*timing-driven placement*).

Atualmente as três principais classes de posicionadores em uso são *min-cut* (*partitioning-based*) (ROSE, 1985), (HUANG, 1997), analítico (ALPERT, 1997), (KAHNG, 1992) e um destaque maior para o *simulated annealing* por ter sido utilizado nos artigos mais recentes. (BETZ, 1997), (SWARTZ, 1995). No caso do *simulated annealing*, tem-se a vantagem de ser possível adicionar facilmente novas metas de otimização ou restrições ao posicionador (CABRAL, 2001).

2.5.3 Roteamento

Após as localizações (posições) para todos os blocos lógicos terem sido definidas, um roteador determina quais *switches* programáveis devem ser ligadas para conectar todos os pinos de entrada e saída do circuito. É usual representar a arquitetura de roteamento de um FPGA através de um grafo $G_A=(N,D)$ orientado (dirigido) (NAG, 1994), (EBELING, 1995). Cada fio e cada pino de bloco lógico torna-se um nó $n \in N$, no grafo de recursos de roteamento e cada *switch* torna-se um arco orientado $a \in D$ (para *switches* unidirecionais, tais como *buffers*) ou um par de arcos orientados (para *switches* bidirecionais, tais como transistores de passagem) ou um par de arcos orientados (para *switches* bidirecionais, tais como transistores de passagem) entre dois nós apropriados. A figura 12 mostra o grafo de roteamento de recursos correspondendo a uma porção de uma FPGA cujos blocos lógicos contém uma única LUT com 2-entradas e 1-saída. Alguns trabalhos têm representado FPGAs como grafos não orientados (ALEXANDER, 1994), mas uma

representação de grafo orientado é necessária se *switches* direcionais, do tipo *buffers tri-state* e multiplexadores, devem ser modelados corretamente.

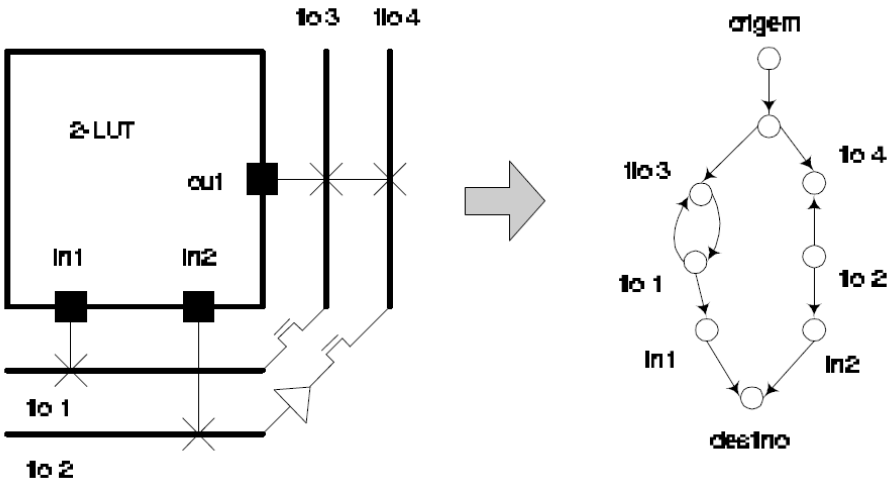


Figura 12: Modelando o roteamento detalhado FPGA como um grafo orientado.

Freqüentemente blocos lógicos FPGA têm pinos logicamente equivalentes. Por exemplo, todos os pinos de entrada de uma LUT são logicamente equivalentes. Isto significa que um roteador pode completar uma dada conexão usando qualquer um dos pinos de entrada de uma LUT. Esta equivalência lógica é modelada no grafo de roteameto de recursos adicionando-se nós fontes (*sources*), nos quais todas as redes começam e nós destinos (*sinks*) onde todos os terminais de rede terminam. Vale salientar que existe um nó fonte para cada conjunto de pinos de saída logicamente equivalentes e um arco do nó fonte para cada um destes pinos de saída. De forma similar, existe um nó destino para cada conjunto de pinos de entrada logicamente equivalente e um arco de cada um desses pinos de entrada para um nó destino. Portanto, cada rede de sinal consiste em um único nó de G_A, nó origem do sinal, junto com um subconjunto de nós correspondente aos seus destinos (*sinks*).

Rotear uma rede de sinal (*net*) corresponde a encontrar uma árvore neste grafo de recurso de roteamento entre os nós que representam os pinos de blocos lógicos a serem conectados, de tal forma que todos os nós destinos possam ser alcançados no nó origem. Quando o número de fios (recursos de roteamento) de um FPGA é limitado deve-se buscar por árvores as mais curtas possíveis. É importante ressaltar que o roteamento de uma rede (*net*) não use recursos de

roteamento que outra rede (*net*) necessite. Assim, a maioria dos roteadores de FPGA tem algum tipo de mecanismo de prevenção de congestionamento para resolver a disputa por recursos de roteamento.

Uma meta adicional de otimização é construir redes sobre ou próximas do caminho crítico, roteando-as usando caminhos mais curtos e recursos de roteamento mais rápidos. Roteadores que tentam aperfeiçoar desempenho desta maneira são chamados de *timing-drive* e desde que grande parte do atraso em um FPGA se deve ao roteamento programável, torna-se essencial aplicar um roteador deste tipo para que se obtenha circuitos com bons desempenhos.

Os roteadores de FPGA podem ser divididos em dois grupos: roteadores *global-detalhado combinado* que determinam totalmente um caminho de roteamento em um único passo, enquanto algoritmos de roteamento em dois passos primeiramente executam o roteamento global para determinar quais pinos de blocos lógicos e canais cada *net* irá usar, e então, executam o roteamento detalhado para determinar os fios que cada *net* irá usar dentro de cada um dos canais especificados. A tarefa de um roteador detalhado para FPGA é freqüentemente difícil ou impossível porque o roteamento FPGA tem flexibilidade limitada e o roteador detalhado é altamente restringido pelas decisões do roteador global sobre quais canais (seqüência de blocos de conexão C) cada *net* deve usar. Roteadores *global-detalhado combinado* têm maiores chances de aperfeiçoar o roteamento, uma vez que eles estão livres de tais restrições.

2.5.3.1 Flexibilidade das Estruturas de Interconexão para FPGAs

No trabalho de Rose et Al (1991) estudou-se o projeto da estrutura de interconexão e o efeito de sua flexibilidade sobre a probabilidade da roteabilidade de um FPGA, assim como sobre os requisitos de recursos de fiação. A probabilidade de roteabilidade se traduz no percentual do número total de conexões que são efetivamente roteadas.

Este trabalho levanta algumas questões:

- Qual é o efeito da flexibilidade do bloco de conexão C, denotada por Fc, sobre a taxa de sucesso de roteamento?
- Qual o efeito da flexibilidade do bloco de interconexão S, denotada por Fs, sobra a taxa de sucesso de roteamento?

- De que maneira interagem Fc e Fs?
- Qual o efeito de Fc e Fs sobre o número de trilhas por canal requeridas para alcançar 100% de roteamento?

De modo a tentar responder todas as questões anteriormente apresentadas, foi implementado um conjunto de circuitos industriais sobre uma variedade de estruturas de interconexão e foram medidos os percentuais de sucesso de roteamento e o número requerido de trilhas por canal para cada circuito. Para isso foi adotado o seguinte procedimento de implementação:

1. Executar o mapeamento tecnológico do circuito original no bloco lógico; o resultado deste passo é uma nova *netlist* que interconecta somente blocos lógicos e é funcionalmente equivalente ao circuito original;
2. Executar o posicionamento (*placement*) da *netlist* resultante usando ALTOR (ROSE, 1985);
3. Executar o roteamento global do circuito, que determina o caminho, consistindo em uma sequência de canais pelos quais cada fio deverá passar. Isto fornece o número de trilhas que serão necessárias em cada canal, supondo que os blocos de interconexão S e de conexão C têm flexibilidade total;
4. Executar o roteamento detalhado do circuito. Para cada caminho global (rota global), determine os fios exatos nos blocos de interconexão S e de Conexão C.

Ao final tem-se como saída duas possibilidades: o percentual de redes que foram roteadas com sucesso dado um número particular de W de trilhas por canal; ou o número de trilhas por canal necessário para atingir 100% das conexões para um da Fs e uma dada razão Fc / W.

Com base nos resultados obtidos neste estudo (ROSE, 1991), as principais conclusões do trabalho foram que os blocos de conexão C devem ter alta flexibilidade para alcançar altos percentuais de roteamento, e que os blocos de interconexão S requerem flexibilidade limitada.

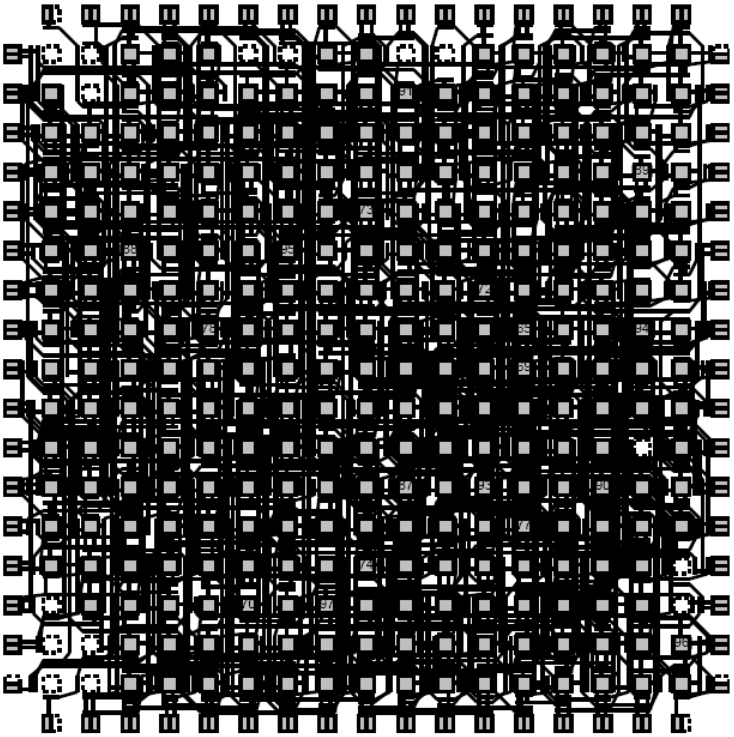


Figura 13: Exemplo de roteamento de FPGA (Imagem retirada da ferramenta VPR).

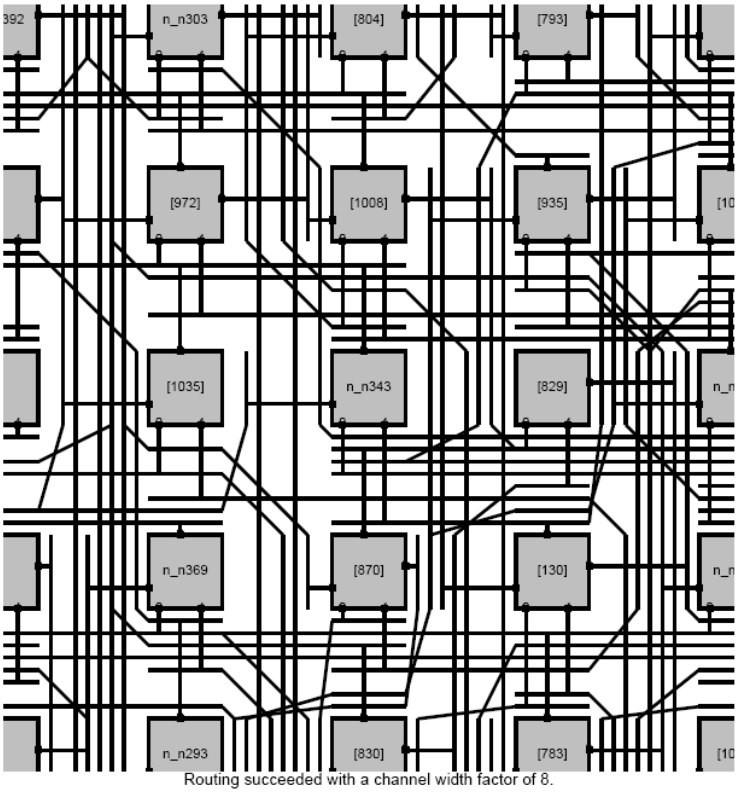


Figura 14: Visão ampliada de um exemplo de roteamento FPGA (Imagem retirada da ferramenta VPR).

Capítulo 3 – Revisão da Literatura

Nesta seção será feita uma revisão bibliográfica relativa aos algoritmos mais recentes propostos na literatura para resolver o problema de roteamento de FPGAs sobre as instâncias MCNC e as metaheurísticas GRASP e ILS.

3.1 Roteador detalhado SAT

O algoritmo SAT, apresentado por Lee et al (2003), apresenta um algoritmo efetivo para roteamento *timing-driven* de FPGAs. O algoritmo resolve o problema de minimizar atrasos ou caminhos críticos para respeitar as condições de tempo. Nesta abordagem, as condições de tempo são manipuladas baseadas em um *framework* de programação matemática utilizando a relaxação Lagrangeana. A abordagem de relaxação Lagrangeana transforma o problema de roteamento em uma seqüência de subproblemas. Cada subproblema pode ser simplificado pela exploração da topologia da *network* (CHEN, 1997). Em cada iteração do algoritmo, mudança no atraso de cada par *source-sink* da *net* é refletido em um valor que corresponde ao multiplicador Lagrangeano. Incorporados na função custo, esse multiplicadores guiam o roteador.

3.1.1 Roteamento de FPGAs Timing-Driven

Como no VPR, neste algoritmo é usado atraso Elmore (ELMORE, 1948) para modelar os componentes no FPGA. O atraso do *source* para o *sink* de uma cadeia de *wire-switchs*, através dos recursos de roteamento, podem ser calculados pelos valores *RC* especificados pela arquitetura do FPGA. O atraso através do par I/O em um módulo lógico pode ser calculado pelos valores especificados pela arquitetura através das capacitâncias do *drive*r de entrada, resistências de saída, e atraso do pino de entrada e o pino de saída. Atrasos através dos módulos I/O podem ser obtidos de modo similar.

Em roteamentos *timing-driven* as condições de tempo em um circuito são especificados como os tempos de chegada à entrada primária ou saída de

elementos de armazenamento, e os tempos necessários nas saídas primárias ou entradas de elementos de armazenamento (HITCHCOCK, 1988). Entretanto, especialmente para um grande número de circuitos, o número de caminho de sinais possíveis para uma entrada primária e para uma saída primária pode ser exponencial quanto aos números de *nets*. Pode-se lidar com esta dificuldade através do particionamento de restrições sobre os atrasos ao longo de caminhos em restrições sobre o atraso de cada par *source-sink.* Para uma *netlist* posicionada, dada com um conjunto de entradas e saídas, o objetivo é rotear todas as *nets* de uma maneira em que o atraso dos caminhos críticos seja minimizado, enquanto as restrições de atraso e congestionamento sejam satisfeitas.

Para realizar análise de tempo para roteamento *timing-driven*, constrói-se um gráfico de tempo $G_t(V_t,E_t)$, um grafo direcionado acíclico, de uma *netlist* de entradas. Como mostrado na Figura 15, os nós do gráfico de tempo correspondem as entradas primárias, saídas primárias, e entradas e saídas de módulos lógicos. Os arcos do grafo de tempo corresponde ao par *source-sink* de cada net ou para I/O de módulos lógicos. Note que um arco em G_t é diferente de uma *net* na *netlist*. Decomposta as restrições de tempo ao longo dos caminhos para os atrasos do *source-to-sink*, cada par *source-sink* corresponde a um arco em G_t até para uma net com múltiplos *fan-outs*. A Figura 16 mostra um exemplo cujos pares source-*sink* (pin3, pin9) e (pin3, pin12) pertencem a caminhos diferentes. Para uma maior consistência nas notações, dois nós fictícios, *s* e *t*, são introduzidos. O nó *s* é conectado a todas as entradas primárias, e todas as saídas primárias são conectadas ao nó *t*.

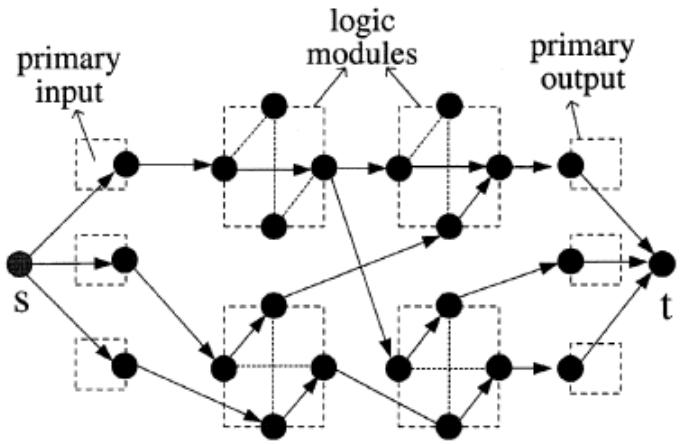


Figura 15: Grafo de tempo para uma *netlist* posicionada (LEE, 2003)

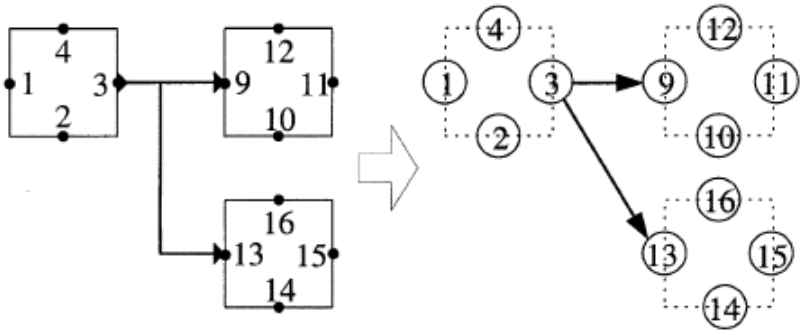


Figura 16: Net de *fun-out* múltiplo e seus correspondentes arcos em um grafo de tempo (LEE, 2003).

Considerando as seguintes nomenclaturas:

- E_s é o subconjunto dos E_t conectados ao nó *s.*
- E_t sendo o subconjunto de E_t conectado para o nó *t*.
- E_m todos os outros nós
- O tempo de chegada no nó *u* é denotado por a_u.
- D_{uv} representa o atraso de roteamento para um par *source-sink (u, v)* de uma *net* ou de atraso entre o par I/O *(u, v)* em um módulo lógico.
- T_k é a árvore de roteamento que consiste de nós em G_r para *net k*.

Para um par *source-sink (u, v)* para *net k*, D_{uv} pode ser expresso como se segue:

$$Duv = \sum_{i \in path(u,v)} di \qquad (1)$$

Onde o caminho *(u, v)* é o conjunto de nós ao longo do caminho do *source u* para o *sink v* em T_k e d_i denota a contribuição de atraso do nó *i* para o atraso ao longo do caminho *(u,v)* onde pode ser calculado através dos valores *R* e *C* dos recursos de roteamento ao longo do caminho. D_{sv} denota o tempo de chegada para cada entrada primária. O problema do roteamento de minimização do atraso do caminho crítico sobre restrições de tempo e congestionamento é achar o vértice disjunto de árvores de roteamento $T=\{T_1; T_2;...;T_n\}$ para todas as *nets* que:

Minimize a_t

Subject to $a_u \leq a_t \forall (u,t) \in E$

$$a_u + D_{uv} \leq a_c \forall (u,v) \in E_m$$

$$D_{sv} \leq a_v \forall (s,v) \in E_s$$

3.1.2 Descrição do algoritmo

Relaxação Lagrangeana é uma técnica para solucionar problemas de otimização com restrições. Na relaxação Lagrangeana, restrições são relaxadas e são adicionadas as funções objetivas depois de serem multiplicadas por constantes chamadas de multiplicadores Lagrangeano. Para um vetor dado de multiplicadores Lagrangeanos, a solução ótima de um subproblema Lagrangeano fornece um limite inferior perto do valor ótimo da função objetivo do problema original. O problema de achar este vetor é chamado de problema Lagrangeano dual. Solucionando o subproblema Lagrangeano com o vetor obtido resolvendo o problema Lagrangeano dual, pode-se obter um limite inferior perto do valor ótimo da função objetivo do problema de otimização original. Esta técnica é discutida nos trabalhos de Ahuja et al (1993) e Bazaraa et al (1993).

3.1.2.1 Relaxação Lagrangeana

Relaxar as restrições de tempo é o problema original. As restrições de congestionamento são manipuladas pelo o roteador de *nets* que resolve o subproblema Lagrangeano. Cada restrição é multiplicado por um correspondente multiplicador Lagrangeano, e adicionado a função objetivo. Sendo:

$$L_\lambda(a,T) = a_t + \sum_{(u,t)\in E_t} \lambda_{ut}(a_u - a_t) + \sum_{(u,t)\in E_m} \lambda_{ut}(a_u + D_{uv} - a_v) + \sum_{(u,t)\in E_s} \lambda_{sv}(D_{sv} - a_v)$$

Essa função objetivo relaxada é chamada de função Lagrangeana, e o subproblema Lagrangeano associado com o conjunto fixo dos multiplicadores Lagrangeanos λ é LS_λ: *Minimize* L_λ *(a,t).* Por causa do valor mínimo de L_λ *(a,t)* para qualquer vetor λ é um limite inferior no valor ótimo da função objetivo do problema original, o limite inferior perto do valor ótimo objetivado do problema original é obtido resolvendo $L^* = max\ LS_\lambda$ *onde* $\lambda \geq 0$ que é conhecido como o problema Lagrangeano dual.

Há condições nos multiplicadores Lagrangeanos λ correspondendo a solução ótima do problema original, e podem ser derivadas usando a condição otimizada Kuhn-Tucker (BAZARAA, 1993). Essa condição implica em $\partial L_\lambda / \partial a_u = 0 \forall u \in V_t$.

Pela aplicação das condição de Kuhn-Tucker na função Lagrangeana, obtêm-se as seguintes condições otimizadas em λ:

$$1 = \sum_{(u,t)\in E_t} \lambda_{ut} \qquad (2)$$

$$\sum_{(w,u)\in E_t} \lambda_{wv} = \sum_{(u,w)\in E_t} \lambda_{uw} \forall w \in V_t - \{s,t\} \qquad (3)$$

Devido a estrutura única deste problema, pode-se simplificar LS_λ pela aplicação dessas condições. Pela rearranjo dos termos, a função Lagrangeana L_λ *(a,T)* pode ser escrito como:

$$L_\lambda(a,T) = \left(1 - \sum_{(u,t)\in E_T} \lambda_{ut}\right) + a_t + \left(\sum_{(w,u)\in E_m} \lambda_{wv} - \sum_{(u,w)\in E_m} \lambda_{uw}\right) a_w + \sum_{(u,v)\in E_M} \lambda_{uv} D_{uv}$$

Quando λ satisfaz as condições ótimas, L_λ *(a,T)* é simplificado para $L'_\lambda(T) = \sum_{(u,v)\in E_s \cup E_M} \lambda_{uv} D_{uv}$. Sendo assim LS_λ é simplificado: LS'_λ: *Minimize* L'_λ *(T).* LS'_λ não tem os termos a_u e solucionar LS_λ é equivalente a solucionar LS'_λ.

Para solucionar o problema Lagrangeano dual, uma abordagem iterativa pode ser usada. Em cada iteração, soluciona-se LS_λ através de LS'_λ para um λ dado, e então atualiza-se os multiplicadores Lagrageanos para a próxima iteração usando a solução da corrente iteração. Os multiplicadores Lagrangeanos para a interação *(r + 1)* são atualizados pelo método de subgradiente (AHUJA, 1993 e BAZARAA, 1993) como se segue:

$$\lambda_{ut}^{r+1} = \max\{0, \lambda_{ut}^{r} + \theta_r (a_u - a_t)\} \forall (u,t) \in E_T$$

$$\lambda_{uv}^{r+1} = \max\{0, \lambda_{uv}^{r} + \theta_r (a_u + D_{uv} - a_v)\} \forall (u,v) \in E_M$$

$$\lambda_{sv}^{r+1} = \max\{0, \lambda_{sv}^{r} + \theta_r (D_{sv} - a_v)\} \forall (s,v) \in E_S$$

onde θ_r é o tamanho de um etapa com a propriedade que $lim_{r\to\infty}\ \theta_r = 0$ e:

$$\lim_{k\to\infty} \sum_{r=1}^{k} \theta_r = \infty$$

Sendo Λ um não negativo λ que satisfaz as condições ótimas (2) e (3). Com isso resolve-se LS'_λ em vez de LS_λ e o λ atualizado é projetado para o mais perto vetor em Λ em cada iteração. A Figura 17 resume o algoritmo *LR_ROUTE*. Com um vetor dado λ, o algoritmo *Net_Route* soluciona LS'_λ.

Algorithm LR_ROUTE
Input: Timing graph $G_t(V_t, E_t)$,
routing graph $G_r(V_r, E_r)$
Output: A routed netlist
begin
1. Initialize $\boldsymbol{\lambda}$ as an arbitrary vector in $\boldsymbol{\Lambda}$
2. Call NET_ROUTE with $\boldsymbol{\lambda}$ to solve LS'_λ.
3. Compute a_u for each $u \in V_t$.
Set each a_u to the smallest possible value that satisfies the timing constraints in a topological order from s to t.
4. Update λ_{uv} for each $(u, v) \in E_t$.
5. Project $\boldsymbol{\lambda}$ to the nearest vector in $\boldsymbol{\Lambda}$.
6. Repeat Step 2-5 until no shared resource exists and $(a_t - LS_\lambda) \leq \epsilon$, where ϵ is an error bound.
end

Figura 17: Algoritmo LR_ROUTE

3.1.2.2 Roteando Nets

Para rotear as *nets*, leva-se em consideração solucionar o subproblema Lagrangeano simplificado LS'_{λ} com um vetor dado λ. Através do roteamento de cada *net* usando uma função custo apropriada, pode-se solucionar este problema. A função objetivo $L'_{\lambda}(T)$ deste problema implica que os arcos no gráfo de tempo necessitam ser roteados tal qual o atraso de Elmore ponderado com os multiplicadores Lagrangeanos são minimizados para um dado λ, porque D_{uv} denota o atraso do roteamento do arco (u,v). Enquanto roteando, entretanto, restrições de congestionamento também precisam ser satisfeitas de uma maneira em que todas as *nets* sejam roteadas com sucesso. Para garantir as restrições de congestionamento, uma variável de decisão para cada nó em G_r é definida como:

$$x_{ik} = \begin{cases} 1, & \text{if the routing tree } T_k \text{ for net } k \text{ uses node } i \\ 0, & \text{otherwise} \end{cases}.$$

De LS'_{λ} e restrições de congestionamento, o problema de rotear uma *net* é construir as árvores de roteamento T para todas as *nets* nas *nets* posicionadas para um conjunto de multiplicadores dados λ_{uv}'s de tal maneira que

$$\text{Minimize} \quad \sum_{(u,v)\in E_s \cup E_M} \lambda_{uv} D_{uv}$$

$$\text{Subject to} \quad \sum_{k} x_{ik} \leq 1 \forall i \in V_r$$

Esse problema pode ser resolvido através de relaxação Lagrangeana.

$$L_{\mu}(x) = \sum_{(u,v)\in E_S \cup E_M} \lambda_{uv} D_{uv} + \sum_{i\in V_r} \mu_i \left(\sum_{k} x_{ik} - 1 \right)$$

$$= \sum_{k} \left\{ \sum_{(u,v)\in E_k} \lambda_{uv} D_{uv} + \sum_{i\in V_r} \mu_i x_{ik} \right\} - \sum_{i\in V_r} \mu_i$$

onde E_k é um conjunto de pares *source-sink* pertencente a árvore de roteamento T_k para uma *net k*. Note que $\sum_{i \in Vr} \mu_i$ é um termo constante. O algoritmo *NET_ROUTE* constrói as árvores de roteamento para todas as *nets* que minimizam

$$L'_{\mu}(x) = \sum_{k} \left\{ \sum_{(u,v) \in E_k} \lambda_{uv} D_{uv} + \sum_{i \in V_r} \mu_i x_{ik} \right\} \qquad (4)$$

Este algoritmo é similar ao algoritmo *PathFinder* (BETZ, 1999). Dado um grafo de roteamento, o NET_ROUTE iterativamente constrói uma árvore de roteamento de custo mínimo para cada *net*. O algoritmo faz *rip-up* de cada *net* em um momento, e re-roteia com custos atualizados. Enquanto uma *net* é roteada, cada par *source-sink* é roteado seqüencialmente em ordem decrescente do λ_{uv}. Inicialmente, nós no grafo de roteamento são permitidos para serem compartilhados por múltiplas *nets*. Depois de cada iteração, o custo dos recursos compartilhados é gradualmente incrementado, e apenas a *net* com maior criticidade tentará usar os nós com os altos custos de congestionamento. Na *netlist* posicionada, cada *net* tem múltiplos *sinks*, e cada par *source-sink (u,v)* da cada *net* tem seu correspondente multiplicador Lagrangeano λ_{uv}. Cada recurso de roteamento *i* tem um multiplicador Lagrangeano μ_i. Para alcançar roteamento viável que minimize $L'_{\mu}(x)$, NET_ROUTE usa multiplicadores de Lagrange como pesos para o custo para usar recursos. De (1), o termo $\lambda_{uv} D_{uv}$ *pode ser expresso como:*

$$\lambda_{uv} D_{uv} = \sum_{i \in path(u,v)} \lambda_{uv} d_i$$

Assim, a contribuição da cada nó *i* no caminho *(u,v)* para (4) é dado por:

$$C_i = \lambda_{uv} d_i + \mu_i .$$

Nessa equação, o primeiro termo é um termo de controle de atraso, e o segundo termo é um termo de controle de congestionamento. Pares *source-sink*, pertencentes a mais de um caminho crítico tem grandes λ_{uv}'s, e o NET_ROUTE constrói árvores de roteamento com custos que tendem mais ao custo de atraso do

que os custos de congestionamento para estes pares. No algoritmo *PathFinder*, o termo sensitivo é definido como:

$$C_i = b_i * p_i$$

onde b_i é base do custo para os recursos de roteamento, e p_i é o termo de penalidade para o controle de congestionamento que consiste de um termo relacionado ao congestionamento durante as iterações prévias e o termo relacionado ao congestionamento na corrente iteração. Cada c_i pode ser interpretado como um multiplicador de Lagrange, e funciona da mesma maneira que μ_i, mas é atualizado de uma maneira diferente do método de subgradiente. Nesta corrente implementação, foi adotado o multiplicador c_i para controle de congestionamento, e definido $\mu_i = c_i$. A Figura 18 resume o algoritmo NET_ROUTE.

Algorithm NET_ROUTE
Input: A placed netlist, λ, $G_r(V_r, E_r)$
Output: A routed netlist
begin
1. **for each** net k **do**
2. Rip up routing for net k
3. **for each** sink v of net k **do**
4. Maze route from source to sink,
where cost at node i is
$C_i = \lambda_{uv} d_i + \mu_i$ for each node i of V_r
5. Update μ_i for all i's in $path(u, v)$
end

Figura 18: Algoritmo NET_ROUTE

3.1.3 Resultados

O levantamento dos resultados foram feitos em cima de 17 grandes circuitos do benchmark MCNC (YANG, 1991). As netlists posicionadas foram geradas usando o posicionador do VPR (BETZ, 1999). Foi assumido um FPGA *island style* (BROWN, 1992), onde cada bloco lógico contém quatro LUTs de quatro entradas e quatro *flip-flops*. Foi definidio que *Fs = 3* e *Fc = W*, onde *W* é o número

de segmentos de fio em cada canal. *Fs* denota o número de conexões para cada segmento de fio chegando em cada *switch-box*. *Fc* denota o número de trilhas que cada pino de bloco lógico pode conectar. Com o propósito de comparação, foi utilizado valor idêntico de atraso intrínseco e modelos de tempo usados no VPR. Entre os 17 circuitos, LR_ROUTE conseguiu melhor resultado para 13 circuitos, e os atrasos de caminho crítico foram reduzidos em 33% com comparável tempo de execução.

3.2 Roteador CornNC

O roteador *Congestion-Optimal Restrained-Norm Path* (CornNC), apresentado por Keith So (2007), apresenta um algoritmo efetivo para rotear circuitos em FPGA no critério de redução do número de trilhas (*wirelength-distance*). Neste trabalho, foi analisada a estabilidade numérica dos espaços escalares usados em roteadores com base na negociação de congestionamento. Foi mostrado que a projeção de congestionamento como métrica secundária para uma avaliação do espaço escalar levará a caminhos indesejáveis, sendo selecionados em alguns casos problemáticos, devido à instabilidade numérica. Baseado neste estudo, apresenta-se um vetor baseado em avaliação do espaço, deixando de lado esses problemas de projeção, bem como contendo propriedades úteis para o roteamento das FPGAs.

3.2.1 Mecanismo de Negociação de Congestionamento

Um algoritmo de roteamento baseado em negociação de congestionamento irá determinar custos para cada recurso do vértice do grafo de roteamento, e ocorrerá através das numeras iterações. Em cada iteração, toda *net* é re-roteada uma vez. Um roteador *maze* é usado para construir a árvore de roteamento necessária para achar o caminho de custo mínimo de um pino de saída para um pino de entrada para cada ramo da árvore requerida. A árvore de roteamento gerada pelo roteador *maze* é retida para cada *net* independentemente se a árvore causar qualquer caminho curto não permitido com o roteamento existente. Ao final da iteração, não haverá nenhum caminho aberto e o custo dos vértices que tenham um caminho curto não permitido será incrementado. O

algoritmo de roteamento continua até que um roteamento legal seja encontrado, ou depois de um número de iterações o roteador reporte que a instancia não é roteável.

3.2.2 Análise do Custo Escalar dos Espaços em Negociação de Congestionamento

Assumindo que o vértice ou o custo do caminho no grafo de recurso de roteamento são denominados como indicado pela tabela 1. Também assume-se que o roteador maze vai usar um algoritmo de busca baseado em um grafo ótimo assim como o *breadth-first search* ou *A* search.*

Cost components	Vertex v	Path x
Total cost	$f(v,i)$	$f(x,i)$
Present congestion	$p(v,i)$	$p(x,i)$
Historical congestion	$h(v,i)$	$h(x,i)$
Congestion component	$c(v,i)$	$c(x,i)$
Secondary cost	$s(v)$	$s(x)$
Occupancy (number of nets)	$\mathrm{occ}(v,i)$	$\mathrm{occ}(x,i)$
Capacity	$\mathrm{cap}(v)$	–

Tabela 1: Tipos dos custos de componentes na iteração *i* do algoritmo (SO,2007).

Considerando o roteador VPR (BETZ, 1997), usado no modo *wirelength-driven*. Nessa instância, o custo secundário de usar vértices é sua *wirelenght* (“comprimento do fio”) no número equivalente a um segmento de 1-CLB. O custo de usar um vértice é definido na equação 1. Os horários de atualização entre sucessivas iterações para o presente histórico de congestionamento são mostrados na Equação 2 através da 5, onde *I* e *P* são definidos pelo usuário como parâmetros. Em outras palavras, o presente custo de congestionamento é incrementado por um multiplicador fixo e o custo do histórico de congestionamento é incrementado se o vértice é sobrecarregado (conter *nets* curtas). Esses horários de atualização são tipicamente dos mais publicados algoritmos de negociação de congestionamento.

$$f(v,i) = occ(v,i).(p(v,i).h(v,i)) + s(v) \quad (1)$$

$$p(v,0) = I \quad (2)$$

$$p(v,1+1) = P \times p(v,1) \quad (3)$$

$$h(v,0) = 1 \quad (4)$$

$$h(v,i+1) = h(v,i) + \min(0, occ(v,i) - cap(v)) \quad (5)$$

Apresentamos o plano de compensação representado na Figura 19 com o propósito de estudar o equilíbrio entre o congestionamento e a componente dos custos secundários. No plano, qualquer caminho contíguo do source até o objetivo pode ser “plotado” através da separação dos custos totais no congestionamento, componentes do custo secundário e candidatos a caminho pode ser subseqüentemente comparados.

Supondo que o caminho *x* foi encontrado pelo grafo de busca ótimo. Isso implica que o caminho de custo mínimo no terreno é igual à soma da projeção de *x* para os eixos, ou seja, *cx* + *sx*, caso contrário, um caminho de menor custo, assim como *w* será escolhido. Assim, a linha de *Pareto* pode ser definida como mostrado na equação paramétrica *f (y) = f (x)*, e uma região inviável é mostrada delimitada pela linha de *Pareto* e os eixos, onde nenhum caminho pode existir por implicação do mínimo caminho de custo a ser *x*.

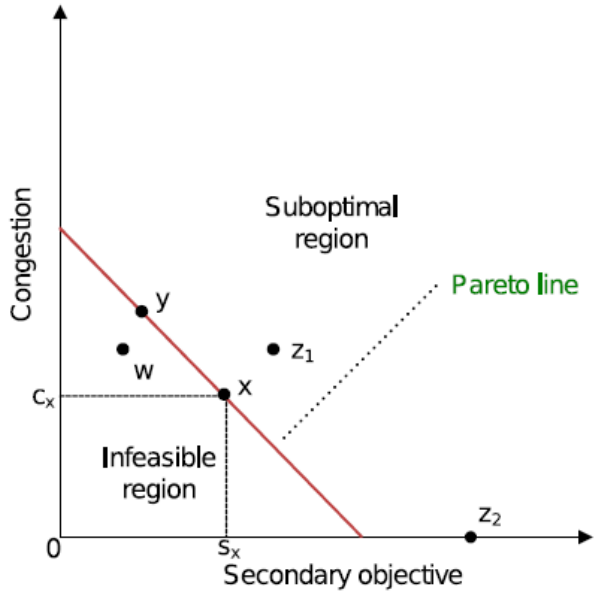


Figura 19: O plano de compensação de caminho após caminho x ser escolhido pelo roteador maze (SO,2007).

Note que os caminhos sub-ótimos no espaço escalar como Z_1 e Z_2 não serão escolhidos, apesar de Z_2 ser livre de congestionamento. Um exemplo motivador deste fenômeno é mostrado na Figura 20. Na figura, os segmentos de fio ocupados são mostrados em cinza. Um caminho de *s* para *t* precisa ser encontrado, mas o grafo de busca encontrará o caminho *sx1x2t* que contém um caminho curto não permitido, mas o caminho *sy1y2y3y4t* legal não será escolhido.

Define-se a norma do custo do caminho na iteração *n*, ||*f(x,n)*|| para ser o *"*táxi*"* ou a norma *L1* no plano de compensação. Isto é uma medição útil para a magnitude da representação dos custos escalares.

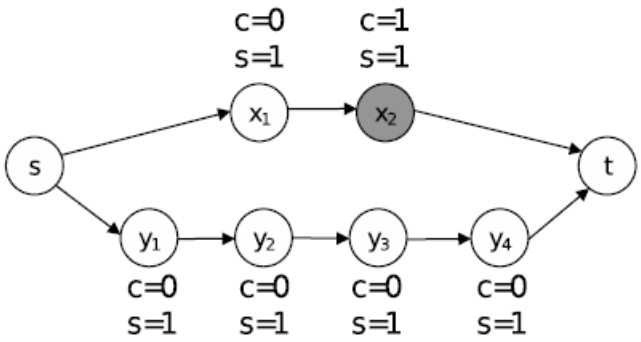


Figura 20: Um exemplo de roteador maze selecionando um caminho mais curto não permitido quando um caminho legal existe (SO,2007).

Proposição 1: A norma de pelo menos um dos caminhos x na iteração n, ||f(n,x)|| é da ordem de Ω (P^n).

Prova: Desde que o algoritmo esteja rodando na iteração *n*, deve haver pelo menos um caminho curto não permitido no grafo de recurso de roteamento. Assumindo *x* como um dos caminhos com um caminho curto não permitido e que uma alternativa legal de caminho ainda não exista. Deste modo *x* permanece sendo o custo do caminho ótimo. Sendo o *u* o caminho mais curto em *x*, temos:

$$\begin{aligned} \|f(x,n)\| &= \|c(x,n)\| + \|s(x,n)\| \\ &\geq \|c(x,n)\| \\ &\geq \sum_{v \in x} \|occ(v,n)\, p(v,n) h(v,n)\| \\ &\geq \|(1)\, p(u,n) h(u,n)\| \\ &\geq \|p(u,n)\| \\ &\geq P^n \|p(u,0)\| \end{aligned}$$

Provando assim o limite inferior necessário.

Um exemplo gráfico é mostrado na Figura 21 cujas linhas de pontos representam a “plotagem” de vários candidatos a caminho. Assim como o número de iterações aumenta, as “plotagens” dos caminhos “difundem” ao longo do eixo de congestionamento a uma taxa exponencial como mostrado.

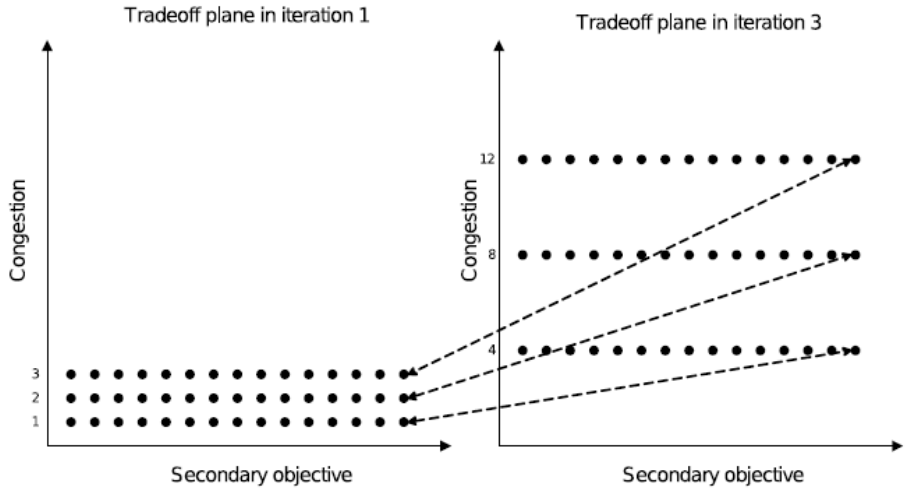


Figura 21: Difusão do plano de *tradeoff* durante a execução do algoritmo supondo *P = 2* e *I = 1*. Setas mostram a correspondência do custo do caminho (SO,2007).

A taxa de crescimento relativo aos custos de congestionamento sobre os custos estáticos secundários também pode causar estabilidade ao problema. Considerando R_i a norma relativa entre o congestionamento e as componentes do custo secundário do caminho na iteração *i*.

$$R_i = \frac{\|c(x,1)\|}{\|s(x,1)\|} \qquad (6)$$

Pode-se mostrar que R_i é também exponencial para i com argumentos similares com os usado na Proposição 1. Para qualquer representação de ponto flutuante, há um limite de absorção dependente *A* no comprimento do significando, além disso, cada adição por 1 é idempotente. Isso é porque o tamanho relativo de 1 para *A* é suficientemente pequeno para que o significando seja exaurido com zeros à esquerda quando é normalizado para 1 para a adição com *A*.

Se $R_i > A$, casos patológicos (comportamento não aceitável) do grafo de busca podem ocorrer como mostrado na Figura 22. Nesta figura, dois caminhos

candidatos estão disponíveis com ambos os caminhos contendo caminho curto não permitido. Nesse casso, a busca deve escolher o caminho com menor custo secundário. Devido à absorção e a expansão ordenada de vértices com custo iguais em uma busca ótima, o menor caminho com um grande custo secundário é escolhido.

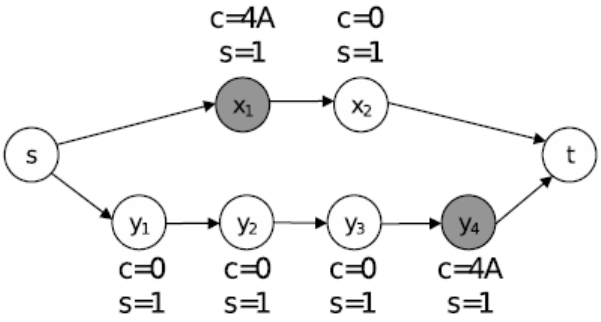


Figura 22: Um exemplo de um roteador *maze* selecionando um caminho de custo secundário sub-ótimo por causa da absorção do ponto flutuante (SO,2007).

Os limites de iteração devido à absorção de ponto flutuante e a representação de limites são mostradas na Tabela 2. Pode-se observar que os limites de absorção são rapidamente alcançados mesmo para as mais detalhadas representações de ponto flutuante. É importante ressaltar que na prática a representação do limite pode ser significantemente menor em instancias congestionadas devido a componente de congestionamento histórica.

Numeric type	`int`	`float`	`double`	`long double`
Absorption threshold, A	–	1.68×10^{7}	9.00×10^{15}	1.84×10^{19}
Corresponding iteration	–	24	53	64
Maximum value	2.15×10^{9}	3.40×10^{38}	1.80×10^{308}	1.19×10^{4932}
Corresponding iteration	31	128	1024	16384

Tabela 2: Absorção e limites de representação de iterações com *I=1* e *P=2*. (Máquina/Compilador: Intel Xeon/GCC3)

Absorção de ponto flutuante pode, em geral, ser endereçado pelo uso racional de pacotes matemáticos, que podem ser usados mais de uma palavra de memória para armazenar cada número. Entretanto, com argumentos similares como os mostrados acima, o número de membros ou palavras usadas para representar

cada caminho candidato será também incrementado linearmente com o número de iterações.

3.2.3 Avaliação do caminho com prioridade de pares

3.2.3.1 Definição e propriedades

Instabilidade numérica ocorre devido à projeção inerente de duas métricas distintas em uma medida escalar. Entretanto, propõe-se que custo de congestionamento e custo secundário de caminhos candidatos podem ser armazenados separadamente. A comparação pode então ser baseada em ambas as métricas.

Definição 1: O custo do caminho ou vértice x expressado como um par prioritário *PP(x)* é um vetor de duas dimensões de seu custo de congestionamento e seu custo secundário, denotado como *PP(x) = <c(x),s(x)>*.

Definição 2: Adição de par prioritário (+) é definido como uma soma de vetores.

$$PP(x) + PP(y) = < c(x) + cy, s(x) + s(y) >$$

Definição 3: Comparação entre pares prioritários ( ) é definido como um ordenamento lexicográfico de congestionamento e custos secundários como se segue:

$$\langle c(x), s(x) \rangle \lessdot \langle c(y), s(y) \rangle$$

se quiser,

$$c(x) < c(y),$$

ou ambos

$$c(x) = c(y) \text{ and } s(x) < s(y)$$

.

Deste modo, uma busca *A** usando avaliação por prioridade de pares deve definir as funções necessárias f' (prioridade), g' (custo do caminho até o momento) e $\hat{f}'$ (heurística para o destino) como as equações 7 a 9. $\hat{f}(x)$ é uma heurística admissível como a usada no caso escalar. O *min-heap* dos valores f' são mantidos com comparação de prioridade de pares ($\preceq$).

$$f'(x) = g'(x) + \hat{f}'(x) \quad (7)$$
$$g'(x) = \langle c(x), s(x) \rangle \quad (8)$$
$$\hat{f}'(x) = \langle 0, \hat{f}(x) \rangle \quad (9)$$

Pode-se mostrar que a busca *A** é ótima sob o operador de comparação $\preceq$ demonstrando que a heurística $\hat{f}'$ é admissível se a heurística escalar subjacente $\hat{f}$ é igualmente admissível (HART, 1968).

Proposição 2: Sendo $\hat{f}(x)$ uma heurística admissível em avaliação de espaço escalar. Então $\hat{f}'(x)$ é uma heurística admissível em avaliação de espaço para pares prioritários sobre o operador $\preceq$.

Prova: Desde de que $\hat{f}$ seja admissível em espaço escalar, então ele é o limite inferior do custo para o destino. O custo atual para conclusão de um a(x) pode ter algum desvio não negativo do limite inferior, ϵ, no custo secundário, isto é:

$$a(x) = \hat{f}(x) + \delta + \epsilon$$

Permitindo a'(x) ser o custo atual de conclusão na avaliação do espaço para pares prioritários. Tem-se:

$$a'(x) = \langle \delta, \hat{f}(x) + \epsilon \rangle$$

Desde de que

$$\hat{f}'(x) = \langle 0, \hat{f}(x) \rangle \prec \langle \delta, \hat{f}(x) + \epsilon \rangle$$

para qualquer δ ou ϵ positivo, prova-se a admissibilidade necessária.

A Figura 23 mostra as regiões inviáveis e sub-ótimas definidas na seção anterior, para a avaliação do espaço para pares prioritários. Ao contrário do espaço de busca escalar, não há curva *Pareto* e apenas caminhos com custo de pares prioritários iguais podem ser possibilidades alternativas.

Nota-se duas propriedades úteis deste espaço de pesquisa no lema seguinte. Esses dois lemas garantem caminhos preferidos. Pode-se ver exemplos nas figuras 20 e 22.

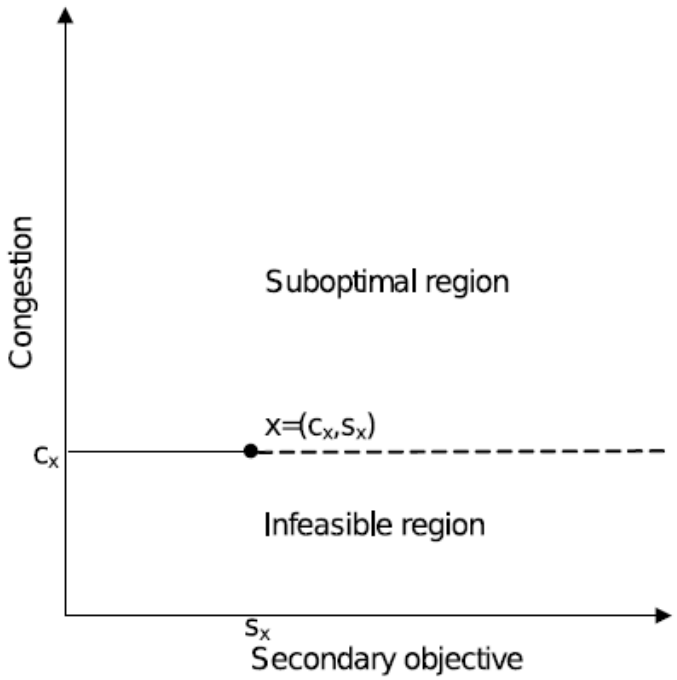


Figura 23: Regiões inviáveis e sub-ótimas implícitas por uma pesquisa gráfica na avaliação do espaço por prioridade de pares (SO,2007).

Lema 1: Se há uma árvore de congestionamento de caminhos *l,* a busca gráfica irá sempre encontrá-lo.

Lema 2: Se há alguns candidatos a caminhos com o mesmo custo de congestionamento, a busca através grafo irá encontrar um caminho com o mínimo custo secundário.

Além disso, como o custo dos congestionamentos é otimizado de forma independente do custo secundário, já não é necessário dimensionar o custo de congestionamento exponencialmente conforme o caso escalar. Pode-se permitir que a negociação de congestionamento ocorra pelo simples diferenciamento entre vértices que são repetidamente usados por aqueles que não o foram.

3.2.3.2 Um Roteador Wirelength com Base em Prioridade de Pares

No roteador CornNC é utilizado o método padrão iterativo de "ripar" e re-rotear todas as *nets*. O custo de congestionamento para cada iteração é definida pelo seguinte cronograma, com o custo de congestionamento de um vértice incrementado se for muito utilizado em iterações prévias.

$$c'(v,0) = 1 \tag{10}$$

$$c'(v,i+1) = \begin{cases} c'(v,i) & \text{if } \mathrm{occ}(v,i) \leq \mathrm{cap}(v) \\ c'(v,i)+1 & \text{otherwise} \end{cases} \tag{11}$$

O custo secundário W L (x) de um caminho x é equivalente ao número de segmentos de fio usados no caminho. Uma heurística $\widehat{WL}(x)$ admissível para custo completo do caminho pode ser construído assumindo que o caminho pode ser completado com fios das distâncias mias curtas *L1* até o destino, e o custo da função de busca *A** definidas deste modo como:

$$f'(x) = g'(x) + \hat{f}'(x) \tag{12}$$

$$g'(x) = \langle \sum_{v \in x} \mathrm{occ}(v) c'(v), WL(x) \rangle \tag{13}$$

$$\hat{f}'(x) = \langle 0, \widehat{WL}(x) \rangle \tag{14}$$

O custo secundário nesta aplicação tem um limite conhecido igual ao número de vértices no grafo de roteamento, e ambos os custos de congestionamento e secundário são discretos. Entretanto, pode-se compactar os dois componentes em um par em um único valor inteiro, através do deslocamento do custo de congestionamento pelo número suficiente de bits. A função de comparação de pares prioritários é equivalente a comparação padrão < nessa representação. A otimização permite reduzir a memória usada e pode acelerar o algoritmo em algumas máquinas.

A norma da função de prioridade $||f'(x)||$ é relacionada com a norma do custo de congestionamento $||c'(x)||$ como o custo secundário é limitado pela constante. Na iteração *n*, a norma é da ordem *O(occ(x,n)n)* que é linear para o produto do número de iterações e a ocupação do caminho.

Para *nets* multi-pinos, acham-se pinos de conexão em ordem decrescente da distância de *L1* conseguindo o mais rápido comportamento de convergência. Isso pode ocorrer devido à atribuição de grandes galhos sendo ortogonal a atribuição corrente de galhos curtos, e assim mais estável entre iterações. A escolha da heurística da ordem dos pinos merece uma melhor investigação. O pseudocódigo para este roteador é mostrado na Figura 24.

```
CornNC Wirelength Router
 1  for iteration ← 1 to M
 2       do for each net with source
 3             do Rip up old routing tree of net
 4                for each target in decreasing Manhattan distance
 5                      do Add partial routing tree elements to search space with cost ⟨0, 0⟩
 6                         Find min-cost path from source to target in search space
 7                         Save found path
 8                Save entire tree as new routing tree of net
 9          if routing is legal
10            then Report success with solution
11            else  Update congestion cost
12  Report failure
```

Figura 24: Pseudocódgio para o Roteador Wirelenght CornNC (SO,2007).

3.2.4 Resultados

O algoritmo CornNC foi capaz de encontrar soluções de roteamento em 16 dos 20 benchmark de posicionamento com menor quantidade de trilhas que seus

antecessores, reduzindo para 18, que é uma melhoria de 10,2% sobre o melhor total anterior. Na Tabela 3 podemos verificar os resultados do CornNC em relação a outros algoritmos.

Circuit	Size	SEGA[12]	VPR3.99[2]	SC-Pathfinder[4]	VPR 4.30[1]	CornNC	Track Reduction
alu4	1522	16	10	9	9	8	1
apex2	1878	20	11	11	10	9	1
apex4	1262	19	12	11	11	10	1
bigkey	1707	9	7	6	6	5	1
clma	8383	25	12	12	10	9	1
des	1591	11	7	7	7	6	1
diffeq	1497	10	7	7	7	6	1
dsip	1370	9	7	6	5	5	0
elliptic	3604	16	10	9	9	8	1
ex1010	4598	22	10	10	9	8	1
ex5p	1064	16	13	12	11	10	1
frisc	3556	18	11	11	11	10	1
misex3	1397	17	10	10	10	9	1
pdc	4575	33	16	16	15	13	2
s298	1931	18	7	7	6	6	0
s38417	6406	10	8	7	6	6	0
s38584.1	6447	12	9	8	7	7	0
seq	1750	18	11	11	10	9	1
spla	3690	26	13	12	12	10	2
tseng	1407	9	6	6	6	5	1
Total		334	197	188	177	159	18(10.2%)

Tabela 3: Resultado comparativos do CornNC em cima dos benchamrk MCNC quanto a quantidade de Trilhas (SO,2007).

3.3 Metaheurísticas

As metaheurísticas são procedimentos destinados a encontrar uma boa solução, eventualmente a ótima, consistindo na aplicação, em cada passo, de uma heurística subordinada, a qual tem que ser modelada para cada problema específico (RIBEIRO, 1996).

Contrariamente às heurísticas convencionais, as metaheurísticas são de caráter geral e providas de mecanismos para tentar escapar de ótimos locais ainda distantes dos ótimos globais.

As metaheurísticas diferenciam-se entre si basicamente pelo mecanismo usado para sair das armadilhas dos ótimos locais. Elas se dividem em duas categorias, de acordo com o princípio usado para explorar o espaço de soluções: busca local e busca populacional.

Nas metaheurísticas baseadas em busca local, a exploração do espaço de soluções é feita por meio de movimentos, os quais são aplicados a cada passo sobre a solução corrente, gerando outra solução promissora em sua vizinhança. Iterated Local Search é um exemplo de método que se enquadram nesta categoria.

Os métodos baseados em busca populacional, por sua vez, consistem em manter um conjunto de boas soluções e combiná-las de forma a tentar produzir soluções ainda melhores. Exemplos clássicos de procedimentos desta categoria são os Algoritmos Genéticos, os Algoritmos Meméticos e o Algoritmo Colônia de Formigas que não são abordados no presente trabalho.

3.3.1 GRASP

A metaheurística Greedy Randomized Adaptive Search Procedure (GRASP) foi inicialmente desenvolvida por Feo e Resende (FEO, 1995) e consiste em um procedimento iterativo de duas etapas, a saber: (i) fase de construção, na qual uma solução viável é gerada; (ii) fase de busca local, cuja vizinhança da solução construída é pesquisada até que se encontre um ótimo local. A Figura 25 mostra o pseudocódigo do procedimento GRASP, onde o algoritmo realiza "Max_iteracoes" iterações e "semente" é a semente inicial utilizada para gerar um número de forma pseudo-aleatória (SUBRAMANIAN, 2007).

```
Procedimento GRASP(Max_iteracoes, semente, α)
1.  CarregarDados( );
2.  para k = 1,..., Max_iteracoes faça
3.      Solucao ← Construção_Gulosa_Aleatoria(α, semente);
4.      Solução ← Busca_Local(Solucao);
5.      Atualizar_Solucao(Solucao, Melhor_Solucao);
6.  fim;
7.  retornar Melhor_Solucao;
Fim GRASP
```

Figura 25: Procedimento GRASP (RESENDE, 2005).

Na fase de contrução, uma solução é iterativamente construída, elemento por elemento. A cada iteração desta fase, os próximos elementos candidatos a serem incluídos na solução são colocados em uma lista C de candidatos, seguindo um critério de ordenação pré-determinado. Este processo de seleção é baseado em uma função adaptativa gulosa g : C → R, que estima o benefício da seleção de cada um dos elementos. A heurística é dita adaptativa porque os benefícios associados com a escolha de cada elemento são atualizados em cada iteração da fase de

construção para refletir as mudanças oriundas da seleção do elemento anterior. A componente probabilística do procedimento reside no fato de que cada elemento é selecionado de forma aleatória a partir de um subconjunto restrito formado pelos melhores elementos que compõem a lista de candidatos. Este subconjunto recebe o nome de lista de candidatos restrita (LCR). Esta técnica de escolha permite que diferentes soluções sejam geradas em cada iteração GRASP. O pseudocódigo representado pela Figura 26, onde α ϵ [0; 1] é um parâmetro do método, descreve a fase de construção GRASP.

Observamos que o parâmetro α controla o nível de gulosidade e aleatoriedade do procedimento “Construcao”. Um valor α = 0 faz gerar soluções puramente gulosas, enquanto α = 1 faz produzir soluções totalmente aleatórias.

procedimento $Construcao(g(.), \alpha, s)$;
1 $s \leftarrow \emptyset$;
2 Inicialize o conjunto C de candidatos;
3 enquanto $(C \neq \emptyset)$ faça
4 $g(t_{min}) = \min\{g(t) \mid t \in C\}$;
5 $g(t_{max}) = \max\{g(t) \mid t \in C\}$;
6 $LCR = \{t \in C \mid g(t) \leq g(t_{min}) + \alpha(g(t_{max}) - g(t_{min}))\}$;
7 Selecione, aleatoriamente, um elemento $t \in LCR$;
8 $s \leftarrow s \cup \{t\}$;
9 Atualize o conjunto C de candidatos;
10 fim-enquanto;
11 Retorne s;
fim $Construcao$;

Figura 26: Procedimento de construção do GRASP (RESENDE, 2005).

Assim como em muitas técnicas determinísticas, as soluções geradas pela fase de construção do GRASP provavelmente não são localmente ótimas com respeito à definição de vizinhança adotada. Daí a importância da fase de busca local, a qual objetiva melhorar a solução construída. A Figura 27 descreve o pseudo-código de um procedimento básico de busca local com respeito a uma certa vizinhança N(.) de *s* para um problema de minimização.

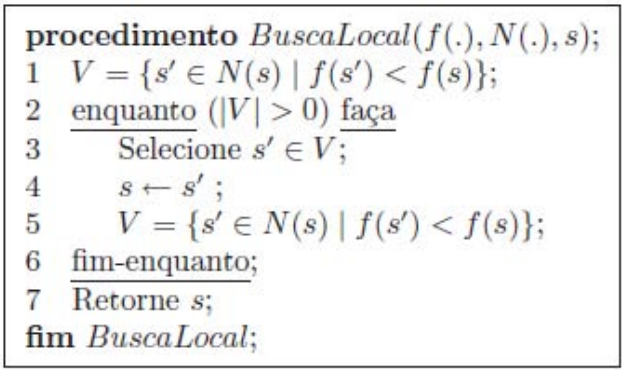


Figura 27: Procedimento de busca local do GRASP (RESENDE, 2005).

A eficiência da busca local depende da qualidade da solução construída. O procedimento de construção tem então um papel importante na busca local, uma vez que as soluções construídas constituem bons pontos de partida para a busca local, permitindo assim acelerá-la.

O parâmetro α, que determina o tamanho da lista de candidatos restrita, é basicamente o único parâmetro a ser ajustado na implementação de um procedimento GRASP.

Em (FEO, 1995) discute-se o efeito do valor de α na qualidade da solução e na diversidade das soluções geradas durante a fase de construção. Valores de α que levam a uma lista de candidatos restrita de tamanho muito limitado (ou seja, valor de α próximo da escolha gulosa) implicam em soluções finais de qualidade muito próxima àquela obtida de forma puramente gulosa, obtidas com um baixo esforço computacional. Em contrapartida, provocam uma baixa diversidade de soluções construídas. Já uma escolha de α próxima da seleção puramente aleatória leva a uma grande diversidade de soluções construídas, por outro lado, muitas das soluções construídas são de qualidade inferior, tornando mais lento o processo de busca local.

O procedimento GRASP procura, portanto, conjugar bons aspectos dos algoritmos puramente gulosos, com aqueles dos procedimentos aleatórios de construção de soluções.

Procedimentos GRASP mais sofisticados incluem estratégias adaptativas para o parâmetro α. O ajuste deste parâmetro ao longo das iterações GRASP, por critérios que levam em consideração os resultados obtidos nas iterações anteriores, produz soluções melhores do que aquelas obtidas considerando-o fixo (PRAIS, 1999, 2000, 2000).

3.3.2 ILS

Suponha que uma solução ótima local tenha sido encontrada por um algoritmo de busca local. Ao invés de reiniciar o mesmo procedimento a partir de uma solução completamente nova, a metaheurística ILS aplica repetidamente uma busca local às soluções iniciais obtidas através de perturbações das soluções ótimas locais previamente visitadas (STÜTZLE, 1998).

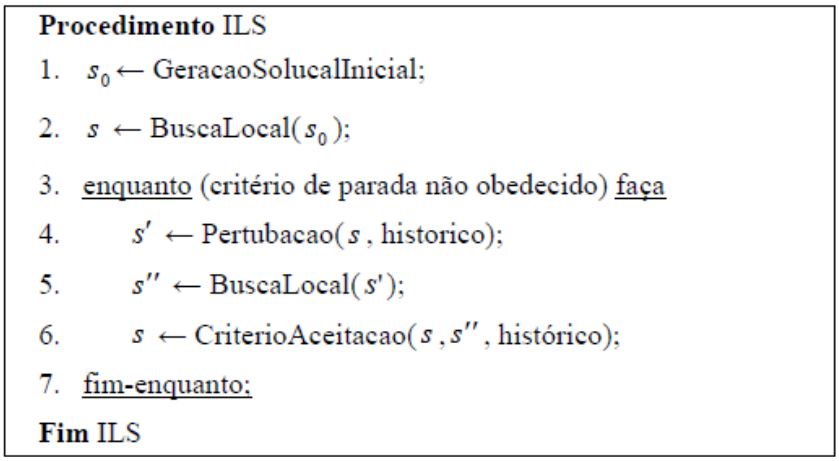


Figura 28: Procedimento ILS (LOURENÇO, 2002).

Segundo Lourenço et al. (2002), a idéia essencial da ILS está atribuída ao fato desta metaheurística realçar seu foco em um subespaço menor, ao invés de considerar o espaço completo de soluções, definido por aquelas que são ótimos locais de um dado procedimento de otimização.

Para se aplicar a metaheurística ILS, quatro procedimentos devem ser especificados, conforme ilustra a Figura 28. São eles:

1. “GeracaoSolucaoInicial”, no qual uma solução inicial é construída;
2. “BuscaLocal”, que refina a solução inicialmente obtida;
3. “Perturbacao”, onde um novo ponto de partida é gerado, por meio de uma perturbação de uma solução encontrada na busca local;
4. “CriterioAceitacao”, que determina a partir de qual solução deve-se prosseguir a busca.

Uma ilustração gráfica do procedimento ILS pode ser observada na Figura 29.

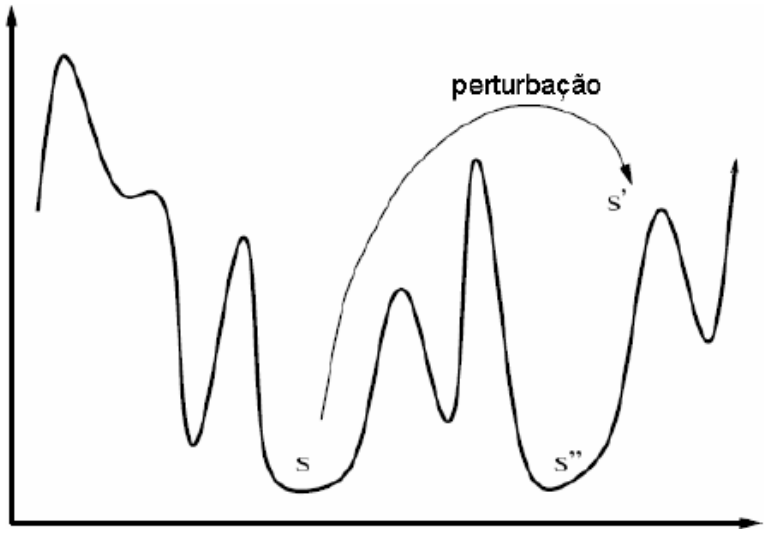


Figura 29: Gráfico do procedimento ILS (LOURENÇO, 2002).

De acordo com Stützle (1998), a modificação efetuada na etapa de perturbação é empregada com o objetivo de escapar de uma solução ótima local ainda distante de um ótimo global. Freqüentemente, o movimento é escolhido aleatoriamente dentro de uma vizinhança de maior ordem em relação à utilizada no procedimento de busca local, ou um movimento que o algoritmo de busca local não possa reverter em apenas um passo.

A princípio, qualquer método de busca local pode ser utilizado, porém seu desempenho, no tocante à qualidade da solução e esforço computacional, depende fortemente do algoritmo escolhido. Geralmente uma heurística de descida (em problemas de minimização) é usada, contudo a adoção de um procedimento mais complexo também é admissível.

O critério de aceitação é utilizado para decidir qual a próxima solução a ser perturbada. A escolha deste critério é importante, haja vista que controla o balanço entre a intensificação e diversificação. O histórico de busca é usado para decidir se alguma solução ótima local, encontrada anteriormente, deve ser escolhida ou se exerce algum tipo de influência na decisão entre s e s''.

O procedimento ILS deve conduzir a boas amostragens do espaço de busca contanto que as perturbações não sejam nem tão grandes nem demasiadamente pequenas. Se forem pequenas, poucas novas soluções serão exploradas, ao passo que se forem muito grandes, o algoritmo tenderá adotar pontos de partidas praticamente aleatórios.

Capítulo 4 - VPR

O Versatile Place and Route (VPR) é uma ferramenta de CAD para alocação e roteamento de FPGA's, publicado por Vaughn Betz e Jonathan Rose no ano de 1997. Em 2009, é lançada a versão 5.0 do VPR e vem suprir a necessidade de aceitar como entrada os tipos de arquiteturas de FPGAS modernas desenvolvidas ao longo dos anos. O VPR passou a ser um conjunto de ferramentas que incluem quatro novos significantes recursos (LUU, 2009):

- Suporta uma gama de arquiteturas *single-drive* (XILINX, 2008, ALTERA, 2008 e LEMIEUX, 2004).
- Aceita arquiteturas heterogêneas, ou seja, que possua *hard-blocks.*
- Provê modelos elétricos otimizados de uma grande gama de arquiteturas em diferentes processos tecnológicos, incluindo negociação de a*rea-delay* para cada arquitetura.
- Um conjunto de testes de regressão para checar funcionalidades e a qualidade dos resultados das saídas das ferramentas.

4.1 Visão Geral

Na Figura 30 podemos observar o fluxo de projeto da ferramenta VPR. As entradas do VPR consistem em tecnologia mapeada de uma *netlist* e um arquivo *XML* descrevendo a arquitetura do FPGA. O VPR pode alocar o circuito ou usar uma alocação pré-existente. Além disso, tem a capacidade de realizar tanto roteamento globlal ou roteamento combinado (global/detalhado) da alocação. As saídas do VPR consistem tanto na alocação quanto no roteamento, assim como as estatísticas úteis para avaliar arquiteturas de FPGAS, assim como *routed wirelength*, *track count* e *maximum net length*. Alguns dos parâmetros arquiteturais podem ser especificados no arquivo de descrição de arquitetura sendo eles:

- O número de entradas e saídas dos blocos lógicos.
- O lado(s) do bloco lógico em que cada entrada e saída é acessível.

- A equivalência lógica entre vários pinos de entrada e saída (ex.: todas as entradas de LUT são funcionalmente equivalentes).
- O número de pads I/O que caibam em uma linha ou coluna de um FPGA.
- As dimensões do vetor de blocos lógicos (ex.: 23 x 30 blocos lógicos).

Além disso, se o roteamento global for realizado, pode-se também especificar:

- A largura relativa dos canais horizontais e verticais.
- A largura relativa dos canais em diferentes regiões do FPGA.

Finalmente, se combinado roteamento global com roteamento detalhado, pode-se também especificar:

- A arquitetura do *switch block* (i.e. como as trilhas de roteamento são interconectadas),
- O número de trilhas que cada pino de entrada do bloco lógico conecta.
- O valor *Fc* da saída do bloco lógico.
- O valor *Fc* dos pinos I/O.

VPR usa a descrição da arquitetura para criar um grafo de roteamento. Todas as trilhas e todos os pinos na arquitetura se tornam um nó no grafo e as bordas do grafo representam conexões permissíveis. O roteador, a visualização de grafos e as rotinas de computação das estatísticas, todas trabalham apenas com o recurso gráfico de roteamento.

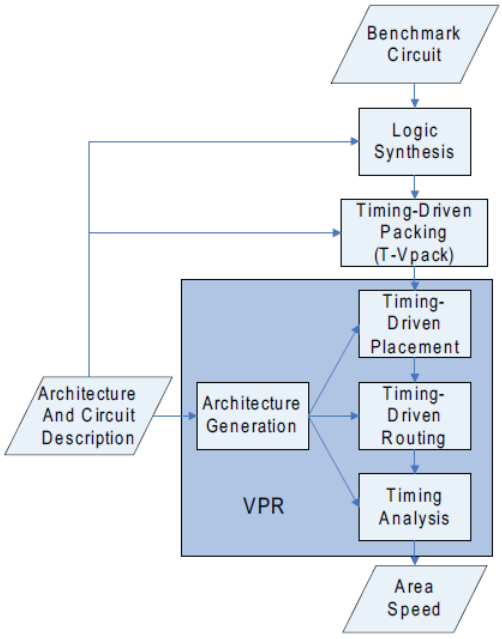


Figura 30: Fluxo CAD (LUU, 2009).

Apesar do VPR ser inicialmente desenvolvido para FPGA´s *island-style* (XILINX, 1994 e AT & T, 1994), ele pode também ser usado com FPGA´s *row-based* (ACTEL, 1994). Na versão 5.0 do VPR ele passou a ser capaz de rotear FPGAs hierárquicas, heterogêneas, e arquiteturas *single-driver*. Por último, os gráficos embutidos no VPR permitem visualização interativa da alocação, do roteamento, dos recursos de roteamento disponíveis e os caminhos possíveis de interconexão (Figuras 13 e 14).

4.2 Algoritmo de Posicionamento

No VPR, um FPGA é modelado como um conjunto de *slots*, ou localizações discretas, em que são posicionados os blocos lógicos ou os blocos de entrada/saída. Objetivando a versatilidade proposta pelo projeto, um arquivo de descrição de arquitetura FPGA especifica:

- A quantidade de pinos de entrada e saída dos blocos lógicos.
- A quantidade de blocos de I/O que cabem em uma linha ou coluna do FPGA.
- A largura relativa dos vários canais de roteamento que cruzam o FPGA.

As dimensões da matriz de blocos lógicos pode ser definida através do

comando de entrada. Caso não seja, o VPR define-a como a menor matriz de blocos lógicos capaz de comportar todos os blocos lógicos do circuito. Esta matriz pode ser especificada como uma matriz quadrada de tamanho N x N, onde:

$$N = \lceil \sqrt{total\_CLB} \rceil$$

Obs.: total_CLB representa a quantidade total de blocos lógicos do circuito.

Na Figura 31, por exemplo, pode-se tirar as seguintes informações: dois canais de entrada/saída podem ocupar uma das posições periféricas referentes a cada uma das linhas ou colunas do FPGA; a matriz de blocos lógicos definida possui três linhas e quatro colunas; e os canais de interligação entre os blocos lógicos são mais largos que os mesmos canais que interligam os blocos lógicos e os blocos de entrada/saída. Deve-se ressaltar que por padrão, os blocos de entrada/saída devem ocupar obrigatoriamente as bordas do FPGA, visto que farão as conexões com o meio externo, dos circuitos implementados no FPGA (LIMA, 2008).

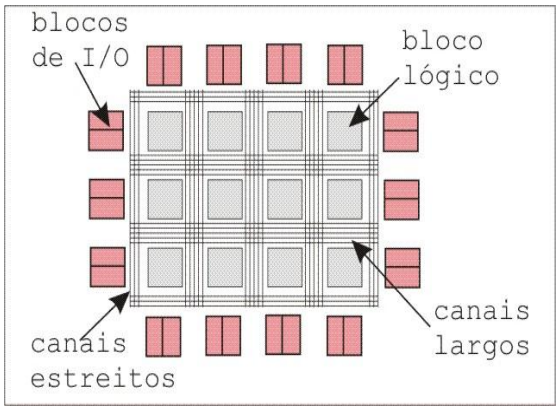


Figura 31: Modelo de FPGA usado no VPR

Durante o posicionamento, o usuário pode deixar o VPR escolher as melhores localizações para os blocos de I/O, ou pode definir posições fixas para estes blocos, através do arquivo de entrada. Esta capacidade permite à ferramenta medir o efeito da atribuição de pinos aleatóriamente em diferentes arquiteturas de FPGAs. Pinos de I/O fixos são bastante comuns em projetos de FPGAs reais em que, a placa do circuito em que o FPGA será montado, é muitas vezes produzida antes do projeto do circuito FPGA ser completado. Nesses casos, a ferramenta de projeto deve manter os pinos de I/O nas posições determinadas pelo projeto na placa do circuito (LIMA, 2008).

4.2.1 Annealing Schedule adaptativo

Uma boa estratégia de *annealing* é essencial para obter soluções de alta qualidade em um tempo de computação aceitável com *simulated annealing*. Lembrando que o *annealing schedule* especifica:

- A Quantidade de tentativas de movimentos por temperatura;
- Como varia a temperatura durante o recozimento;
- Quando o recozimento deve terminar.

Como o objetivo do VPR é poder trabalhar com as mais diversas arquiteturas FPGAs, que usam funções de custo diferentes, bem como uma grande variedade de circuitos de diversas faixas de tamanhos, o VPR faz uso de um recurso de adaptação automática do *annealing schedule* de acordo com o problema de posicionamento especificado. Tal providência se faz necessária, visto que *annealing schedule* fixo não produz bons resultados com qualquer arquitetura (LIMA, 2008).

4.3 Algoritmo de Roteamento

O roteador do VPR é baseado no algoritmo *pathfinder negotiated congestion* (EBELING, 1995 e LEE, 1961). Basicamente, este algoritmo inicialmente roteia cada net pelo seu caminho mais curto, se poder achar, resguardado de qualquer reuso de segmentos de fios ou pinos de blocos lógicos que podem resultar. Uma iteração do roteador consiste de seqüencialmente *ripping-up* e re-rotear (pelo menor custo de caminho achado) todas as net's do circuito. O custo de usar um recurso de roteamento é uma função do corrente reuso de recursos e qualquer reuso que ocorrer em prévias iterações de roteamento. Através do aumento gradual do custo do reuso de recursos de roteamento, o algoritmo força as *nets* com rotas alternativas a evitarem usar os recursos reusados, deixando apenas a *net* que necessita mais de um determinado recurso subjacente.

O algoritmo *Pathfinder* original e o roteador VPR usam o algoritmo de *Dijkstra* (*maze router*) (LEE, 1961) para conectar cada net. Para uma *net* terminal *k*, o *maze router* é invocado *k-1* vezes para realizar todas as conexões requeridas. Na sua primeira chamada, o roteamento *maze* se expande do *source* da *net* até atingir

qualquer um dos *k-1 net sinks*. O caminho do *source* para o *sink* é agora a primeira parte do roteamento desta *net*. A expansão do roteamento *maze* é esvaziado e uma nova expansão é iniciada a partir de todo o roteamento encontrado até o momento. Depois de *k-1* invocações do roteador *maze*, todos os *k* terminais da *net* estarão conectados.

Esta abordagem requer considerável processamento para *nets* de *fanout* alto. *Nets* com alto *fanout* geralmente abrangem a maior parte ou a totalidade dos FPGA's. Portanto, em invocações tardias do roteador *maze* o roteamento parcial usado como o *source* da *net* será bem grande, e irá tomar um longo tempo para expandir o roteador *maze* até o próximo *sink*.

Pode-se encontrar métodos mais eficientes. Quando uma *net* é alcançada, adiciona-se todos os segmentos de recursos roteados necessários para conectar com o *sink* e o roteamento parcial atual para expansão com um custo de zero. Não esvaziando a expansão do roteamento *maze* atual, apenas continua expandindo normalmente. Desde que o novo caminho adicionado para o roteamento parcial tenha custo zero, o roteador *maze* requer relativamente pouco tempo para adicionar essa nova expansão e o próximo *sink* será alcançado muito mais rapidamente que se a expansão inteira tivesse sido começado a partir do zero. Figura 32 ilustra a diferença graficamente.

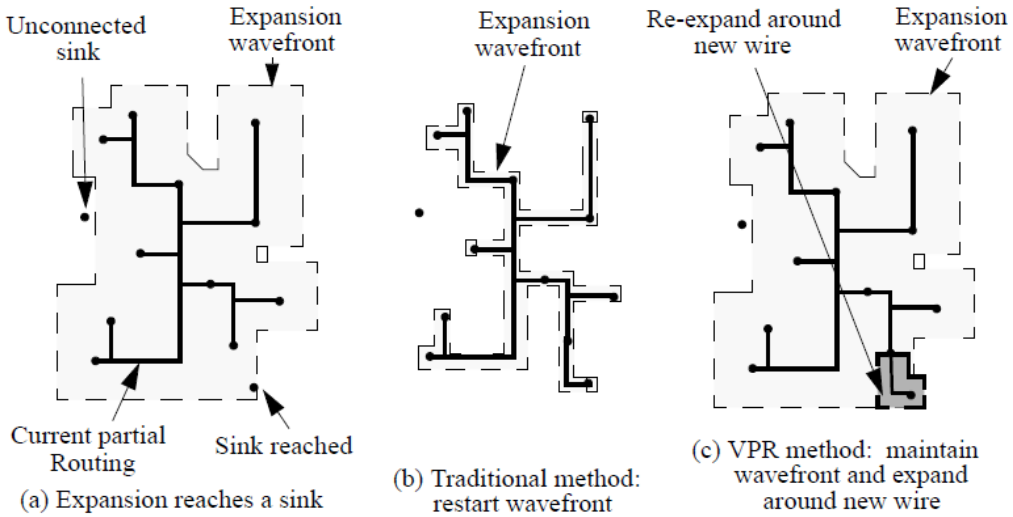


Figura 32: Quando um *sink* é atingido (a), uma nova expansão pode ser construída do zero (b), ou incrementa-lo (c). (BETZ, 1997)

Capítulo 5 – Algoritmo Proposto

O algoritmo proposto consiste em um procedimento de busca adaptativo, guloso e randomizado (Greedy Randomized Adaptive Search Procedure - GRASP) que utiliza o método de *rip-up* com ordenação seletiva e re-roteamento. Para realização da busca local, usamos *rip-up* e re-roteamento baseado na metaheurística ILS.

5.1 Algoritmo Proposto

O algoritmo possui duas etapas: a primeira corresponde à construção gulosa randomizada e a segunda é relativa à fase de refinamento, em que se utiliza a metaheurística ILS. O pseudocódigo encontra-se ilustrado na Figura 33, onde s* representar a melhor solução e α um parâmetro tratado em detalhes na subseção 3.3.1.

```
Procedimento GRASP/ILS(Max_iteracoes, semente, α)
1. CarregarDados( );
2. para k = 1,..., Max_Grasp_iteracoes faça
3.     s ← Construção_Gulosa_Aleatoria(semente, α);
4.     s ← ILS(N(.), f(.), s);
5.     Atualizar_Solucao(s, s*);
6. fim;
7. retornar s*;
Fim GRASP/ILS
```

Figura 33: Procedimento GRASP/ILS

5.1.1 Procedimento construtivo

O procedimento construtivo consiste em rotear as *nets* através da LCR (Lista de Candidatos Restrita), diferentemente do VPR, no qual as *nets* são roteadas na ordem em que são dispostas no arquivo de entrada de *netlist*, as *nets* são ordenadas pelo critério do seu tempo de atraso. Também foi testado outro

parâmetro para construir a LCR: o critério das *nets* como o maior número de terminações, cujos resultados não foram promissores.

Após a seleção da net na LCR, o roteamento dos pares *source-sink* de cada *net* são realizados através do procedimento de roteamento *timing-driven* encontrado no VPR 5.0 (FUU, 2008). Assim que todos os pares forem roteados, uma segunda *net* é selecionada na LCR e seus pares *source-sink* são roteados e assim sucessivamente para todas as *nets*. Caso as *nets,* no final deste procedimento, estiverem compartilhando algum recurso em comum, significa que o roteamento ainda não é viável. Os recursos compartilhados são penalizados e as *nets* são novamente roteadas até um máximo de *max_Iteracoes* tentativas.

```
Procedimento Construção_Gulosa_Aleatoria(semente, α)
1.para i = 1,..., max_tentativas faça
2.   para k = 1,..., max_iteracoes faça
3.      boleano viavel ← TRUE;
4.      para j = 1,..., Numero_de_Nets faça
5.         escolha_Gulosa_da_Net(α)
6.         viavel ← Roteia_net( );
7.         se (!viavel) break;
8.         senão rip-up( );
9.                penaliza-recursos-em-comum( );
10.     fim;
11.     se (viavel) break;
12.  fim;
13.  atualizar_Solucao(s, s*);
14.fim;
15.para i = 1,..., max_tentativas faça
16.   para k = 1,..., max_iteracoes faça
17.      boleano viavel ← TRUE;
18.      para j = 1,..., Numero_de_Nets faça
19.         escolhe_ net_com_maior_delay(s*)
20.         viavel ← Roteia_net( );
21.         se (!viavel) break;
22.         senão rip-up( );
23.               penaliza-recursos-em-comum( );
24.      fim;
25.      se (viavel) break;
26.   fim;
27.   Atualizar_Solucao(s, s*);
28.fim;
29.retornar s*;
Fim Construção_Gulosa_Aleatoria
```

Figura 34: Procedimento Construtivo

De *max_tentativas* de roteamento é retirado o melhor roteamento viável. Foi observado que, ao final do roteamento, tínhamos uma lista de *nets* ordenadas por atraso. Retirar as penalidades dos recursos de roteamento e tentar rotear esta lista novamente mostrou melhoria em vários testes. Observado isso, um procedimento guloso de roteamento *timing-driven* foi adicionado ao algoritmo para rotear esta lista. Nesta etapa, o *max_iteracoes* tem um valor maior que o *max_Iteracoes* da etapa de roteamento com LCR. Ao se obter um roteamento viável, caso este roteamento seja melhor, ele passa ser a melhor soluação obtida, caso não seja, sua lista de *nets,* no final do roteamento, é ordenado por atraso e utilizado para uma nova tentativa de roteamento. Este processo é repetido *max_tentativas* vezes.

5.1.2 Procedimento de Busca Local (ILS)

A fase de busca local, responsável por refinar a solução inicialmente obtida na fase de construção, é efetuada pela metaheurística ILS. Consiste basicamente em re-rotear as *nets* da melhor solução da fase de construção utilizando o mesmo roteador *timing-driven*, porém uma pequena perturbação na lista de *nets* é que causa a convergência a soluções na vizinhança da solução obtida na fase de construção.

Na Figura 35, podemos observar um exemplo de perturbação. Na primeira tabela, temos a lista de *nets* ordenadas por tempo de atraso obtido da melhor solução da fase de construção. A perturbação consiste em colocar a *net* de maior atraso no final da lista de *nets* a serem roteadas. Com a nova lista, tenta-se rotear utilizando o método *timing-driven.* Em caso de não haver melhorias, utilizamos a lista de *nets* da primeira tabela de atraso e causamos uma nova perturbação. Colocamos a segunda *net* com maior atraso para o final da lista e tenta-se rotear novamente.

Lista de Nets ordenada por tempo de atraso.

Net 80	**Net 53**	**Net 60**	**Net 8**	**Net 10**	**Net 3**	**...**	**Net 33**

Primeira Perturbação

Net 53	**Net 60**	**Net 8**	**Net 10**	**Net 3**	**...**	**Net 33**	**Net 80**

Lista de Nets ordenada por tempo de atraso caso não haja melhoria.

Net 80	**Net 53**	**Net 60**	**Net 8**	**Net 10**	**Net 3**	**...**	**Net 33**

Segunda Perturbação

Net 80	**Net 60**	**Net 8**	**Net 10**	**Net 3**	**...**	**Net 33**	**Net 53**

Figura 35: Perturbação sem melhoria

Quando houver melhoria, obtemos uma nova lista de *nets* ordenadas e recomeça o processo de perturbação e re-roteamento sobre esta lista. Sempre que houver uma melhoria, o processo recomeça. Caso não haja melhorias, o processo termina após *ils_iteracoes* iterações.

Lista de Nets ordenada por tempo de atraso.

Net 80	**Net 53**	**Net 60**	**Net 8**	**Net 10**	**Net 3**	**...**	**Net 33**

Primeira Perturbação

Net 53	**Net 60**	**Net 8**	**Net 10**	**Net 3**	**...**	**Net 33**	**Net 80**

Lista de Nets ordenada por tempo de atraso caso haja melhoria.

Net 73	**Net 54**	**Net 7**	**Net 82**	**Net 65**	**Net 62**	**...**	**Net 70**

Segunda Perturbação

Net 54	**Net 7**	**Net 82**	**Net 65**	**Net 62**	**...**	**Net 70**	**Net 73**

Figura 36: Perturbação com melhoria

Na Figura 37, podemos observar o pseudocódigo do procedimento busca local ILS.

```
Procedimento ILS(N(.), f(.), r, s)
1.para i = 1,..., ils_iteracoes faça
2.    para k = 1,..., max_iteracoes faça
3.       boleano viavel ← TRUE;
4.       perturbação( ils_iteracoes );
5.       para j = 1,..., Numero_de_Nets faça
6.          escolhe_ net_com_maior_delay(s*)
7.          viavel ← Roteia_net( );
8.          se (!viavel) break;
9.          senão rip-up( );
10.            penaliza-recursos-em-comum( );
11.      fim;
12.      se (viavel) break;
13.   fim;
14.   atualizar_solucao(s, s*);
15.   se( s==s* ) i = 1;
16.fim;
Fim ILS
```

Figura 37: Procedimento de busca local (ILS)

5.1.3 Critério de Parada

Uma das peculiaridades do caminho crítico nos circuitos roteados para FPGA é que se o roteamento nunca fizer um caminho no eixo contrário a direção à ponto de destino, esse caminho pode ser considerado ótimo.

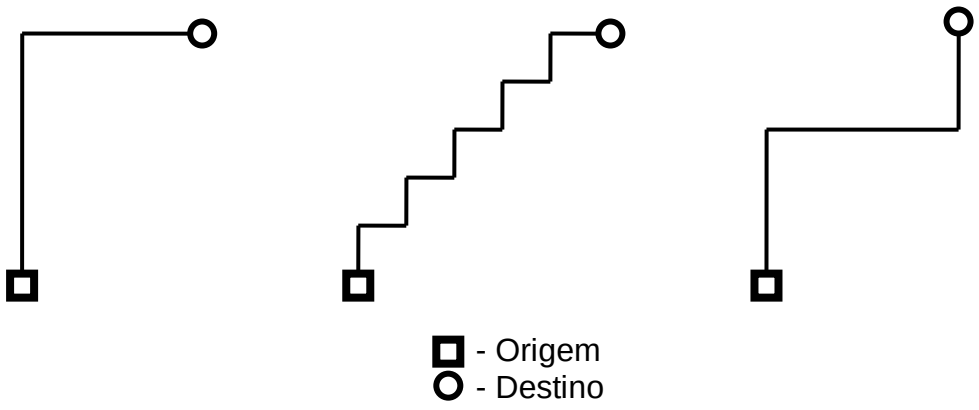


Figura 38: Exemplos de caminhos ótimos

Observado que na primeira iteração do roteador *timing-driven* o roteamento é feito com baixa penalidade nas trilhas, permitindo que os caminhos se sobreponham, eles atingem nesta etapa valores próximos ao ótimo em relação ao caminho crítico. Para ser mais exato, em média, 0,01% pior que o melhor caminho

crítico permitido no circuito, considerando um *placement* específico para o circuito. Tendo este valor em mãos, qualquer roteamento viável obtido durante a execução do algoritmo atingir valores iguais ou menores a este causam o término do programa.

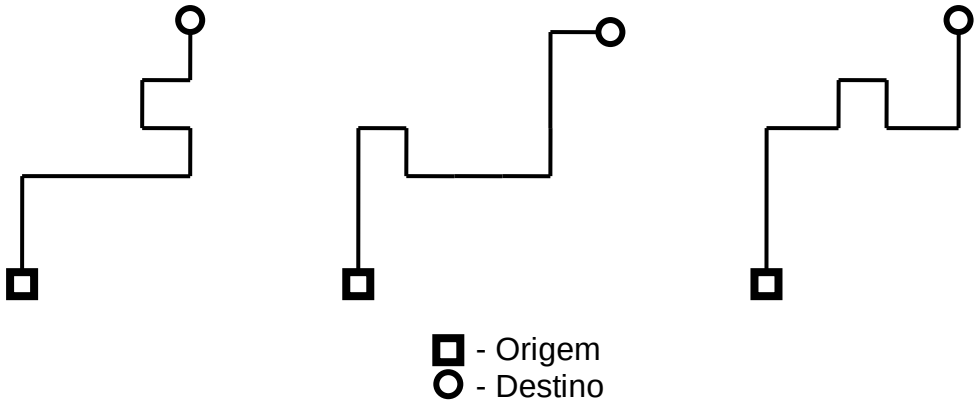


Figura 39: Exemplos de caminhos não ótimos

Capítulo 6 – Resultados

O algoritmo proposto foi implementado na linguagem de programação C utilizando o compilador GCC e foi executado em um PC Intel Core 2 Quad Q8200 2.33 GHz com 1,98GB de memória RAM e sistema operacional Ubuntu 32 bits Professional Edition.

6.1 Benchmark

Os resultados foram obtidos através do roteamento dos 20 circuitos do *benchmark* MCNC, conforme consta na tabela 4, na arquitetura de FPGA do tipo *Island Style,* mapeada tecnologicamente para células compostas por uma LUT-4 (Lookup table de quatro entradas) e um *flip-flop*, conforme definido em (BETZ, 1997).

Circuito	CLBs	I/O	nets	Dimensão do FPGA
clma	8383	144	8445	92x92
s38417	6406	135	6435	81x81
s38584	6447	342	6486	81x81
ex1010	4598	20	4608	68x68
pdc	4575	56	4591	68x68
spla	3690	62	3706	61x61
elliptic	3604	245	3736	61x61
frisc	3556	136	3576	60x60
bigkey	1707	426	1937	54x54
apex2	1878	41	1916	44x44
dsip	1370	426	1600	54x54
seq	1750	76	1791	42x42
s298	1931	10	1935	44x44
diffeq	1497	103	1562	39x39
alu4	1522	22	1536	40x40
misex3	1397	28	1411	38x38
apex4	1262	28	1271	36x36
ex5p	1064	71	1072	33x33
tseng	1047	174	1100	33x33
e64-4lut	274	130	339	17x17

Tabela 4: Circuitos MCNC

Neste trabalho todos os dados experimentais utilizados para alimentar o sistema em estudo são modelos não reais e em todas as simulações realizadas foram considerados os parâmetros *default* do VPR. Para comparação de tempo crítico é verificado o menor tamanho de trilha necessária para rotear os circuitos do *benchmark* MCNC, o VPR foi executado em uma única iteração fixando o número de trilhas para o valor mínimo conhecido.

Circuito	Trilhas	Circuito	Trilhas	Circuito	Trilhas	Circuito	Trilhas
alu4	11	**Des**	10	**ex1010**	11	**s38417**	8
apex2	12	**Diffeq**	8	**frisc**	15	**s38584**	8
apex4	13	**Dsip**	8	**misex3**	11	**seq**	12
bigkey	9	**Elliptic**	12	**pdc**	19	**spla**	15
clma	14	**ex5p**	14	**s298**	8	**tseng**	8

Tabela 5: Número de trilhas obtidas pela configuração default do VPR 5.0

O processo de alocação foi realizado pela ferramenta de alocação do VPR 5.0. Os arquivos que resultaram deste processo foram utilizados tanto para o roteamento da ferramenta VPR 5.0 como para o roteamento da heurística GRASP/ILS.

6.2 Testes *Timing-Driven*

As tabelas 6, 8, 10, 12 e 14 ilustram os resultados obtidos com o VPR 5.0 e pela heurística GRASP/ILS obtidos sobre o *benchmark* MCNC, onde “Melhor” é o melhor resultado obtido naquele circuito entre 10 execuções, “Média” é a média das 10 execuções e “Pior” é o pior resultado obtido nas 10 execuções. Os valores estão expressos em nanosegundos. Cinco valores de α foram utilizados, sendo eles: 0.1, 0.2, 0.3, 0.4 e 0.5.

O primeiro teste foi realizado com o α=0.1. Comparando o campo “Média” com os resultados do VPR 5.0, o roteador GRASP/ILS apresentou um melhoramento de 5,14% nos caminhos críticos dos circuitos MCNC em relação ao VPR 5.0. Foi o α que apresentou a 4ª melhor média entre os resultados. O α=0.2 apresentou um melhoramento de 6,20%, sendo o melhor resultado apresentado; α=0.3 obteve 5,61%, sendo o segundo melhor resultado; α=0.4 com 4,99% de melhoramento obtendo o pior resultado; α=0.5 com 5,54% obtendo o terceiro melhor resultado.

Nas tabelas 7, 9, 11, 13 e 15 compara-se o tempo de execução da heurística GRASP/ILS com o VPR 5.0. Os valores estão expressos em segundos.

	Critical Path						
	VPR 5	GRASP/ILS ALFA 0.1					
Circuito		Melhor	Var %	Média	Var %	Pior	Var %
alu4	103,68 ns	79,34 ns	23,47	83,57 ns	19,39	101,73 ns	1,88
apex2	94,46 ns	94,46 ns	0,00	95,09 ns	-0,67	100,68 ns	-6,59
apex4	105,03 ns	86,18 ns	17,95	88,78 ns	15,47	93,22 ns	11,24
bigkey	76,64 ns	74,28 ns	3,08	74,34 ns	3,00	74,89 ns	2,29
clma	207,59 ns	206,95 ns	0,30	206,96 ns	0,30	206,97 ns	0,30
des	91,12 ns	90,99 ns	0,14	91,05 ns	0,07	91,12 ns	0,00
diffeq	62,62 ns	62,59 ns	0,05	63,03 ns	-0,64	66,90 ns	-6,83
dsip	56,86 ns	56,85 ns	0,01	57,52 ns	-1,15	58,64 ns	-3,14
elliptic	143,01 ns	108,41 ns	24,19	114,47 ns	19,96	149,84 ns	-4,78
ex5p	93,46 ns	72,61 ns	22,31	75,40 ns	19,32	80,73 ns	13,62
ex1010	189,98 ns	186,06 ns	2,07	186,97 ns	1,58	192,10 ns	-1,11
frisc	134,43 ns	133,81 ns	0,46	133,88 ns	0,41	134,42 ns	0,01
misex3	95,57 ns	76,84 ns	19,59	86,16 ns	9,85	97,33 ns	-1,84
pdc	150,98 ns	150,98 ns	0,00	162,02 ns	-7,31	218,59 ns	-44,78
s298	141,31 ns	130,62 ns	7,56	132,88 ns	5,97	136,27 ns	3,57
s38417	99,64 ns	99,01 ns	0,63	100,48 ns	-0,84	107,82 ns	-8,21
s38584	114,76 ns	114,76 ns	0,00	114,76 ns	0,00	114,77 ns	-0,01
seq	90,54 ns	78,95 ns	12,80	79,56 ns	12,13	81,31 ns	10,19
spla	139,96 ns	122,58 ns	12,42	129,72 ns	7,32	144,99 ns	-3,59
tseng	55,26 ns	54,65 ns	1,11	54,83 ns	0,78	56,36 ns	-1,98
TOTAL	2246,88 ns	2080,90 ns	7,39	2131,45 ns	5,14	2308,66 ns	-2,75

Tabela 6: Resultados Critical Path do GRASP/ILS com α 0.1

Na tabela 6, pode-se observar bons resultados nas instâncias “alu4”, “apex4”, “elliptic”, “ex5p”, “misex”, “seq” e “spla”, com um melhoramento que varia de 12,42% até 23,47%. Nas instâncias “bigkey”, “ex1010”, “s298” e “tseng” houve um melhoramento não muito expressivo que varia de 1,11% até 7,56%. Os circuitos restantes: “apex2”, “clma”, “des”, “diffeq”, “dsip”, “frisc”, “pdc”, “s38417”, não havia como melhorar os resultados, pois estão pertos de um roteamento considerado possivelmente ótimo no critério *Critical Path* para a quantidade de trilhas que foram roteados, apresentando assim uma variação que vai 0,00% até 0,63%.

Quanto ao tempo, conseguiu-se tempo competitivo em poucas instâncias devido ao critério de parada que foi adicionado ao algoritmo. O critério de parada só funcionou para as instâncias que obtiveram resultados pertos do melhor valor conhecido e durante a fase de construção, ou seja, as instâncias que tiveram variação entre 0,00% e 0,63%. Nos outros casos, o tempo foi muito superior ao VPR 5.0 chegando a casos de quase 100 vezes mais tempo. Esta característica se repetiu para todos os α’s.

	Tempo de execução						
	VPR 5.0	GRASP/ILS ALFA 0.1					
Circuito		Melhor	Var %	Média	Var %	Pior	Var %
alu4	22 s	388 s	-1663,64	488 s	-2118,18	814 s	-3600,00
Apex2	15 s	17 s	-13,33	79 s	-426,67	214 s	-1326,67
Apex4	16 s	328 s	-1950,00	435 s	-2618,75	636 s	-3875,00
bigkey	12 s	13 s	-8,33	78 s	-550,00	251 s	-1991,67
Clma	103 s	539 s	-423,30	1792 s	-1639,81	3395 s	-3196,12
Dês	27 s	85 s	-214,81	310 s	-1048,15	372 s	-1277,78
diffeq	5 s	119 s	-2280,00	133 s	-2560,00	158 s	-3060,00
Dsip	28 s	623 s	-2125,00	798 s	-2750,00	1108 s	-3857,14
elliptic	46 s	41 s	10,87	163 s	-254,35	429 s	-832,61
ex5p	11 s	110 s	-900,00	238 s	-2063,64	308 s	-2700,00
ex1010	58 s	403 s	-594,83	1102 s	-1800,00	2036 s	-3410,34
Fris	40 s	123 s	-207,50	647 s	-1517,50	828 s	-1970,00
misex3	12 s	310 s	-2483,33	366 s	-2950,00	479 s	-3891,67
Pdc	56 s	148 s	-164,29	508 s	-807,14	826 s	-1375,00
s298	9 s	556 s	-6077,78	663 s	-7266,67	863 s	-9488,89
s38417	40 s	302 s	-655,00	1034 s	-2485,00	1369 s	-3322,50
s38584	91 s	128 s	-40,66	511 s	-461,54	1116 s	-1126,37
Seq	15 s	96 s	-540,00	283 s	-1786,67	423 s	-2720,00
Spla	37 s	942 s	-2445,95	1159 s	-3032,43	1867 s	-4945,95
tseng	4 s	11 s	-175,00	82 s	-1950,00	113 s	-2725,00
TOTAL	647 s	5282 s	-716,38	10869 s	-1579,91	17605 s	-2621,02

Tabela 7: Tempo de execução Critical Path do GRASP/ILS com α 0.1

	Critical Path						
	VPR 5.0	GRASP/ILS ALFA 0.2					
Circuito		Melhor	Var %	Média	Var %	Pior	Var %
alu4	103,68 ns	78,73 ns	24,07	82,36 ns	20,56	99,38 ns	4,15
apex2	94,46 ns	94,46 ns	0,00	94,97 ns	-0,54	99,50 ns	-5,34
apex4	105,03 ns	82,25 ns	21,68	89,50 ns	14,79	93,83 ns	10,66
bigkey	76,64 ns	74,28 ns	3,08	74,46 ns	2,85	75,48 ns	1,51
clma	207,59 ns	206,95 ns	0,30	208,36 ns	-0,37	220,99 ns	-6,46
des	91,12 ns	90,98 ns	0,15	91,02 ns	0,11	91,06 ns	0,07
diffeq	62,62 ns	62,59 ns	0,05	62,60 ns	0,04	62,61 ns	0,03
dsip	56,86 ns	56,85 ns	0,01	57,39 ns	-0,93	60,98 ns	-7,25
elliptic	143,01 ns	108,41 ns	24,19	108,84 ns	23,89	112,01 ns	21,68
ex5p	93,46 ns	72,60 ns	22,32	74,89 ns	19,87	81,33 ns	12,98
ex1010	189,98 ns	186,06 ns	2,07	186,10 ns	2,04	186,41 ns	1,88
frisc	134,43 ns	133,80 ns	0,47	133,93 ns	0,37	134,42 ns	0,01
misex3	95,57 ns	73,83 ns	22,75	79,12 ns	17,21	82,25 ns	13,93
pdc	150,98 ns	150,97 ns	0,01	151,44 ns	-0,30	152,76 ns	-1,18
s298	141,31 ns	130,75 ns	7,47	132,15 ns	6,48	133,27 ns	5,69
s38417	99,64 ns	99,03 ns	0,62	99,40 ns	0,24	101,18 ns	-1,55
s38584	114,76 ns	114,76 ns	0,00	114,76 ns	0,00	114,77 ns	-0,01
seq	90,54 ns	78,95 ns	12,80	80,10 ns	11,54	84,10 ns	7,11
spla	139,96 ns	120,45 ns	13,94	129,32 ns	7,60	147,54 ns	-5,42
tseng	55,26 ns	54,65 ns	1,11	56,97 ns	-3,09	77,80 ns	-40,78
TOTAL	2246,88 ns	2071,34 ns	7,81	2107,66 ns	6,20	2211,66 ns	1,57

Tabela 8: Resultados Critical Path do GRASP/ILS com α 0.2

Na tabela 8, os melhores resultados são os das instâncias “alu4”, “apex4”, “elliptic”, “ex5p”, “misex”, “seq” e “spla” e são melhores que os resultados apresentados na tabela 6. Pode-se encontrar uma variação nos resultados que vão de 12,80% até 24,19%. Assim como na tabela 6, as instâncias “bigkey”, “ex1010”, “s298” e “tseng” houve um melhoramento não muito expressivo, sendo a variação de 1,11% até 7,47%. Nos circuitos restantes: “apex2”, “clma”, “des”, “diffeq”, “dsip”, “frisc”, “pdc”, “s38417”, não havia como melhorar os resultados, pois estão pertos de um roteamento considerados o melhor conhecido no critério *Critical Path* para a quantidade de trilhas que foram roteados, apresentando assim uma variação que vai 0,00% até 0,62%.

	Tempo de execução						
	VPR 5.0	**GRASP/ILS ALFA 0.2**					
Circuito		**Melhor**	**Var %**	**Média**	**Var %**	**Pior**	**Var %**
Alu4	22 s	539 s	-2350,00	682 s	-3000,00	1034 s	-4600,00
apex2	15 s	19 s	-26,67	166 s	-1006,67	354 s	-2260,00
apex4	16 s	527 s	-3193,75	666 s	-4062,50	1010 s	-6212,50
Bigkey	12 s	10 s	16,67	45 s	-275,00	203 s	-1591,67
Clma	103 s	145 s	-40,78	1636 s	-1488,35	3491 s	-3289,32
Dês	27 s	37 s	-37,04	344 s	-1174,07	595 s	-2103,70
Diffeq	5 s	195 s	-3800,00	216 s	-4220,00	258 s	-5060,00
Dsip	28 s	102 s	-264,29	1074 s	-3735,71	1862 s	-6550,00
Elliptic	46 s	44 s	4,35	218 s	-373,91	453 s	-884,78
ex5p	11 s	175 s	-1490,91	371 s	-3272,73	484 s	-4300,00
ex1010	58 s	519 s	-794,83	1296 s	-2134,48	2026 s	-3393,10
Frisc	40 s	203 s	-407,50	861 s	-2052,50	1477 s	-3592,50
misex3	12 s	482 s	-3916,67	535 s	-4358,33	691 s	-5658,33
Pdc	56 s	129 s	-130,36	613 s	-994,64	1368 s	-2342,86
s298	9 s	842 s	-9255,56	901 s	-9911,11	1117 s	-12311,11
s38417	40 s	1519 s	-3697,50	1633 s	-3982,50	2025 s	-4962,50
s38584	91 s	111 s	-21,98	522 s	-473,63	1434 s	-1475,82
Seq	15 s	191 s	-1173,33	364 s	-2326,67	564 s	-3660,00
Spla	37 s	1481 s	-3902,70	1620 s	-4278,38	1969 s	-5221,62
Tseng	4 s	41 s	-925,00	121 s	-2925,00	145 s	-3525,00
TOTAL	647 s	7311 s	-1029,98	13884 s	-2045,90	22560 s	-3386,86

Tabela 9: Tempo de execução Critical Path do GRASP/ILS com α 0.2

	Critical Path						
	VPR 5.0	GRASP/ILS ALFA 0.3					
Circuito		Melhor	Var %	Média	Var %	Pior	Var %
alu4	103,68 ns	77,81 ns	24,95	81,37 ns	21,51	88,25 ns	14,88
apex2	94,46 ns	94,46 ns	0,00	95,27 ns	-0,86	102,49 ns	-8,51
apex4	105,03 ns	83,16 ns	20,82	88,68 ns	15,57	91,84 ns	12,56
bigkey	76,64 ns	74,28 ns	3,08	74,28 ns	3,08	74,29 ns	3,07
clma	207,59 ns	206,95 ns	0,30	206,96 ns	0,30	206,97 ns	0,30
des	91,12 ns	91,00 ns	0,13	91,04 ns	0,08	91,07 ns	0,05
diffeq	62,62 ns	62,59 ns	0,05	62,59 ns	0,04	62,61 ns	0,03
dsip	56,86 ns	56,85 ns	0,01	56,92 ns	-0,10	57,48 ns	-1,09
elliptic	143,01 ns	108,41 ns	24,19	110,92 ns	22,44	122,64 ns	14,24
ex5p	93,46 ns	73,03 ns	21,86	75,03 ns	19,72	78,51 ns	16,00
ex1010	189,98 ns	186,06 ns	2,07	192,44 ns	-1,30	246,15 ns	-29,56
frisc	134,43 ns	133,81 ns	0,46	133,88 ns	0,41	134,42 ns	0,01
misex3	95,57 ns	78,40 ns	17,97	82,76 ns	13,40	88,33 ns	7,58
pdc	150,98 ns	150,97 ns	0,01	156,20 ns	-3,46	187,66 ns	-24,30
s298	141,31 ns	130,59 ns	7,58	135,35 ns	4,21	161,03 ns	-13,96
s38417	99,64 ns	99,03 ns	0,62	100,78 ns	-1,14	113,41 ns	-13,82
s38584	114,76 ns	114,76 ns	0,00	114,76 ns	0,00	114,77 ns	-0,01
seq	90,54 ns	78,95 ns	12,80	80,25 ns	11,37	89,26 ns	1,42
spla	139,96 ns	119,07 ns	14,93	126,51 ns	9,61	136,42 ns	2,53
tseng	55,26 ns	54,64 ns	1,13	54,82 ns	0,80	55,75 ns	-0,89
TOTAL	2246,88 ns	2074,81 ns	7,66	2120,82 ns	5,61	2303,34 ns	-2,51

Tabela 10: Resultados Critical Path do GRASP/ILS com α 0.3

Na tabela 10, apesar da “Média” ser inferior que a da tabela 8, ocorre uma melhora no máximo valor da variação dos resultados das instâncias “alu4”, “apex4”, “elliptic”, “ex5p”, “misex”, “seq” e “spla” com destaque ao circuito “alu4”. Pode-se encontrar uma variação nos resultados nestas instâncias que vão de 12,80% até 24,95%. Assim como na tabela 6, nas instâncias “bigkey”, “ex1010”, “s298” e “tseng” houve um melhoramento não muito expressivo, sendo a variação de 1,13% até 7,58%. Pode-se observar uma pequena melhoria no circuito “tseng”. Nos circuitos restantes: “apex2”, “clma”, “des”, “diffeq”, “dsip”, “frisc”, “pdc”, “s38417”, não havia como melhorar os resultados, pois estão pertos de um roteamento considerado o melhor conhecido no critério *Critical Path* para a quantidade de trilhas que foram roteados, apresentando assim uma variação que vai 0,00% até 0,62%.

	Tempo de execução						
	VPR 5.0	GRASP/ILS ALFA 0.3					
Circuito		Melhor	Var %	Média	Var %	Pior	Var %
alu4	22 s	369 s	-1577,27	474 s	-2054,55	727 s	-3204,55
apex2	15 s	60 s	-300,00	166 s	-1006,67	284 s	-1793,33
apex4	16 s	347 s	-2068,75	407 s	-2443,75	524 s	-3175,00
bigkey	12 s	22 s	-83,33	94 s	-683,33	150 s	-1150,00
clma	103 s	147 s	-42,72	2088 s	-1927,18	3532 s	-3329,13
des	27 s	42 s	-55,56	319 s	-1081,48	469 s	-1637,04
diffeq	5 s	128 s	-2460,00	151 s	-2920,00	199 s	-3880,00
dsip	28 s	689 s	-2360,71	848 s	-2928,57	1168 s	-4071,43
elliptic	46 s	43 s	6,52	150 s	-226,09	362 s	-686,96
ex5p	11 s	228 s	-1972,73	264 s	-2300,00	392 s	-3463,64
ex1010	58 s	540 s	-831,03	1166 s	-1910,34	1775 s	-2960,34
frisc	40 s	273 s	-582,50	752 s	-1780,00	1369 s	-3322,50
misex3	12 s	297 s	-2375,00	335 s	-2691,67	415 s	-3358,33
pdc	56 s	83 s	-48,21	734 s	-1210,71	1468 s	-2521,43
s298	9 s	565 s	-6177,78	724 s	-7944,44	912 s	-10033,33
s38417	40 s	997 s	-2392,50	1149 s	-2772,50	1403 s	-3407,50
s38584	91 s	273 s	-200,00	581 s	-538,46	854 s	-838,46
seq	15 s	124 s	-726,67	208 s	-1286,67	364 s	-2326,67
spla	37 s	955 s	-2481,08	1207 s	-3162,16	1610 s	-4251,35
tseng	4 s	16 s	-300,00	70 s	-1650,00	119 s	-2875,00
TOTAL	647 s	6198 s	-857,96	11887 s	-1737,25	18096 s	-2696,91

Tabela 11: Tempo de execução Critical Path do GRASP/ILS com α 0.3

	Critical Path						
	VPR 5.0	GRASP/ILS ALFA 0.4					
Circuito		Melhor	Var %	Média	Var %	Pior	Var %
alu4	103,68 ns	79,70 ns	23,13	86,59 ns	16,48	110,36 ns	-6,44
apex2	94,46 ns	94,46 ns	0,00	94,47 ns	-0,01	94,55 ns	-0,10
apex4	105,03 ns	84,35 ns	19,69	91,42 ns	12,96	106,16 ns	-1,07
bigkey	76,64 ns	74,28 ns	3,08	74,28 ns	3,08	74,29 ns	3,07
clma	207,59 ns	206,95 ns	0,30	207,34 ns	0,12	210,77 ns	-1,53
des	91,12 ns	90,99 ns	0,14	91,05 ns	0,07	91,19 ns	-0,08
diffeq	62,62 ns	62,58 ns	0,07	62,59 ns	0,04	62,61 ns	0,03
dsip	56,86 ns	56,85 ns	0,01	56,98 ns	-0,21	57,46 ns	-1,05
elliptic	143,01 ns	108,41 ns	24,19	109,60 ns	23,36	118,72 ns	16,98
ex5p	93,46 ns	72,61 ns	22,31	75,61 ns	19,10	83,93 ns	10,20
ex1010	189,98 ns	186,06 ns	2,07	186,20 ns	1,99	186,70 ns	1,73
frisc	134,43 ns	133,81 ns	0,46	133,82 ns	0,45	133,83 ns	0,44
misex3	95,57 ns	80,74 ns	15,51	89,85 ns	5,98	113,34 ns	-18,59
pdc	150,98 ns	150,97 ns	0,01	155,81 ns	-3,20	187,73 ns	-24,34
s298	141,31 ns	130,57 ns	7,60	135,94 ns	3,80	165,09 ns	-16,83
s38417	99,64 ns	99,01 ns	0,63	99,02 ns	0,62	99,04 ns	0,61
s38584	114,76 ns	114,76 ns	0,00	114,76 ns	0,00	114,77 ns	-0,01
seq	90,54 ns	78,95 ns	12,80	79,41 ns	12,29	81,41 ns	10,09
spla	139,96 ns	121,23 ns	13,38	132,73 ns	5,17	165,01 ns	-17,90
tseng	55,26 ns	54,65 ns	1,11	57,25 ns	-3,60	80,07 ns	-44,89
TOTAL	2246,88 ns	2081,91 ns	7,34	2134,73 ns	4,99	2337,01 ns	-4,01

Tabela 12: Resultados Critical Path do GRASP/ILS com α 0.4.

Na tabela 12, ocorreu uma piora no máximo valor da variação dos resultados das instâncias “alu4”, “apex4”, “elliptic”, “ex5p”, “misex”, “seq” e “spla”. Pode-se encontrar uma variação nos resultados nestas instâncias que vão de 12,80% até 24,19%. Assim como na tabela 6, nas instâncias “bigkey”, “ex1010”, “s298” e “tseng” houve um melhoramento não muito expressivo, sendo a variação de 1,11% até 7,60%. Os circuitos restantes: “apex2”, “clma”, “des”, “diffeq”, “dsip”, “frisc”, “pdc”, “s38417”, não havia como melhorar os resultados, pois já estão pertos de um roteamento considerado o melhor conhecido no critério *Critical Path* para a quantidade de trilhas que foram roteados, apresentando assim uma variação que vai 0,00% até 0,63%.

	Tempo de execução						
	VPR 5.0	GRASP/ILS ALFA 0.4					
Circuito		Melhor	Var %	Média	Var %	Pior	Var %
alu4	22s	549s	-2395,45	626s	-2745,45	805s	-3559,09
apex2	15s	23s	-53,33	160s	-966,67	383s	-2453,33
apex4	16s	532s	-3225,00	574s	-3487,50	692s	-4225,00
bigkey	12s	23s	-91,67	130s	-983,33	379s	-3058,33
Clma	103s	554s	-437,86	1533s	-1388,35	2618s	-2441,75
Dês	27s	109s	-303,70	432s	-1500,00	704s	-2507,41
Diffeq	5s	203s	-3960,00	220s	-4300,00	263s	-5160,00
Dsip	28s	1004s	-3485,71	1133s	-3946,43	1411s	-4939,29
Elliptic	46s	47s	-2,17	144s	-213,04	372s	-708,70
ex5p	11s	240s	-2081,82	374s	-3300,00	532s	-4736,36
ex1010	58s	414s	-613,79	1516s	-2513,79	2474s	-4165,52
Frisc	40s	174s	-335,00	868s	-2070,00	1370s	-3325,00
Misex3	12s	467s	-3791,67	548s	-4466,67	597s	-4875,00
Pdc	56s	191s	-241,07	1020s	-1721,43	2184s	-3800,00
s298	9s	831s	-9133,33	967s	-10644,44	1237s	-13644,44
s38417	40s	800s	-1900,00	1417s	-3442,50	1754s	-4285,00
s38584	91s	108s	-18,68	581s	-538,46	1836s	-1917,58
Seq	15s	241s	-1506,67	423s	-2720,00	539s	-3493,33
Spla	37s	1502s	-3959,46	1632s	-4310,81	2162s	-5743,24
Tseng	4s	32s	-700,00	111s	-2675,00	160s	-3900,00
TOTAL	647s	8044s	-1143,28	14409s	-2127,05	22472s	-3373,26

Tabela 13: Tempo de execução Critical Path do GRASP/ILS com α 0.4

	Critical Path						
	VPR 5.0	GRASP/ILS ALFA 0.5					
Circuito		Melhor	Var %	Média	Var %	Pior	Var %
alu4	103,68 ns	77,02 ns	25,71	81,90 ns	21,00	83,50 ns	19,46
apex2	94,46 ns	94,46 ns	0,00	94,49 ns	-0,03	94,70 ns	-0,25
apex4	105,03 ns	83,58 ns	20,42	89,87 ns	14,43	92,51 ns	11,91
bigkey	76,64 ns	74,28 ns	3,08	74,46 ns	2,85	75,47 ns	1,52
clma	207,59 ns	206,95 ns	0,30	208,50 ns	-0,44	222,32 ns	-7,10
des	91,12 ns	90,99 ns	0,14	91,06 ns	0,06	91,19 ns	-0,08
diffeq	62,62 ns	62,59 ns	0,05	62,60 ns	0,04	62,61 ns	0,03
dsip	56,86 ns	56,85 ns	0,01	57,16 ns	-0,52	58,07 ns	-2,13
elliptic	143,01 ns	108,41 ns	24,19	108,50 ns	24,13	109,02 ns	23,76
ex5p	93,46 ns	72,60 ns	22,32	76,64 ns	18,00	86,99 ns	6,93
ex1010	189,98 ns	186,07 ns	2,06	186,56 ns	1,80	188,19 ns	0,94
frisc	134,43 ns	133,80 ns	0,47	133,82 ns	0,46	133,83 ns	0,44
misex3	95,57 ns	80,14 ns	16,14	86,76 ns	9,21	100,40 ns	-5,06
pdc	150,98 ns	150,99 ns	-0,01	153,30 ns	-1,54	172,55 ns	-14,28
s298	141,31 ns	130,59 ns	7,58	131,94 ns	6,63	133,53 ns	5,51
s38417	99,64 ns	99,02 ns	0,62	99,03 ns	0,61	99,06 ns	0,58
s38584	114,76 ns	114,76 ns	0,00	114,76 ns	0,00	114,77 ns	-0,01
seq	90,54 ns	78,95 ns	12,80	82,69 ns	8,67	96,71 ns	-6,81
spla	139,96 ns	120,25 ns	14,09	133,62 ns	4,53	171,43 ns	-22,48
tseng	55,26 ns	54,64 ns	1,13	54,71 ns	1,01	55,17 ns	0,16
TOTAL	2246,88 ns	2076,94 ns	7,56	2122,35 ns	5,54	2242,01 ns	0,22

Tabela 14: Resultados Critical Path do GRASP/ILS com α 0.5

Na tabela 14, ocorre uma melhora no máximo valor da variação dos resultados das instâncias “alu4”, “apex4”, “elliptic”, “ex5p”, “misex”, “seq” e “spla”. Pode-se encontrar o melhor máximo valor com 25,71% na instância “alu4”. A variação dos resultados nestas instâncias vão de 12,80% até 25,71%. Assim como na tabela 6, nas instâncias “bigkey”, “ex1010”, “s298” e “tseng” houve um melhoramento não muito expressivo, sendo a variação de 1,13% até 7,58%. Os circuitos restantes: “apex2”, “clma”, “des”, “diffeq”, “dsip”, “frisc”, “pdc”, “s38417”, não havia como melhorar os resultados, pois já estão pertos de um roteamento considerado o melhor conhecido no critério *Critical Path* para a quantidade de trilhas que foram roteados, apresentando assim uma variação que vai 0,00% até 0,63%.

	Tempo de execução						
	VPR 5.0	GRASP/ILS ALFA 0.5					
Circuito		Melhor	Var %	Média	Var %	Pior	Var %
alu4	22 s	364 s	-1554,55	426 s	-1836,36	523 s	-2277,27
apex2	15 s	18 s	-20,00	98 s	-553,33	348 s	-2220,00
apex4	16 s	348 s	-2075,00	376 s	-2250,00	436 s	-2625,00
bigkey	12 s	15 s	-25,00	55 s	-358,33	104 s	-766,67
clma	103 s	299 s	-190,29	1089 s	-957,28	3017 s	-2829,13
des	27 s	166 s	-514,81	381 s	-1311,11	654 s	-2322,22
diffeq	5 s	134 s	-2580,00	152 s	-2940,00	187 s	-3640,00
dsip	28 s	654 s	-2235,71	761 s	-2617,86	961 s	-3332,14
elliptic	46 s	53 s	-15,22	151 s	-228,26	383 s	-732,61
ex5p	11 s	154 s	-1300,00	258 s	-2245,45	357 s	-3145,45
ex1010	58 s	762 s	-1213,79	1398 s	-2310,34	2607 s	-4394,83
frisc	40 s	61 s	-52,50	622 s	-1455,00	1095 s	-2637,50
misex3	12 s	310 s	-2483,33	376 s	-3033,33	586 s	-4783,33
pdc	56 s	455 s	-712,50	618 s	-1003,57	909 s	-1523,21
s298	9 s	570 s	-6233,33	762 s	-8366,67	970 s	-10677,78
s38417	40 s	424 s	-960,00	1120 s	-2700,00	1498 s	-3645,00
s38584	91 s	128 s	-40,66	422 s	-363,74	879 s	-865,93
seq	15 s	104 s	-593,33	317 s	-2013,33	491 s	-3173,33
spla	37 s	938 s	-2435,14	1190 s	-3116,22	1650 s	-4359,46
tseng	4 s	26 s	-550,00	78 s	-1850,00	124 s	-3000,00
TOTAL	647 s	5983 s	-824,73	10650 s	-1546,06	17779 s	-2647,91

Tabela 15: Tempo de execução Critical Path do GRASP/ILS com α 0.5

Os resultados apresentados mostrou que dos cinco α's: 0.1, 0.2, 0.3, 0.4 e 0.5, o α de valor 0.2 mostrou com os melhores resultados no critério "média". Isso não significa que não obteve-se resultados satisfatórios com os outros α's, mas que o α mais indicado para rotear um circuito, independente de sua natureza, seria o valor 0.2.

6.3 Testes para Redução da Quantidade de Trilhas – VPR 5.0

Os testes no critério de número de trilhas consistiram em tentar rotear os cricuitos dos *benchmark* MCNC com número de trilhas inferiores ao que inicialmente o VPR 5.0 conseguiu rotear. Tentou-se, tanto com o VPR 5.0, quanto com o GRASP/ILS com α=0.2, rotear os circuitos utilizando número de trilhas fixos inferiores ao usado anteriormente. Como se pode observar na Tabela 16, o VPR 5.0 não conseguiu reduzir nenhum número de trilhas dos já obtidos. Essa verificação é necessária, pois se um roteador não consegue rotear um circuito com *x* trilhas não quer dizer que ele não consiga com *x-1* trilhas (BETZ, 1997). O GRASP/ILS usou 12 trilhas a menos que o VPR 5.0 para rotear os circuitos MCNC, ou seja, usou 5,30% menos trilhas para rotea-los. Para realizar os testes para redução do número de trilhas foram utilizados os mesmos arquivos de *placement* utilizados nos testes de tempo crítico.

	Trilhas	
Circuito	**VPR 5.0**	**GRASP/ILS 0.2**
alu4	11	**10**
apex2	12	12
apex4	13	13
bigkey	9	**8**
clma	14	**13**
des	10	**9**
diffeq	8	8
dsip	8	8
elliptic	12	**11**
ex5p	14	**13**
ex1010	11	11
frisc	15	**13**
misex3	11	11
pdc	19	**17**
s298	8	8
s38417	8	8
s38584	8	8
seq	12	12
spla	15	**14**
tseng	8	**7**
Total	**226**	**214**

Tabela 16: Resultados número de trilhas do GRASP/ILS com α 0.2

6.4 FPGA Place and Route Challenge

Em 1997, Vaughn Betz propõe um desafio chamado de "*FPGA Place and Route Challenge*" (Desafio de posicionamento e roteamento de FPGA) (BETZ,1997), no qual pagaria 1,00 US$ para cada trilha que utilizassem a menos que o programa VPR para posicionar e rotear os circuitos da MCNC. Este desafio foi proposto para incentivar a pesquisa e o desenvolvimento de programas para roteamento de FPGAs. Para facilitar a comparação entre os resultados, no site http://www.eecg.toronto.edu/~vaughn/challenge/challenge.html estão disponíveis a arquitetura utilizada, os arquivos de *placement* utilizados para obtenção dos resultados obtidos pelo VPR 4.30, os resultados de alguns algoritmos encontrados na literatura e o código fonte do VPR.

	Trilhas		
Circuito	**VPR 4.30**	**CornNC**	**GRASP/ILS 0.2**
alu4	**9**	**8**	**9**
apex2	**10**	**9**	**11**
apex4	**11**	**10**	**12**
bigkey	**6**	**5**	**6**
clma	**10**	**9**	**11**
Dês	**7**	**6**	**7**
diffeq	**7**	**6**	**7**
Dsip	**5**	**5**	**6**
elliptic	**9**	**8**	**10**
ex5p	**11**	**10**	**12**
ex1010	**9**	**8**	**10**
Frisc	**11**	**10**	**11**
misex3	**10**	**9**	**10**
Pdc	**15**	**13**	**16**
s298	**6**	**6**	**7**
s38417	**6**	**6**	**7**
s38584	**7**	**7**	**8**
Seq	**10**	**9**	**10**
Spla	**12**	**10**	**12**
tseng	**6**	**5**	**6**
Total	**177**	**159**	**190**

Tabela 17: Resultados número de trilhas do GRASP/ILS com α 0.2 em comparação com os resultados do *FPGA Place and Route Challenge*

Na tabela 17, encontram-se os resultados do VPR 4.30, que superam em 21,68% o VPR 5.0 rodando com algoritmo padrão de placement em cima do *benchmark* MCNC. Também se encontra os resultados do algoritmo CornNC e do

GRASP/ILS com α=0.2 em relação ao benchmark MCNC utilizando os arquivos de *placement* fornecidos pelo Vaughn Betz no site do *FPGA Place and Route Challenge*. Segundo o autor, para obter os resultados do VPR 4.30, foi utilizado o *placement bounding Box*, a quantidade de iterações foi configurada para 100, *pres_fac* foi configurado para 1.5 e o algoritmo de roteamento foi o *breadth-first*. Em relação ao melhor resultado encontrado na literatura, o Corn-NC, o algoritmo GRASP/ILS utilizou 16,31% mais trilhas. Em comparação ao VPR 4.30, o GRASP/ILS utilizou 6,84% mais trilhas.

Na tabela 18, se compara os resultados do VPR 5.0 roteando os circuitos do *benchmark* MCNC com os arquivos de *placement* fornecidos no site do *FPGA Place and Route Challenge* e a mesma configuração utilizada para obtenção dos resultados do VPR 4.30, sendo elas: a quantidade iterações foi configurada para 100, a penalidade dada aos recursos compartilhados (*pres_fac*) foi configurado para 1.5 e o algoritmo de roteamento foi o *breadth-first.* Os resultados obtidos é que o GRASP/ILS utilizou 4 trilhas a menos que o VPR 5.0 para rotear os cricuitos, ou seja, 2,10% menos trilhas. Na coluna VPR 4.30*, tentou-se reproduzir em laboratório os resultados encontrados na literatura, sem sucesso.

	Trilhas			Critical Path		
Circuito	**VPR 5.0**	**VPR 4.30***	**GRASP/ILS 0.2**	**VPR 4.30***	**GRASP/ILS 0.2**	**Var %**
alu4	9	9	9	1,82E-07	1,08E-07	40,46
apex2	11	10	**11**	2,13E-07	**1,28E-07**	**39,85**
apex4	12	11	**12**	1,98E-07	**1,13E-07**	**42,71**
bigkey	6	6	6	2,10E-07	1,05E-07	49,70
clma	11	11	11	3,82E-07	2,50E-07	34,55
des	8	7	7	1,82E-07	1,15E-07	36,65
diffeq	7	7	7	1,20E-07	8,41E-08	29,67
dsip	6	6	6	1,76E-07	7,87E-08	55,23
elliptic	10	9	**10**	2,44E-07	**1,83E-07**	**25,07**
ex5p	13	13	**12**	1,53E-07	**1,09E-07**	**28,66**
ex1010	10	9	**10**	4,24E-07	1,86E-07	56,11
frisc	12	11	11	2,64E-07	1,68E-07	36,53
misex3	11	10	10	1,66E-07	1,05E-07	36,45
pdc	16	15	**16**	4,09E-07	**2,38E-07**	**41,71**
s298	7	7	7	3,78E-07	1,94E-07	48,59
s38417	7	7	7	3,98E-07	1,79E-07	55,02
s38584	8	8	8	2,49E-07	1,15E-07	53,85
seq	11	10	10	2,14E-07	1,26E-07	41,10
spla	13	12	12	2,87E-07	1,74E-07	39,44
tseng	6	6	6	1,02E-07	7,41E-08	27,11
Total	**194**	**184**	**188**	**4,95E-006**	**2,83E-006**	**42,72**

Tabela 18: Resultados número de trilhas/*Critcal Path* do GRASP/ILS com α 0.2 com o VPR 4.30 e 5.0 no *FPGA Place and Route Challenge*

Comparado com os resultados do GRASP/ILS com os resultados obtidos pelo VPR 4.30*, observa-se que o GRASP/ILS obteve o mesmo número de trilhas em várias instâncias e conseguiu reduzir uma trilha na instância “ex5p”. Comparando o critério tamanho do caminho crítico das instâncias de mesmo número de trilhas observa-se ganhos de até 56,11%.

6.5 Avaliações Probabilísticas

Nesta seção, é colocada como meta a análise da regularidade das heurísticas aqui propostas. Isso se torna importante na medida que heurísticas com componentes aleatórias, como é o caso do GRASP, podem apresentar soluções bem distintas a cada execução de uma dada instância. Desta forma, foi utilizada uma outra abordagem para análise de desempenho empírico de metaheurísticas, proposta em Aiex et al. (Aiex, 2002). Este tipo de análise consiste em tomar os tempos de processamento que cada algoritmo requer para atingir um valor alvo na função objetivo. Para tanto, no instante em que cada algoritmo encontra uma solução melhor ou igual (em problemas de maximização) ao alvo fixado, o tempo é registrado e a execução é interrompida.

Para esta bateria de testes, cada algoritmo foi executado 50 vezes para cada instância avaliada usando os mesmos parâmetros inicialmente adotados nas análises anteriores. Após 50 execuções, os tempos foram tomados e dispostos em ordem crescente numa lista *L*. A cada tempo de CPU *t* obtido, foi associada a probabilidade $pi = \left(i - 1/2\right)/50$, onde *i* é a ordem que *t* aparece na lista ordenada *L*. O gráfico foi desenhado tomando-se cada ponto *(ti, pi)*.

As três instâncias analisadas foram *apex2, clma, dsip.* Conforme consta na tabela 4, para a arquitetura escolhida, o *apex2* possui 1878 blocos lógicos, 41 pinos de entrada e saída e 1916 *nets*; o *clma* possui 8383 blocos lógicos, 144 pinos de entrada e saída e 8445 *nets*; e a instância *dsip* possui 1370 blocos lógicos 426 pinos de entrada e saída e 1600 *nets.* Como alvo fixado, foi utilizado os valores de tamanho crítico obtidos pelo VPR 5.0 para as três instâncias.

Os resultados obtidos na instância *apex2* foram os melhores entre as três instâncias e podem ser observados na Figura 40. Há uma grande concentração de execuções que atingiram o alvo fixado em menos de 50s.

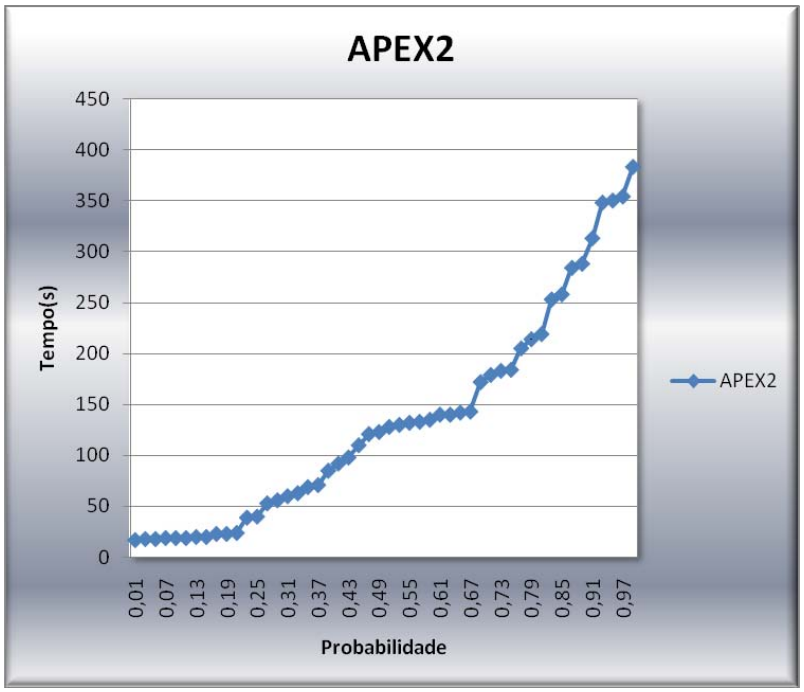


Figura 40: Análise Probabilística da instância *apex2*.

O teste em cima do maior circuito MCNC obteve um comportamento linear que pode ser observado na Figura 41. A partir do tempo de 3000s, apesar do gráfico ainda ter um comportamento linear, temos uma probabilidade menor de ocorrências acima deste tempo.

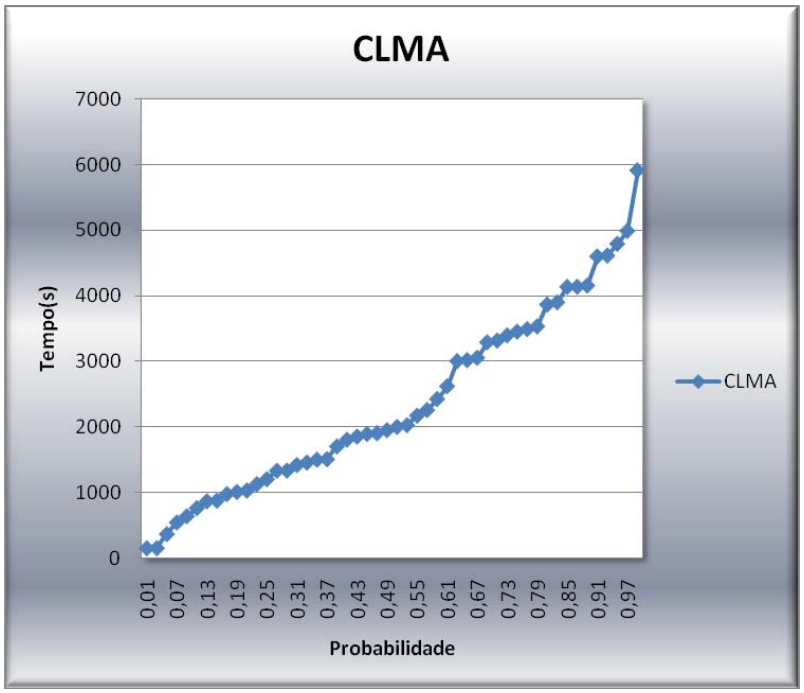


Figura 41: Análise Probabilística da instância *clma*.

Os resultados obtidos em cima do circuito *dsip* são apresentados na Figura 42. Apenas em uma ocorrência o algoritmo atingiu o alvo fixado na primeira iteração. A maioria das execuções atingiu o alvo fixado entre o tempo de 600s e 1200s.

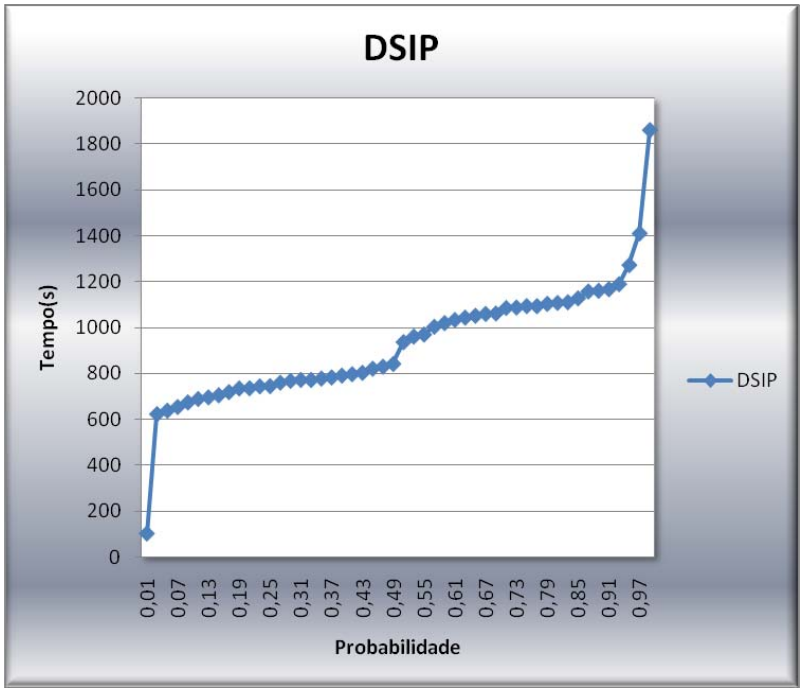


Figura 42: Análise Probabilística da instância *dsip*.

Conclusões

Esta dissertação tratou do Problema de Roteamento de Circuitos Integrados Baseados em FPGAs. Para resolvê-lo, um algoritmo baseado na metaheurísticas GRASP e ILS foi proposto. É a primeira vez que estas metaheurísticas foram utilizadas para solucionar este problema.

Para construção da solução inicial, foi utilizado o método proposto pelo GRASP, com a utilização de LCR para gerar soluções variadas. O método de roteamento de *nets* utilizado é o *timing-driven* encontrado no VPR 5.0 (LUU,2009). Na etapa de busca local, a metaheurística ILS foi utilizada onde a perturbação consistia em colocar uma das *nets* com maior tempo de atraso no final da lista de *nets* ordenadas por atraso para em seguida tentar roteá-las.

O algoritmo desenvolvido foi testado no *benchmark* MCNC e comparado com resultados obtidos pelo VPR 5.0. Os critérios utilizados para comparação dos resultados foram o tamanho do caminho crítico e o número mínimo de trilhas para rotear o circuito.

Nos testes, dos 20 circuitos, o algoritmo foi capaz de melhorar 18 instâncias no critério tamanho crítico, com ganhos de até 25,7% e empatar nas outras duas instâncias. Na média, conseguiu melhorar em 6,20% os resultados obtidos pelo VPR 5.0. No critério de número de trilhas os resultados não foram satisfatórios, utilizando 21,68% de trilhas a mais que o algoritmo Corn-NC (SO,2007). Comparando com o VPR 5.0, com o *placement* padrão realizado pela ferramenta, o algoritmo mostrou um resultado satisfatório, melhorando em 5,30% a quantidade de trilhas utilizadas. Ao executar o VPR 5.0 com a mesma configuração que o VPR 4.30 para obtenção dos resultados do *FPGA Place and Route Challenge,* o algoritmo GRASP/ILS conseguiu melhorar em 2,1% os resultados.

Como trabalhos futuros sugerem-se: a incorporação de procedimentos mais eficientes para reduzir a dependência do fator α para geração da solução inicial; implementação de outros mecanismos de perturbação; o desenvolvimento de estratégias paralelas para o algoritmo proposto; testes mais aprofundados no critério de redução de trilhas através de: mudança de α's, variação da penalidade dada aos recursos compartilhados (*pres_fac*) e quantidade de iterações para rotear.

Referências

Ahuja, R. K., Magnanti, T. L., e Orlin, J. B. “Network Flows: Theory, Algorithms, and Applications.” Englewood Cliffs, NJ: Prentice-Hall,1993.

Aiex, R.; Resende, M. G. C. e Ribeiro, C. C. “Probability distribution of solution time in GRASP: An experimentak investigations”. Journal of Heuristics, v. 8, páginas 343-373, 2002.

Alexander, M. J., Cohoon, J. P., Ganley, J. L., Robins, G. “An Architecture-Independent Approach to FPGA Routing Based on Multi-Weighted Graphs”. In *Proc. European Design Automation Conf.,* páginas 259-264, 1994.

Alpert, C., Chan, T., et al, D. Huang. “Quadratic Placement Revisited”. In *Design Automation Conference*, páginas 752-757, 1997.

Altera Corporation. Stratix IV device handbook version SIV5V1-1.1, July 2008. http://www.altera.com/literature/hb/ stratix-iv/stratix4_handbook.pdf.

Bazaraa, M. S., Sherali, H. D. e Shetty, C. M. “Nonlinear Programming: Theory and Algorithms”, 2nd ed. New York: Wiley, 1993.

Beauchamp, M. J. et al. “Embedded floating-point unitsin FPGAs.” In FPGA ’06: Proceedings of the 14th international ACM/SIGDA symposium on Field programmable gate arrays, páginas 12–20, New York, NY, USA, 2006. ACM.

Betz, V. The “FPGA Place and Route Challenge”. http://www.eecg.toronto.edu/~vaughn/challenge/challenge.html. ,1997

Betz, V., e Rose, J. “VPR: A new packing, placement and routing tool for FPGA research.” In Proceedings of the 7th International Workshop on Field Programmable Logic, 1997.

Betz, V., e Rose, J. “VPR: A new packing, placement and routing tool for FPGA research,” in Proc. 7th Annu. Workshop Field Programmable Logic Applic., páginas 213–222, 1999.

Brown, S., Francis, R., Rose, J. e Vranesic Z. “Field-Programmable Gate Arrays.” Norwell, MA: Kluwer, 1992.

Cabral, L. A. F. “Paralelizando a Fase de Roteamento de Circuitos Baseados em FPGAs”. Tese – Universidade Federal do Rio de Janeiro, 2001.

Chen, C.P., Chu, C. C. N., e Wong, D. F. “Fast and exact simultaneous gate and wire sizing by lagrangian relaxation,” in Proc. Int. Conf. Computer-Aided Design, páginas 614–621, 1997.

Cong, J. e Xu, S. “Delay-optimal technology mapping for FPGAs with heterogeneous LUTs.” In Design Automation Conference, 1998. Proceedings, páginas 704–707, 1998.

Ebeling, C., McMurchie, l., Hauck, S. A., e Burns S. “Placement and Routing Tools for the Triptych FPGA,” IEEE Trans. on VLSI, Dec. Páginas 473 -482, 1995.

Elmore, W. C. “The transient response of damped linear network with particular regard to wideband amplifiers,” J. Appl. Phys., vol. 19, no. 1, páginas 55–63, 1948.

Feo, T.A. e Resende, M.G.C. “Greedy Randomized Adaptive Search Procedures.” Journal of Global Optimization, 6, páginas 109-133, 1995.

Hart, P. E., Nilsson, N. J. e Raphael, B. “A formal basis for the heuristic determiniation of minimum cost paths. IEEE Trans. Syst. Sci. Cybernetics, 4: páginas 100–107,1968.

He, J. e Rose, J. “Advantages of heterogeneous logic block architectures for FPGAs.” In Proceedings of the IEEE Custom Integrated Circuits Conference, páginas 7.4.1 – 7.4.5, 1993.

Hitchcock, R. B., Smith, G. L. E Cheng, D. D. “Timing analysis of computer hardware,” IBM J. Res. Develop., vol. 26, no. 1, páginas 100–105, 1982.

Huang D., Kahng, A. “Partitioning-Based Standard-Cell Global Placement with an exact Objective”. In *ACM Symp on Physical Design*, páginas 18-25, 1997.

Jamieson P. e Rose, J. “Architecting hard crossbars on FPGAs and increasing their area efficiency with shadow clusters.” In International Conference on Field-Programmable Technology, 2007, páginas 57–64, 2007.

Kahng, A. B., Robins, G. “A New Class of Iterative Steiner Tree Heuristics With Good Performance”. IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems, v. 11, páginas 893-902, 1992.

Lee, C. Y. “An Algorithm for Path Connections and its Applications, “IRE Trans. Electron. Comput., Vol. EC=10, páginas 346 – 365, 1961.

Lee, S. e Wong, M. D. F. “Timing-Driven Routing for FPGAs Based on Lagrangian Relaxation” IEEE Transactions on Computer-Adided Design of Integreated Circuits and Systems, Vol. 22, No. 4, páginas 506-511, 2003

Lemieux, G. et al. “Directional and single-driver wires in FPGA interconnect.” In IEEE International Conference on Field-Programmable Technology, páginas 41–48, December 2004.

Lima, E. M. V. “Posicionamento Automático em Circuitos Implementados em FPGAs: Desenvolvimento de Rotinas de Aceleração Computacional.” Dissertação, Universidade Federal da Paraíba, 2008.

Lourenço, H. R., Martin, O. C., Stutzle, T., “Iterated Local Search. In: Fred Glover e Gary A. Kochenberger (eds.)”, Handbook of Metaheuristics, páginas 321-353. Kluwer Academic Publishers, Norwell, MA, 2002.

Luu, J., Kuon, L., Jamieson, P., Campbell, T., Ye, A., Fang, W. M. e Rose, J. “VPR 5.0: FPGA CAD and Architecture Exploration Tools with Single-Driver Routing, Heterogeneity and Process Scaling” (2008) Proceeding of the ACM/SIGDA international symposium on Field programmable gate arrays, California, USA, páginas 133-142, 2009.

Prais, M., and Ribeiro, C. C., “Parameter Variation in GRASP Implementations”. In Proceedings of the Third Metaheuristics International Conference, páginas 375–380, Angra dos Reis, Brazil, 1999.

Prais, M., and Ribeiro, C. C., “Reactive GRASP: An application to a matrix decomposition problem in TDMA traffic assignment”. INFORMS Journal on Computing, 12, páginas 164–176, 2000.

Prais, M., and Ribeiro, C. C., “Variação de Parâmetros em Procedimentos GRASP”. Investigación Operativa, 9 , páginas 1–20, 2000.

Nag, S., Rutenbar, R. “Performance-Driven Simultaneous Place and Route for Row-Based FPGAs”. In *ICCAD*, páginas 332-338, 1995.

Resende, M. G. C. e Ribeiro, C. C. “Greedy randomized adaptive search procedures.” In: Fred Glover e Gary A. Kochenberger (eds.), Handbook of Metaheuristics, Kluwer Academic Publishers, Norwell, MA, páginas 219-250, 2005.

Ribeiro, C. C., “Metaheuristics and Applications.” In *Advanced School on Artificial Intelligence*, Estoril, Portugal, 1996.

Rose, J., Francis, R., Lewis, D., Chow, P. “Architecture of Field Programmable Gate Arrays: The Effect of Logic Block Funcionality on Area Effciency”. *JSSC*, páginas 168-173, 1990.

Rose, J., Brown, S.“Flexibility of Interconnection Structures in Field Programmable Gate Arra.” *IEEE Journal of Solid State Circuits*, v. 26, n. 3, páginas 277-288, 1991.

Rose, J., Francis, Vranesic, Z. G., Snelgrove, W. M. “ALTOR: An Automatic Standard Cell Layout Program”. In *Proc. Can. Conf. VLSI ,* páginas 168-173, 1985.

Singh, A., Mukherjee, A., Marek-Sadowska, M. “Interconnect Pipelining in a Throughput-Intensive FPGA Architecture”, ACM/SIGDA Ninth International Symposium on Field-Programmable Gate Arrays, páginas 153-160, 2001.

Siozios, K, Sotiriadis, K, Pavlidis, V. F. e Soudris, D. “Exploring Alternative 3D FPGA Architectures: Design Methodology and CAD Tool Support” Field Programmable Logic and Applications, International Conference on Amsterdam, páginas : 652-655, 2007.

So, k. “Solving Hard Instances of FPGA Routing with a Congestion-Optimal Restrained-Norm Path Search Space” International Symposium on Physical Design, Austin, Texas, páginas 151-158, 2007.

Stützle, T. G. “Local Search Algorithms for Combinatorial Problems.” Tese (doutorado) Technische Universit¨at Darmstadt, Alemanha, 1998.

Subramanian, A., Cabral, L. A. F., Carvalho, G. R. “A hybrid metaheuristic for the Vehicle Routing Problem with Simultaneous Pick-up and Delivery.” In: XIII International Conference on Industrial Engineering and Operations Management., 2007, Foz do Iguaçu. Anais do XIII ICIEOM, 2007.

Swartz, W., Sechen, C. “Timing Driven Placement for Large Standard Cell Circuits”. In *Design Automation Conference*, páginas 211-215, 1995.

Torok, D. L. “Projeto Visando a Prototipação do Protocolo de Acesso ao Meio em Redes Ethernet.” Porto Alegre: PUC, 2001.

Tsu, W., Macy, K., Joshi, A., Huang, R., Walker, N., Tung, T., Rowhani, O., George, V., Wawrzynek, J. e DeHon, A. “HSRA: High-Speed, Hierarchical Synchronous Reconfigurable Array”, ACM Seventh International Symposium on Field-Programmable Gate Arrays, 1999.

Xilinx. Virtex-5 user guide, March 2008. UG190 (v4.0) http://www.xilinx.com/support/documentation/ user_guides/ug190.pdf.

Yang, S. “Logic Synthesis and Optimization Benchmarks, Version 3.0,” Microelectronics Center of North Carolina, Research Triangle Park, NC, 1991.