LUCIANA ROCHA CARDOSO

ALGORITMO DE POSICIONAMENTO POLINOMIAL PARA FPGA BASEADO EM TRAVESSIA DE GRAFOS

Dissertação apresentada à Universidade Federal de Viçosa, como parte das exigências do Programa de Pós-Graduação em Ciência da Computação, para obtenção do título de *Magister Scientiae*.

VIÇOSA
MINAS GERAIS - BRASIL
2013

Ficha catalográfica preparada pela Seção de Catalogação e Classificação da Biblioteca Central da UFV

T

C268a
2013

Cardoso, Luciana Rocha, 1975-

Algoritmo de posicionamento polinomial para FPGA baseado em travessia de grafos / Luciana Rocha Cardoso. - Viçosa, MG, 2013.

x, 74 f. : il. (algumas color.) ; 29 cm.

Inclui apêndice.

Orientador: Ricardo dos Santos Ferreira.

Dissertação (mestrado) - Universidade Federal de Viçosa.

Inclui bibliografia.

1. Computadores - Circuitos. 2. Circuitos lógicos. I. Universidade Federal de Viçosa. Departamento de Informática. Programa de Pós-Graduação em Ciência da Computação. II. Título.

CDD 22 ed. 005.1

LUCIANA ROCHA CARDOSO

ALGORITMO DE POSICIONAMENTO POLINOMIAL PARA FPGA BASEADO EM TRAVESSIA DE GRAFOS

Dissertação apresentada à Universidade Federal de Viçosa, como parte das exigências do Programa de Pós-Graduação em Ciência da Computação, para obtenção do título de *Magister Scientiae.*

APROVADA: dia de mês de ano.

_____ _____

José Augusto Miranda Nacif

André Gustavo dos Santos

(Coorientador)

_____

Ricardo dos Santos Ferreira

(Orientador)

VIÇOSA
MINAS GERAIS - BRASIL
2013

Dedico essa dissertação
Aos meus pais, Wilton e Maria Helena

A minhas irmãs, Flávia e Fabiana

Aos meus filhos, Marcelo Filho e Luciana

Ao meu orientador
Ricardo

Ao meu coorientador
André Gustavo

AGRADECIMENTOS

A DEUS por ter sido sempre tão generoso e presente em minha vida, me dando sabedoria e sobriedade nos momentos mais difíceis.

Agradeço aos meus pais, Wilton e Maria Helena que me propiciaram uma vida digna onde eu pudesse crescer, acreditando que tudo é possível, desde que sejamos honestos, íntegros de caráter e tendo a convicção de que desistir nunca seja uma ação contínua em nossas vidas; que sonhar e concretizar os sonhos só dependerão de nossa vontade.

As minhas irmãs, Flávia e Fabiana, pelo carinho, compreensão e pela grande ajuda. Aos meus filhos, Marcelo Filho e Luciana, pelo carinho compartilhado e pela paciência em me ter subtraído durante longos anos.

Stefany, te agradeço pela paciência, pois muitas vezes você suportou minha chatice sem ter culpa de nada. Agradeço-te pelo companheirismo, pois você sempre esteve junto a mim nas alegrias e nas tristezas me apoiando e me empurrando adiante não permitindo que eu caísse.

Aos meus amigos, Reinaldo Giliard e Priscila, pelo apoio, carinho, residência e incentivo em todos os momentos de minha jornada. Ao meu grande Glauber, por ter me apoiado nessa fase com incentivo para os estudos, amizade e compreensão nos momentos mais difíceis, sem você seria impossível concluir.

Ao meu orientador, Prof. Ricardo, pela amizade, pelo companheirismo e pelo incentivo ao longo dos anos até a conclusão deste trabalho. Por estar sempre disposto a ajudar, mesmo quando tudo parecia perdido. Pela perseverança e por ter acreditado que este trabalho fosse concluído. Ao meu coorientador, André Gustavo pela inestimável orientação.

Ao secretário da Pós-graduação, Altino, pelas conversas e por sempre se prontificar a ajudar durante todo o processo do mestrado.

Aos demais Professores do Departamento de Informática da UFV que contribuíram de forma direta ou indireta para a minha formação acadêmica.

Finalmente, a todos que fizeram parte desta longa e salutar jornada, os meus mais sinceros agradecimentos.

Muito obrigada!

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS E GRÁFICOS

LISTA ABREVIATURAS

AR - *Reconfigurable Architecture*

ASIC - *Application Specific Integrated Circuits*

BF - *Busca em Largura*

BLIF - *Berkeley Logic Interchange Format*

CAD - *Computer Aided Engineer*

CI – *Circuitos Integrados*

CLB - *Blocos Lógicos Configuráveis*

DF - *Busca em Profundidade*

E1 - *Estratégia 1*

E2 - *Estratégia 2*

FPGA - *Field-Programmable Gate Arrays*

GPP - *General Puporse Processors*

HDL - *Hardware Description Language*

I/O – *Input/Output*

IOB – *Input/Output Bloks*

JIT - *Just-int-time*

LUT – *Lookup Tables*

P&R – *Posicionamento e Roteamento*

SA - *Simulated Annealing*

SRAM - *Static Random Access Memory*

VHDL - *VHSIC Hardware Description Language*

VHSIC - *Very High Speed Integrated Circuits*

VPR - *Versatile Placement and Routing Tool*

RESUMO

CARDOSO, Luciana Rocha, M.Sc., Universidade Federal de Viçosa, julho de 2013. **Algoritmo de Posicionamento Polinomial para FPGA Baseado em Travessia de Grafos.** Orientador: Ricardo dos Santos Ferreira. Coorientador: André Gustavo dos Santos.

O *hardware* reconfigurável é uma solução intermediária entre *software* e *hardware*, oferece a flexibilidade do *software* e o desempenho do *hardware*. Nos anos oitenta, os FPGAs surgiram como circuitos reconfiguráveis comerciais e escaláveis. Possuem muita flexibilidade e resolveram com sucesso vários problemas nas últimas décadas com uma aceleração de 2 a 3 ordens de grandeza em relação à versão em *software*. Além disso, as aplicações são heterogêneas e demandam soluções que se adaptem em tempo de execução. Com a evolução da tecnologia, a complexidade do FPGA aumentou significativamente passando de centenas de blocos lógicos/interconexões para a ordem de milhões. Este avanço possibilita mais desempenho e flexibilidade, entretanto a complexidade para mapear as aplicações aumenta significativamente. Sendo que para soluções geradas em tempo de execução, o tempo para mapear as aplicações é mais crítico. O mapeamento é realizado em dois passos: posicionamento e roteamento. O problema de encontrar a melhor maneira de posicionar uma computação nos blocos lógicos (posicionamento) é NP-Completo. Uma vez posicionados, os blocos devem ser interligados (roteamento) que também é um problema NP-completo. Ambos os problemas são grandes desafios atuais para as ferramentas de FPGA. Esta dissertação tem foco no problema de posicionamento. Várias heurísticas já foram propostas ao longo das três últimas décadas, porém poucas apresentaram um desempenho adequado para posicionar e rotear blocos lógicos em tempo de execução. Este trabalho propõe uma exploração do espaço de projeto de um algoritmo polinomial de posicionamento baseado em travessia em grafos. Variações de travessia (largura e profundidade) são propostas e a qualidade da solução é avaliada medindo o tempo de execução, o caminho crítico e o número total de conexões. As variações tem foco nos pontos com múltiplos *fanout* e *fanouts* reconvergentes.

ABSTRACT

CARDOSO, Luciana Rocha, M.Sc., Universidade Federal de Viçosa, July, 2013. **Positioning Polynominal Algorithm for FPGA based-Crossing Graphs.** Adviser: Ricardo dos Santos Ferreira. Co-Adviser: André Gustavo dos Santos.

Nowadays, many applications requires high performance solutions, which could not be achieved by a software approach. Problems in bioinformatics, simulations, telecommunications, encryption, compression of large volumes of data are examples of these applications. A hardware approaches are high-performance, however, these approaches have high design costs. The reconfigurable hardware is an intermediate solution between software and hardware, as it offers the flexibility of the software and the performance of the hardware. Since 1980s, FPGA circuits have emerged as commercial and scalable platform for reconfigurable hardware. Several problems have been modeled and implemented by using FPGA with acceleration from 2 to 3 orders of magnitude. However, the FPGA complexity has significantly increased from hundreds of logical blocks/interconnections to the order of millions in last decades. First, the application should be mapped to the FPGA. The mapping consists on two steps: placement and routing. The placement will map the computing elements in logic blocks. This step is NP-Complete. Then, the routing step will perform the interconnections between the logic blocks. It is also a NP-complete problem. Both problems are one of current challenges in FPGA tools. Several heuristics have been proposed, but few of them are suitable for run-time approach. This dissertation proposes an exploration of the design space of a polynomial algorithm based on graph traversal. Depth-first and Breadth-first traversals are evaluated by looking the reconvergent fanout and high fanout nodes. Experimental results analyze the execution time, critical path, distribution degree and total wire length.

1. INTRODUÇÃO

A Lei de Moore estabelecida em 1968 continua válida e a cada 18 meses a densidade dos circuitos eletrônicos dobra. O desafio é como explorar os recursos, cada vez mais abundantes e complexos para otimizar a energia, o desempenho, os custos, etc. Diversas arquiteturas vêm sendo utilizadas [3], tais como processadores de propósito gerais (GPPs – *General Purporse Processors*), plataformas de arquiteturas reconfiguráveis (AR) e circuitos integrados para aplicações específicas (ASIC - *Application Specific Integrated Circuits*) e mais recentes processadores baseados em unidade (GPU).

Em um extremo, o GPP é uma arquitetura projetada para executar uma grande variedade de aplicações computacionais, ou seja, possui flexibilidade. No outro extremo, temos as arquiteturas baseadas em ASIC, que é um circuito integrado (CI) para executar tarefas específicas, cuja aplicação é diretamente implementada no *hardware*, gerando uma solução eficiente, mas com a desvantagem da inflexibilidade, pois não permitem alterações do circuito. Para implementar modificações deve-se reprojetar e refabricar o circuito[36].

As AR são uma solução intermediária, que pode ser mais eficiente que os GPPs e mais flexíveis que os ASICs, pelo fato de permitir que o *hardware* seja reprogramado adaptando-se a uma determinada aplicação. A reconfiguração pode também ser realizada em tempo de execução para explorar o paralelismo da aplicação em função dos dados de entrada e outros parâmetros em tempo de execução.

Diversas arquiteturas reconfiguráveis vêm sendo propostas e utilizadas [3]. Além disso, vários sistemas embarcados vem sendo projetados com as ARs [3, 36 e 37] como os CGRA, FPGA, etc. Os FPGA (*Field-Programmable Gate Arrays*), são dispositivos semicondutores que podem ser programados e reprogramados após sua fabricação revolucionando a forma como o *hardware* digital tem sido projetado e construído nas últimas três décadas [24]. Atualmente, os FPGAs tem uma alta

densidade e um alto desempenho, oferecendo um circuito rápido e programável de baixo custo e rápido. Os FPGAs atuais são baseados na tecnologia SRAM, e podem ser reconfigurados totalmente ou parcialmente em tempo de execução [24], proporcionando ao FPGA um alto nível de flexibilidade. Entretanto, a flexibilidade aumenta a complexidade do mapeamento motivando o desenvolvimento de novas heurísticas.

O mapeamento é uma das operações mais caras devido à complexidade dos FPGA. O mapeamento envolve dois passos: posicionamento e roteamento. Ambos são problemas NP-completo [38]. Devido à complexidade do FPGA e do problema de mapeamento, existem poucas soluções com reconfiguração dinâmica em tempo de execução [8, 9, 12, 14, 16, 23, 24, 25, 27].

Esta dissertação explora uma nova heurística de posicionamento baseada em travessia em profundidade de grafos que foi proposta em [34], onde o posicionamento de blocos lógicos é realizado em tempo de execução. O objetivo é explorar outras travessias e estudar o comportamento da heurística. O FPGA é modelado por um grafo, similar ao modelo proposto em [34]. O grafo tem uma representação implícita. A representação tem uma correspondência direta com a estrutura física do FPGA. O posicionamento é implementado por uma função de mapeamento entre dois grafos (do FPGA e do circuito a ser mapeado). A localidade dos vértices é explorada durante este processo. Diferente da proposta apresentada em [34], esta dissertação propõem a exploração da travessia em largura (BF) e/ou profundidade (DF) em ambos os grafos, modificando também o sentido, em que a travessia é executada. Além da localidade das ligações, esta dissertação diferentemente da abordagem de [34], propõe avaliar durante o posicionamento os blocos lógicos que estão interligados a vários pontos e que tem um alto grau de saída (*fanout* alto) ou que tem grau 2.

1.1 O Problema e sua Importância

Como já mencionado, as arquiteturas FPGAs podem adaptar o *hardware* para gerar soluções que exploram o paralelismo para acelerar as aplicações. Podem também gerar módulos personalizados eficientes para a aplicação em tempo de execução.

O tempo de projeto para mapear as aplicações em FPGA é um dos problemas que dificultam sua popularização. A reconfiguração em tempo de execução oferece várias possibilidades e é definida como a capacidade de modificar ou alterar a configuração funcional do dispositivo durante a operação, através de *hardware* ou *software*. E esta abordagem tem aplicações em telecomunicações reduzindo número de componentes com reutilização para várias funções [1, 2].

Para mapear as aplicações vários passos são executados. Dois deles têm um alto custo em tempo de execução: posicionamento e roteamento. O problema de posicionar uma computação nos blocos lógicos e interligá-los é NP-Completo. Várias heurísticas foram propostas. O posicionamento tem um impacto significativo sobre o desempenho do roteamento. Se o posicionamento é ruim, serão necessários muitos recursos para efetuar o roteamento e um longo tempo de execução. Em alguns casos não existe um roteamento possível para todas as conexões. Portanto a qualidade dos algoritmos de posicionamento afeta diretamente a utilização da arquitetura FPGA. Nesta dissertação, o tema abordado é o posicionamento em FPGA.

1.2 Objetivos

Esta dissertação tem enfoque no estudo de uma heurística de posicionamento e roteamento proposta em [34]. A heurística é gulosa e pode ser usada em tempo de execução. A heurística é baseada apenas na travessia em profundidade. Esta dissertação propõem simplificar a função de custo, explorar outras travessias (largura e o sentido do percurso no grafo) e a exploração dos vértices com múltiplos *fanout* e *fanout* 2.

1.3 Estrutura do Trabalho

O Capítulo 2 apresenta a fundamentação teórica, os principais algoritmos propostos para posicionamento e o algoritmo proposto em [34] que é a base para o desenvolvimento das variações propostas por esta dissertação, por ser um dos únicos algoritmos da literatura viável de ser usado em tempo de execução. O Capítulo 3 demonstra as modificações no algoritmo proposto em [34], mostrando as

contribuições desta dissertação, que permite tratar os aspectos durante o posicionamento sem aumentar o tempo de execução. O Capítulo 4 mostra os resultados obtidos. O Capítulo 5 apresenta as conclusões e as contribuições resultantes dessa pesquisa.

2. FUNDAMENTAÇÃO TEÓRICA

Neste capítulo serão abordados conceitos e definições relacionados aos problemas de posicionamento e roteamento em circuitos FPGA. O foco da dissertação é o posicionamento, mas como o roteamento é influenciado diretamente pelo posicionamento, os princípios básicos dos algoritmos de roteamento também serão abordados.

2.1 Arquitetura dos FPGA

As aplicações de circuitos FPGAs incluem processamento de sinal digital, redes, sistemas de armazenamento, aeroespaciais e sistemas de defesa, prototipagem ASIC, imagens médicas, visão computacional, reconhecimento de voz, criptografia, bioinformática, computador de emulação de *hardware*, e assim por diante.

Um FPGA é um dispositivo semicondutor pré-fabricado, composto por blocos lógicos e interconexões reconfiguráveis para gerar circuitos personalizados [1,2].

A reprogramação dos blocos lógicos e interconexões do FPGA é realizada em geral por memórias SRAM que fazem com que a funcionalidade de unidades lógicas do FPGA se adapte a cálculos específicos, estabelecendo interligações entre as unidades lógicas através de canais de roteamento pré-fabricados. Com isso, cada aplicação pode ser mapeada em um FPGA, programando com base nas características específicas, gerando uma solução de alto desempenho sem perder em flexibilidade.

Apesar do posicionamento e roteamento serem problemas estudados a mais de três décadas, as pesquisas atuais relacionadas ao mapeamento e desenvolvimento de algoritmos de eficientes [24, 25, 29, 30, 31, 32, 33, 34] vem mostrando a importância destes problemas para o desenvolvimento de ferramentas para FPGA.

Como já mencionado, o posicionamento é importante, pois a sua ineficiência afeta diretamente o mapeamento e pode impossibilitar o roteamento do circuito.

Os FPGA permitem a prototipagem rápida, reduzindo o *time-to-market* no desenvolvimento de novos produtos, e são cada vez mais usados para programar qualquer função lógica como um circuito ASIC [16]. Seus componentes lógicos programáveis são fisicamente conectados em uma hierarquia reconfigurável para executar funções combinacionais complexas. Além disso, os FPGA tem suporte a conexões em cascata propiciando sua utilização para operações aritméticas de alta velocidade [1, 2].

O FPGA é uma matriz de blocos lógicos configuráveis (CLBs – *Configurable Logic Blocks*) cercados por recursos de roteamento (canais de roteamento vertical e horizontal – *routing chan*nels) e elementos de chaveamento ou *routing switches* (conjunto de chaves de roteamento ou roteadores ou comutadores). Os CLBs implementam as funções lógicas. Neste trabalho considera-se CLBs com um único bloco lógico ou *lookup table* – LUT. Os recursos de roteamento são conectados às entradas e saídas (IOBs – *input/output blocks*). Os IOBs estão posicionados nas extremidades da matriz e são usadas como terminais de entrada, saída ou bidirecional, fazendo a interface com os dispositivos externos. Em geral, todos os canais de roteamento têm a mesma largura (número de fios) [4, 5, 6]. A Figura 1 ilustra um FPGA, onde L representa os CLBs; I/O *cell* representam os IOBs; S representam os roteadores ou comutadores; e C representam os canais de roteamento horizontal e/ou vertical.

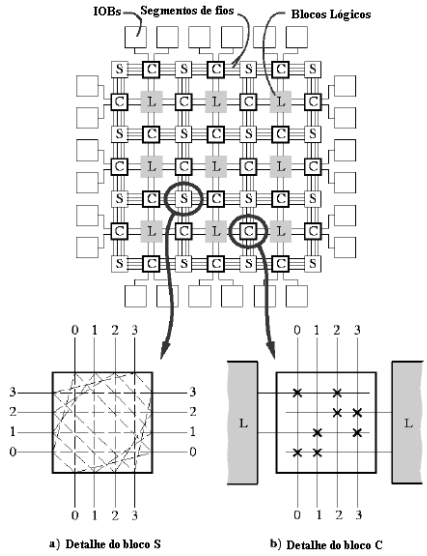


Figura 1: Estrutura FPGA
Fonte: CARVALHO. Heurísticas para a fase de Roteamento de Circuitos Integrados Baseados em FPGAs. p. 17.

As LUTS (Figura 2) podem executar qualquer função lógica de k-entradas e em geral possuem um FF (flip-flop) na saída que permite implementar circuitos sequenciais. Nesta dissertação iremos considerar apenas circuitos combinacionais. A função da LUT é armazenado em uma SRAM (que é como uma tabela verdade da função). A função do FPGA é programada através dessas memórias e das memórias que controlam as interconexões. O bloco de conexão (*connection block*), Figura 1(b), conecta os pinos das LUTs aos canais de roteamento horizontal ou vertical. Os canais são conectados através de um roteador ou comutador (Figura 1(a)) [7].

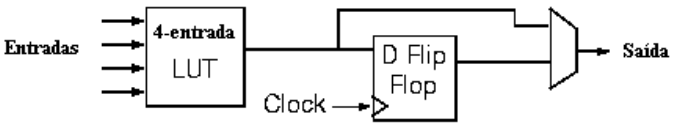


Figura 2: Estrutura do Bloco Lógico
Fonte: BETZ. FPGA Architecture for the Challenge

As conexões de roteamento do FPGA (Figura 1(b)) ficam localizadas em roteadores ou comutadores que fazem a conectividade dos segmentos de fios horizontais/verticais de uma direção com os segmentos horizontais/verticais nas outras direções (leste, oeste, norte e sul). Cada comutador contém várias chaves programáveis e necessita de vários *bits* para configurá-los. Os canais de roteamento são segmentados, entre os comutadores. Assim para uma LUT se conectar a outra LUT é necessário passar um segmento de fio do canal adjacente, depois passar por

um comutador e posteriormente outro segmento de fio para chegar a LUT mais próxima, no melhor caso. O roteamento de um dado circuito consiste na programação dos fios, blocos de conexão e comutadores para gerar as interligações das LUTs [8, 9] e possibilitar a implementação de qualquer função lógica.

A capacidade de roteamento está diretamente relacionada à topologia dos comutadores que conectam os canais verticais e horizontais. Infelizmente, um comutador que possibilite todos os padrões de conexão é inviável devido ao alto custo, por ter restrições de roteamento entre seus pinos de entrada e saída. Em geral cada fio de entrada em uma face do comutador é conectada apenas a um fio em cada uma das outras faces. A Figura 1(a) ilustra o *switch* de Wilton [17] que é um padrão de comutador, no qual as linhas verticais e horizontais são ligadas diretamente, mas a ligação com uma mudança vertical para horizontal ou vice-versa segue um padrão módulo circular [17].

Chaves programáveis conectam cada pino de entrada/saída da LUT ou IOB a um ou mais segmentos de fiação no canal adjacente. Depois o fio é conectado ao comutador, onde as chaves programáveis ou multiplexadores permitem a conexão com outros canais. A Figura 1(b) ilustra as chaves, os canais segmentados e LUTs. Neste exemplo, cada entrada ou saída da LUT pode se conectar apenas a dois fios do canal e o canal possui 4 fios ou trilhas. Nesta dissertação será considerada uma arquitetura de FPGA na qual a LUT tem 4 entradas, uma em cada lado que podem se conectar a qualquer fio do canal adjacente. A saída da LUT fica na borda sul e só pode conectar a este canal. Esta restrição é devido ao FPGA que foi modelado para teste. Mas a abordagem pode ser usada em qualquer outro FPGA com restrições de conexões diferentes.

O projeto de FPGA segue um fluxo semelhante ao projeto de circuito integrados passando por um fluxo de ferramentas CAD (*Computer-Aided-Design*), como ilustrado na Figura 3.

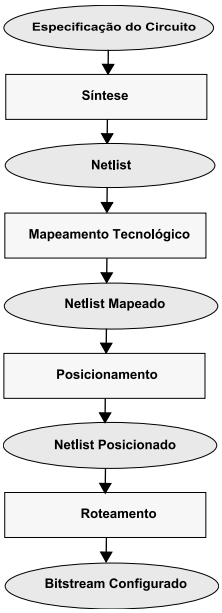


Figura 3: Típica ferramenta de fluxo CAD FPGA

A descrição do circuito inicializa todo o processo referente ao projeto físico. Pode ser feita em uma linguagem de alto nível ou em uma linguagem de *hardware* como VHDL ou Verilog. A síntese gera uma descrição estrutural do circuito com um conjunto *de* funções lógicas e *flipflops*.

Na fase de mapeamento tecnológico, dado a tecnologia alvo, a descrição do circuito é convertida em um *netlist* com as restrições de funções disponíveis na tecnologia alvo. No caso desta dissertação será um conjunto de LUT com k-entradas, onde k = 4, ou seja, o circuito deve ser descrito usando apenas LUT com k-entradas. Posteriormente, a descrição em LUT é posicionada nos blocos lógico do circuito alvo, no nosso caso será um FPGA. O posicionamento evita em geral as conexões de longa distância, reduzindo assim atrasos excessivos entre elas. Um posicionamento ruim pode gerar uma descrição onde o roteamento físico não seja possível. O roteamento gera a programação dos comutadores para interligar os blocos já posicionados no passo anterior. Após o roteamento, os bits de configurações que irão programar o FPGA são gerados (*bitstream*).

Como já mencionado, esta dissertação tem foco no problema de posicionamento que é uma das últimas etapas do fluxo de ferramentas CAD no projeto de FPGA. A seguir, um contexto histórico dos principais trabalhos será apresentado, seguido da descrição do algoritmo proposto em [34] que é a base dessa dissertação.

2.2 Trabalhos Relacionados

Devido à alta complexidade e tempo envolvidos em gerar e reconfigurar um FPGA por completo, nas duas últimas décadas surgiram FPGA parcialmente reconfiguráveis. A reconfiguração parcial permite que apenas uma parte do circuito seja modificado, reduzindo o tempo para geração e carregamento dos bits de configuração (*bitstreams*), dando ao FPGA a capacidade de se adaptar a novos contextos em tempo de execução (ou *online*).

No cenário *online*, encontrar e gerenciar espaços vazios disponíveis permite um maior reuso do FPGA. Como se trata de situações dinâmicas para serem usadas em tempo de execução, as etapas de posicionamento e roteamento devem ser implementadas de forma eficiente. A abordagem [26] apresenta uma solução para realizar a deslocação/realocação dinâmica de blocos lógicos e suas interligações sem interromper a execução, resolvendo o problema de fragmentação dos espaços livres no FPGA através de trocas rápidas de contexto. Um algoritmo de posicionamento baseado no algoritmo *staircase* foi proposto em [27]. Abordagens recentes [24, 28] propõem reduzir a fragmentação da reconfiguração parcial usando um *hardware* virtual reconfigurável. Em [24] foi apresentado um novo *framework* de compilação JIT (Just-In-Time). O *bitstream* para um FPGA virtual é calculado de forma rápida. O posicionamento e o roteamento são gerados em tempo de execução, através de um posicionamento inicial permitindo a reutilização dos recursos de *hardware* no FPGA virtual que pode estar mapeado sobre diferentes FPGA reais. A vantagem da virtualização FPGA [28] é que as etapas, de posicionamento e roteamento são independentes do hardware físico, beneficiando da capacidade da reconfiguração dinâmica em tempo de execução, permitindo também que FPGAs que não suportam esse recurso, ou seja, o FPGA virtual, possa ser usado por qualquer tipo de FPGA.

A maior parte dos algoritmos de posicionamento propostos na literatura são implementados em tempo de compilação ou síntese do circuito, isto é, em um cenário estático. O principal foco é o mapeamento do circuito como um todo. Com o aumento da complexidade dos projetos e da disponibilização de FPGA mais complexos, os desafios são o posicionamento de circuitos com mais de 100.000 blocos. Na maioria dos algoritmos, o tempo de execução não é a primeira prioridade e a execução pode demorar horas ou até mesmo dias. Recentemente, o uso da

computação paralela e/ou o uso de GPU vem sendo proposto para reduzir o tempo de execução dos algoritmos de posicionamento.

O VPR é um *framework* acadêmico proposto pela Universidade de Toronto que permite a avaliação e teste de arquiteturas de FPGA e de algoritmos de posicionamento e roteamento. A maioria dos algoritmos usa com base para comparação à ferramenta VPR [5,6].

Em [29] foi proposta uma abordagem paralela determinística para o algoritmo de posicionamento *simulated annealing* (SA). A metaheurística SA é muito usada em várias soluções de posicionamento. Entretanto, o tempo de execução pode ser elevado. A implementação proposta em [29] avalia vários movimentos em paralelo. Essa abordagem acelera de 2.8x a 3.7x a solução do VPR. Em [30] é proposto um algoritmo DAST (*Dynamically Adaptive Stochastic Tunneling algorithm*) para evitar o problema de "congelamento" em uma abordagem SA usando um esquema de detecção de aprisionamento de mínimos locais com base em DFA e uma programação dinâmica adaptativa para o problema de posicionamento em FPGAs, mostrando uma redução de 18,3% no tempo de execução de alguns *benchmarks* em comparação com a ferramenta VPR.

Uma abordagem baseada no algoritmo Star+ foi proposta em [31] com o intuito de desenvolver uma solução escalável com um método analítico independente do tamanho da rede, que pode ser resolvido por um sistema de equações não-lineares. Os resultados mostraram uma redução do caminho crítico em 8-9% e uma aceleração no tempo de execução de 5x em comparação com o VPR no seu modo rápido.

Como já mencionado, o tempo de posicionamento e roteamento para circuitos com mais de 100k LUT é elevado, podendo chegar a horas, ou até mesmo dias. E infelizmente, as abordagens de SA não são adequadas neste cenário. Abordagens de posicionamento de circuitos complexos foram propostas em [32, 33]. Como o foco desta dissertação são algoritmos de posicionamento viáveis em tempo de execução onde os circuitos em geral são formados módulos que ocupam centenas de blocos ou poucos milhares, as abordagens anteriores não são adequadas. Recentemente, uma abordagem baseada em uma solução gulosa [34] gerou um ganho de aceleração de três ordens de grandeza em relação ao VPR, tornando-se sua utilização viável em tempo de execução. Esta abordagem será detalhada a seguir e é a base para o desenvolvimento desta dissertação.

2.3 Heurística Baseada em Profundidade

Várias técnicas e abordagens para a reconfiguração parcial dinâmica foram desenvolvidas recentemente [24, 25, 26, 27, 28, 29, 30, 31, 32, 33]. Para tirar proveito do conceito de reconfiguração parcial dinâmica se faz necessário adaptar os algoritmos com o intuito de reduzir o tempo de posicionamento para circuitos FPGAs. Entretanto, o tempo de execução ainda permanece elevado para um cenário online.

Na abordagem proposta em [34], descrita em detalhes nesta seção, tanto o FPGA quanto o circuito a ser mapeado são representados por grafos. A representação do grafo FPGA tem uma correspondência direta com a sua estrutura física. O posicionamento é implementado por uma função de mapeamento entre os dois grafos. Assim, a localidade dos vértices é explorada durante este processo de posicionamento e é implementada com uma travessia em profundidade (DF). Além do problema de posicionamento, a heurística proposta em [34], apresenta também um algoritmo de roteamento guloso baseado no algoritmo de roteamento XY *Network-on-chi*p (NoC) [39, 40]. Esta solução abriu novas possibilidades a serem exploradas. Nesta dissertação iremos propor variações no algoritmo de [34]. O foco será apenas na parte de posicionamento do algoritmo. Inicialmente, iremos descrever a solução proposta em [34] e nos capítulos seguintes iremos apresentar as modificações e resultados gerados por esta dissertação.

O termo vértice será usado ao longo dessa seção para se referir ao grafo do circuito ou grafo de LUTs. Um exemplo é ilustrado na Figura 4. O termo nó será usado para o grafo que representa o FPGA. A Figura 5 ilustra um FPGA como uma matriz quadrada. O grafo FPGA é composto por quatro tipos de nós: LUT, SW (comutadores), entrada/saída (IOBs) e canais de trilhas ou segmentos de fios. As LUTs são indicadas pelos índices de linha e coluna. Para maiores informações sobre os canais de segmentação e *switchbox*, consultar [17], para maiores detalhes do comutador e para detalhes do modelo do grafo consultar [34].

O algoritmo [34] faz a travessia em profundidade no grafo do circuito priorizando o caminho crítico, começando nas saídas do circuito em direção as entradas, o algoritmo mapeia os vértices LUTs para o(s) nó(s) LUTs. Os nós SW e os canais de fios realizam as ligações. Portanto, uma aresta do grafo do circuito é mapeada em um conjunto de nós SW e nós segmentos.

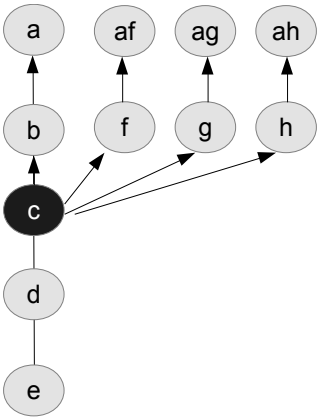


Figura 4: Exemplo grafo de LUT k-entrada, k máximo é 4.

Um dos pontos que pode ser explorado para melhorar os resultados em termos da qualidade da solução [34] (principalmente em relação ao total de segmentos e comutadores usados), é a exploração de vértices com múltiplas saídas. Como mencionado, o posicionamento é baseado em profundidade e prioriza o caminho crítico. Considere o exemplo da Figura 4. Ao percorrer o caminho e→d→c→b→a, considere o posicionamento do vértice c. O algoritmo [34] irá posicionar c em uma posição adjacente ao vértice b, uma vez que a primeira visita ao vértice determina seu posicionamento.

Entretanto, o vértice c tem múltiplas saídas (alto grau ou *fanout*) e está conectada aos vértices f, g e h. Como a posição de f, g e h será determinada pelos vértices antecessores (af, ag, e ah), eles podem ficar distantes de c. Como o algoritmo [34] prioriza o caminho crítico inicial, os vértices de múltiplas saídas não são detectados e serão objetos de estudo dessa dissertação.

A Figura 5 mostra uma possibilidade de posicionamento para o caminho e→d→c→b→a. Observa-se que todos os vértices ficam adjacentes e o roteamento deste caminho tem um custo baixo os nós são adjacentes no FPGA.

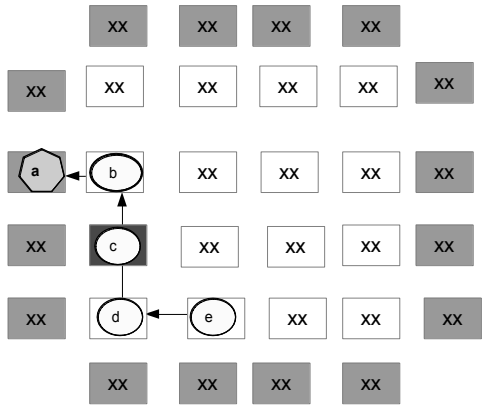


Figura 5: Possibilidade de posicionamento e-a.

A Figura 6 apresenta uma possibilidade de posicionamento para os nós af, ag e ah, e seus descendentes f, g e h. Neste exemplo, vértices f, g e h são também saídas do vértice c, ficaram distantes de c e próximos de seus antecessores.

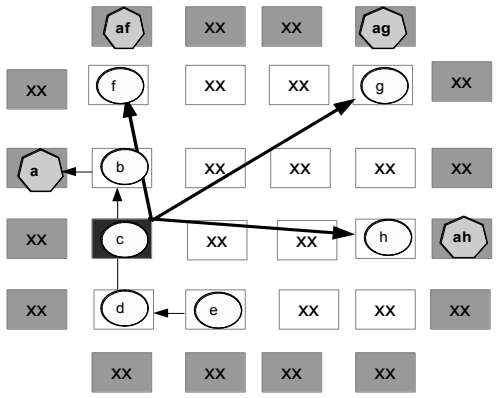


Figura 6: Possibilidade de posicionamento dos vértices de saída do vértice c.

A Figura 7 apresenta uma possibilidade de roteamento. Vamos supor um custo simples, na qual a distância pode ser medida para distância em X e Y. O custo da aresta c→b é 1. Já as outras arestas c→f, c→g e c→h terão custo 3, 4 e 2 respectivamente.

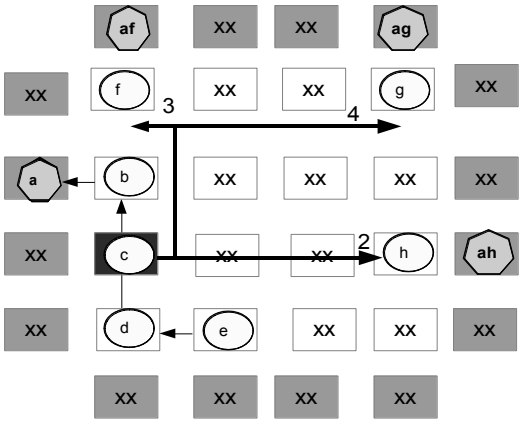


Figura 7: Possibilidade de roteamento com custo simples.

A Figura 8 apresenta o grafo original do circuito etiquetado com o custo do roteamento das arestas. Vale ressaltar que os caminhos que passam por c→f ou c→g ou c→h podem agora se tornar críticos, pois os nós f, g e h estão posicionados distantes de c.

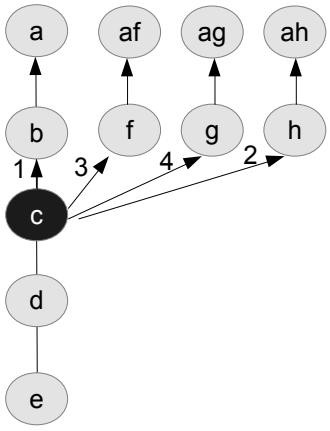


Figura 8: Grafo etiquetado com custo.

Outro problema é a complexidade da rotina de roteamento quando existem vértices com múltiplos *fanout* com um posicionamento distante. Em [34], foi realizado um experimento para comparar o tempo de execução do roteamento VPR [5] para um dado posicionamento. Dois posicionamentos foram avaliados: o posicionamento em profundidade [34] e posicionamento do VPR para caminho crítico. Os resultados mostraram que o posicionamento [34] não deteriora o caminho crítico, porém o tempo para executar o roteamento foi 10x para os *benchmarks* maiores (como PDC e SPLA que possuem em torno de 3000 blocos lógicos) em comparação com o tempo necessário para o roteamento do posicionamento gerado pelo VPR.

Esta dissertação propõe fazer a travessia no sentido inverso, detectar vértices com grau alto ou com grau 2 durante a travessia para implementar estratégias que priorizem o posicionamento deles e de seus descendentes em regiões próximas.

A abordagem de [34] faz a travessia em profundidade no sentido saída→entrada faz o posicionamento e roteamento ótimo de 35% das arestas, pois explora a localidade dos vértices de *fanout* 1.

Em geral, para os circuitos avaliados em [34], cada vértice tem um grau médio de saída 3. Como a posição é determinada, pela primeira visita, 33% das arestas são roteadas diretas se existem nós adjacentes. Esta localidade que é explorada pela abordagem proposta em [34]. A questão é se existem outras propriedades que podem ser exploradas, onde iremos investigar as travessias em largura e os múltiplos *fanout* nesta dissertação.

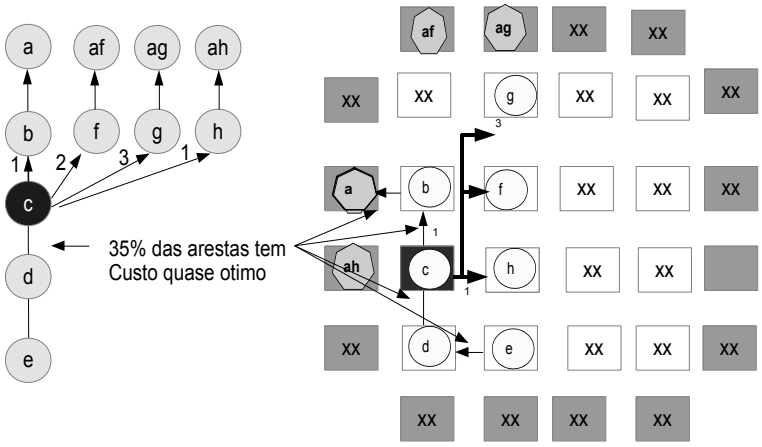


Figura 9: Possibilidade de posicionamento com custo ótimo.

Entretanto, o grau de saída não é uniformemente distribuído como mostram os experimentos em [34]. Poucos vértices têm alto grau e muitos tem baixo grau, como será mostrado com os experimentos de histogramas que serão apresentados no Capitulo 4. Em geral mais de 60% dos vértices tem grau menor ou igual a 2. Os vértices com alto grau que podem ser por exemplo 5% vértices e ao mesmo tempo podem concentrar 30% das arestas ou mais, e quando estas arestas são mapeadas em segmentos e comutadores, o custo das conexões pode consumir 70% dos recursos de roteamento. Em [34], o uso de fios longos é avaliado para reduzir os custos de arestas com múltiplos *fanout*. FPGA costuma oferecer estes recursos para reduzir atrasos.

A Figura 10 apresenta uma alternativa avaliada e proposta nesta dissertação de realizar a busca em largura (BF) no lugar de realizar a busca em profundidade

proposta em [34]. Além disso, o percurso é realizado no sentido inverso, das entradas em direção às saídas. A busca em profundidade (DF) também pode ser explorada no sentido entrada-saída, que é outra contribuição desta dissertação.

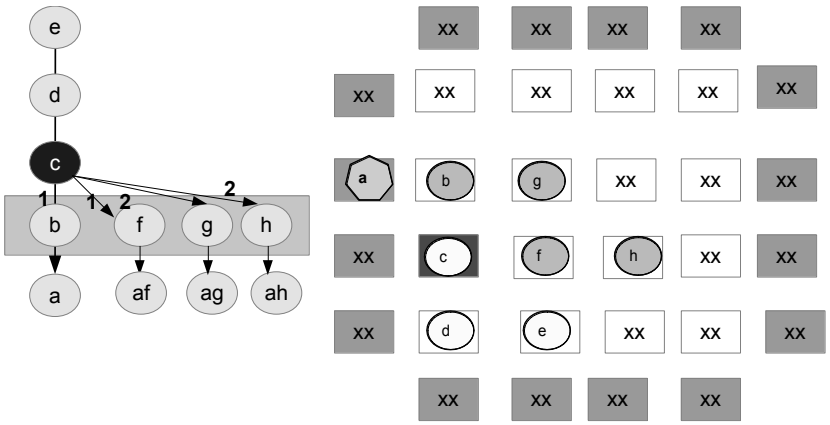


Figura 10: Possibilidade de posicionamento BF.

A proposta apresentada nesta dissertação utiliza o mesmo FPGA virtual que foi usado no algoritmo proposto em [34]. Além da travessia no sentido inverso (entrada→saída) priorizando os blocos com um grau de saída elevado, será realizada a busca em profundidade (DF) e a busca em largura (BF) para mapear o grafo do circuito e do FPGA. Outras modificações no algoritmo [34] também foram implementadas.

3. POSICIONAMENTO EM LARGURA E PROFUNDIDADE

3.1 Introdução

Como já mencionado, o problema de posicionar uma computação nos blocos lógicos e interligá-los em um FPGA é NP-Completo [38]. Esta dissertação propõe explorar variações de uma heurística de posicionamento proposta em [34]. As semelhanças e diferenças em a abordagem desta dissertação e do algoritmo proposto originalmente em [34] serão destacadas ao longo deste capítulo.

Inicialmente, a seção 3.2 apresenta exemplos do mapeamento do grafo do circuito baseado nas travessias em largura e profundidade. Nesta dissertação o sentido da travessia é o inverso do proposto em [34]. A Seção 3.3 apresenta duas estratégias que exploram os vértices de alto grau e de *fanout* 2, respectivamente. As modificações na função custo-proposta nesta dissertação são detalhadas nas seções 3.4 e 3.5, assim como modelo do grafo FPGA que foi proposto em [34].

3.2 Mapeamento

Como já mencionado, diferente da abordagem proposta em [34], esta dissertação propõe percorrer o grafo no outro sentido, partindo das entradas em direção as saídas. A Figura 11 ilustra um exemplo de um grafo representado por um grafo de LUT de k-entradas. As entradas externas (entradas e saídas) são representadas pelos vértices em cinza e as LUTs pelos vértices em branco. Neste exemplo k = 4 mas apenas o vértice 13 tem 4-entradas. Como citado anteriormente, o exemplo é simples, apenas as entradas tem um grau de saída maior que 1.

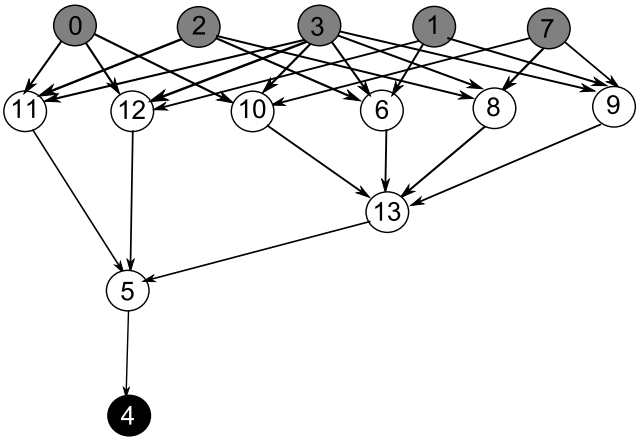


Figura 11: Exemplo grafo de LUT k-entrada, k máximo é 4.

As propostas apresentadas em [3] e [34] utilizam o P&R (posicionamento e roteamento) baseado na travessia dos grafos. Esta dissertação se limita apenas ao problema de posicionamento em FPGAs sem implementar o roteamento. O roteamento é feito com o VPR [5].

O exemplo da Figura 11 será usado ao longo deste capítulo para mostrar os percursos durante o mapeamento baseados na travessia em profundidade ou *Depth-Fisrt* (DF) e na travessia em largura ou *Breadth-First* (BF).

3.2.1. Mapeamento em DF

Semelhante à abordagem [34], durante a travessia em DF, os vértices descendentes são ordenados pelo maior caminho que possa alcançar as saídas, priorizando o caminho crítico durante a travessia. Suponha que na travessia em DF o vértice 0 foi o primeiro a ser selecionado. Este vértice possui três descendentes: 10, 11 e 12. A travessia que possui a maior distância das saídas passa pelo vértice 10 e seus descendentes. Assim, o vértice 10 será o primeiro a ser visitado para priorizar o caminho crítico durante o mapeamento que seguirá a ordem de 0→10→13→5→4, iniciando na entrada 0 e terminado na saída 4.

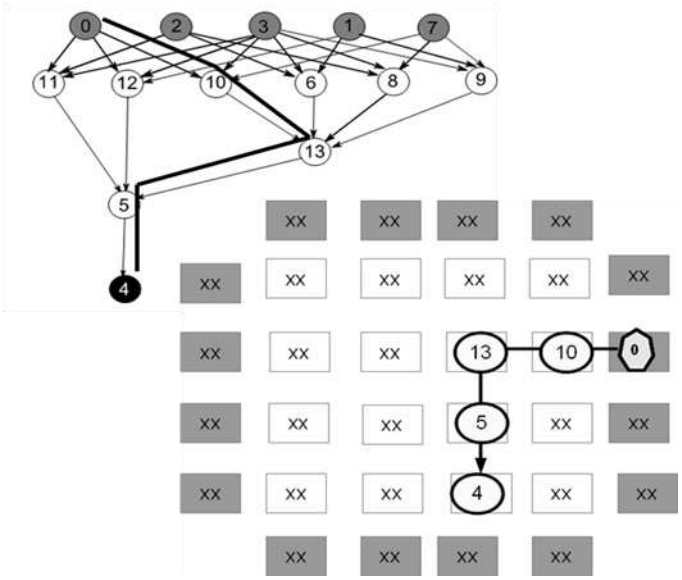


Figura 12: Primeira etapa de DF: posicionamento do vértice entrada 0.

Na travessia em DF, o algoritmo percorrer aresta por aresta em profundidade no caminho 0 a 4. As seguintes arestas serão percorridas 0→10, 10→13, 13→5, 5→4. Primeiro a entrada 0 é posicionada em uma entrada/saída (IOB) do FPGA. Posteriormente, cada aresta ***a*** → ***b*** percorrida irá posicionar o vértice destino **b** em profundidade no grafo do FPGA, ou seja, próximo ou adjacente ao vértice **a**, como proposto no algoritmo [34] porém no sentido inverso (entrada → saídas). A Figura 12 ilustra o resultado do posicionamento, onde podemos observar que é dada prioridade a posições que possuem menor custo de posicionamento (próximas). Seguindo a ordenação em DF, como ilustrado na Figura 13, a próxima aresta é a aresta 0 → 11, onde o vértice destino 11 é posicionado próximo do vértice 0. Como a aresta de saída do vértice 11 leva ao vértice 5, que já foi visitado e posicionado, a próxima aresta é 0 → 12 que determina a posição do vértice 12. Como todas os descendentes da entrada 0 já foram visitadas, a busca em DF deve recomeçar por outra entrada não visitada.

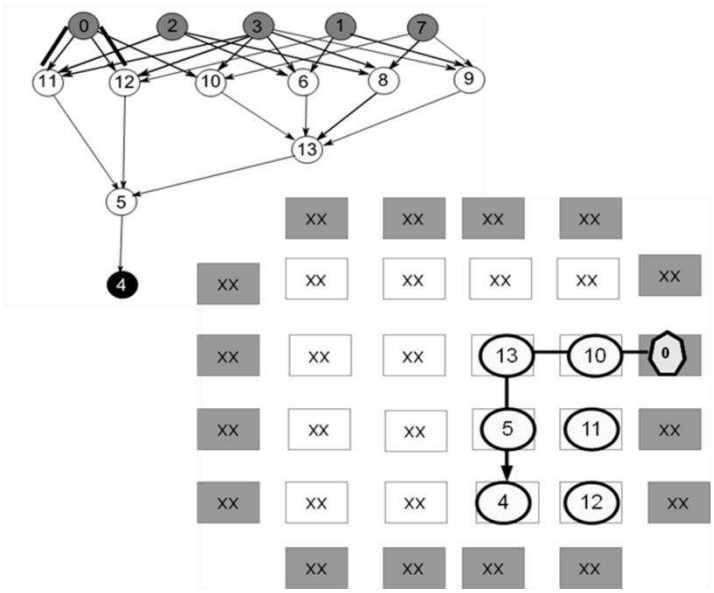


Figura 13: Segunda etapa de DF: posicionamento dos vértices 11 e 12.

Continuando, suponha que a DF recomece pela entrada 1 que possui três arestas: 1 → 12, 1 → 6 e 1 → 9. O vértice 1 é posicionado em uma das bordas vazias e os vértices 6 e 9 são posicionados nas proximidades do vértice 1. O vértice 12 já está posicionado. O mesmo acontece com as arestas 6→13, 9→13, pois o nó 13 já foi visitado. A Figura 14 ilustra essa etapa do posicionamento.

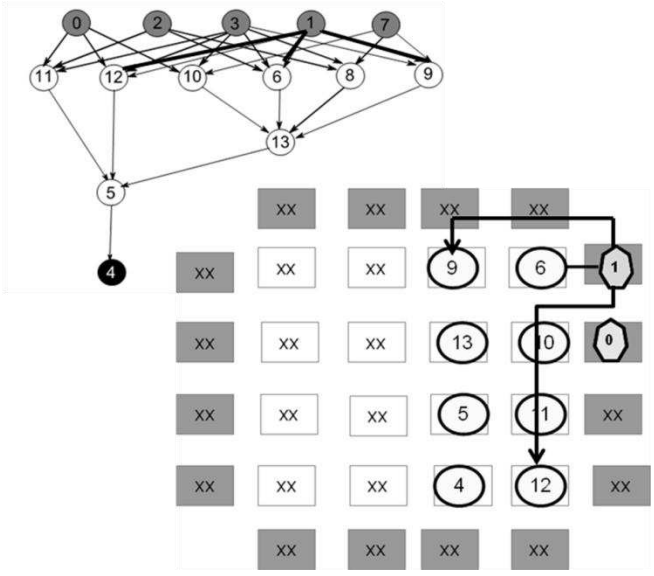


Figura 14: Etapa Parcial de DF: posicionamento do vértice entrada 1 e seus descendentes.

A DF recomeçará novamente por uma entrada não visitada, por exemplo, pelo vértice 2, que possui três arestas: 2 → 11, 2 → 8 e 2 → 6. O vértice 2 é posicionado

em uma das bordas vazias. Dentre suas arestas, apenas o vértice 8 ainda não foi posicionado. Podemos observar que como não existem nós próximos ao nó 2. A Figura 15 ilustra essa etapa de mapeamento, onde o vértice 8 fica a duas colunas de distância da entrada 2, que é a primeira posição livre nas proximidades da entrada 2.

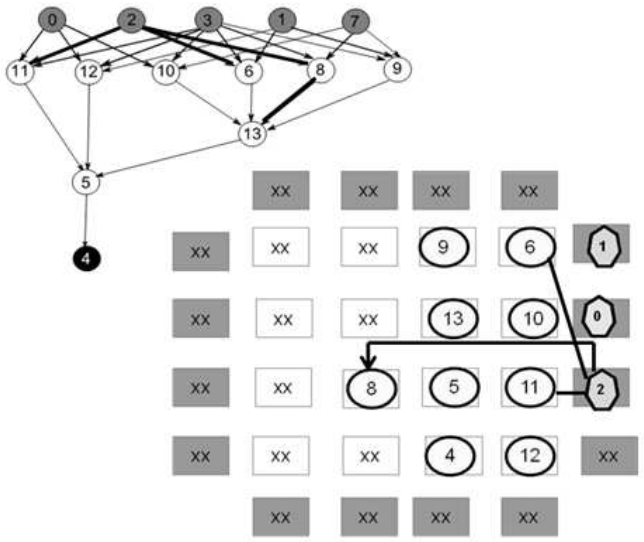


Figura 15: Etapa Parcial de DF: posicionamento do vértice entrada 2 e seus descendentes.

Restam ainda as entradas 3 e 7. Todos os seus descendentes do vértice 3 já foram posicionados. A Figura 16 ilustra a distância do nó 3 e seus descendentes que já foram posicionados segundo as arestas de outros predecessores.

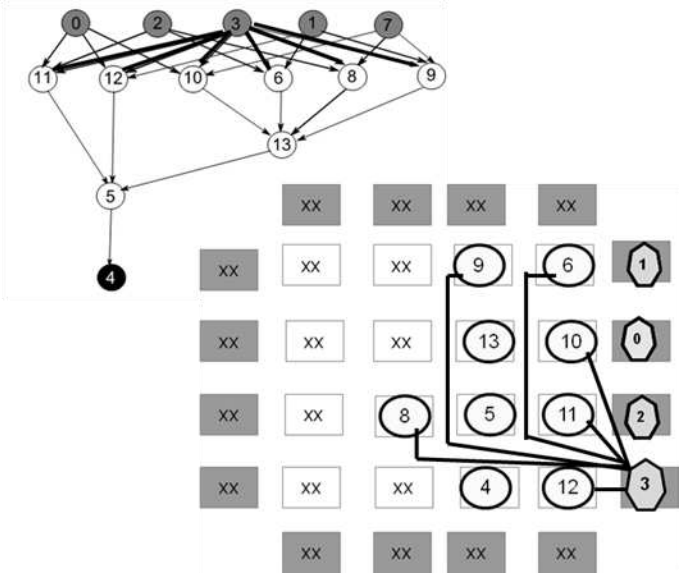


Figura 16: Etapa Parcial de DF: posicionamento do vértice entrada 3 e seus descendentes.

Resta apenas o vértice 7 que será posicionado em uma das bordas vazias. Seus descendentes (10, 8 e 9) já foram posicionados. A Figura 17 mostra a distância do nó 7 e seus descendentes.

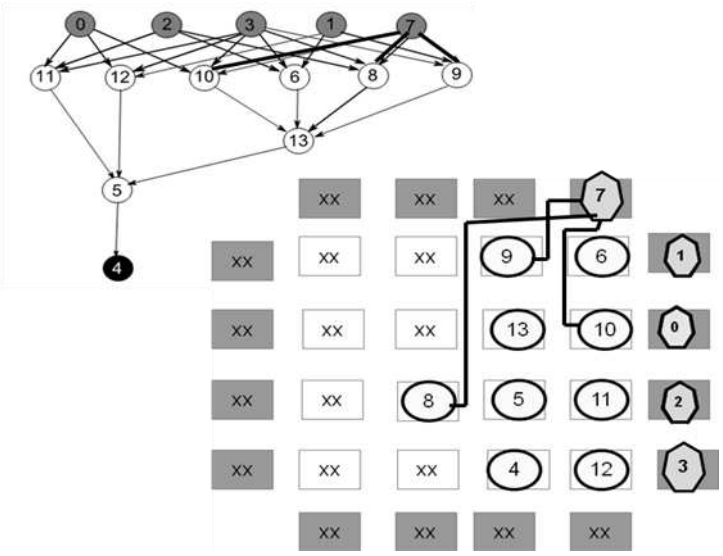


Figura 17: Etapa Parcial de DF: posicionamento do vértice entrada 7 e seus descendentes.

Finalmente, após o posicionamento de todos os vértices e seus descendentes, é possível estimar o custo total do roteamento. A Figura 19 mostra o custo estimado em número de conexões para cada aresta. A soma dos custos das conexões usadas para mapear todas as arestas é 79, para este posicionamento. Esta medida é referenciada por CTER (Custo Total Estimado para o Roteamento). O custo médio de conexões por aresta é 3,03. Assim, o vértice 3 que possui o mais alto grau concentra 23,07% do total das arestas e 31,64% dos recursos de roteamento.

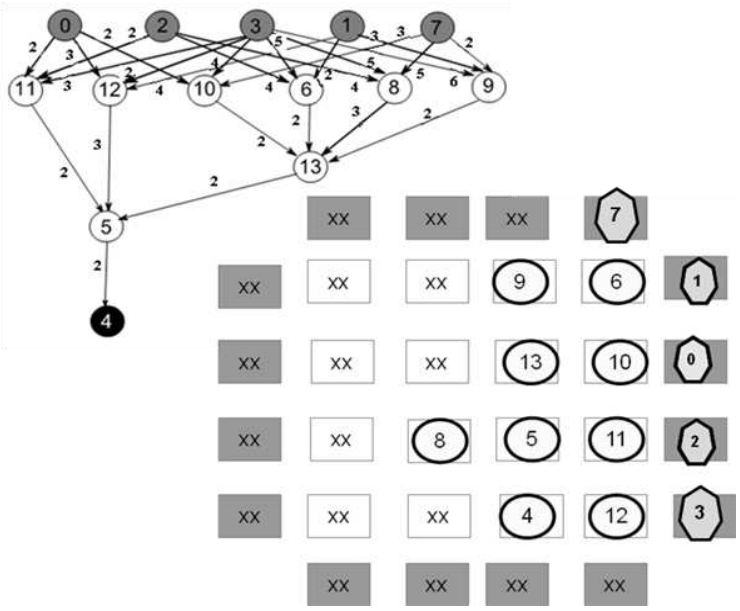


Figura 18: Etapa final de DF: Custo Total Estimado para o roteamento.

3.2.2. Mapeamento em BF

Além da busca em profundidade, esta dissertação explora a busca em largura. Na travessia do grafo em BF, o algoritmo irá priorizar os nós com maior grau de saída (*fanouts*) para estabelecer uma ordem parcial na travessia. A travessia começa pelo vértice de maior grau. Na ordenação das arestas adjacentes ao vértice, o critério será o grau de saída dos vértices destino. Por exemplo, se o vértice **a** possui três arestas: a → b, a → c e a → d. Suponha que o grau de saídas seja 2, 3 e 4 para b, c e d, respectivamente. Os vértices com maior grau de saída que serão visitados, seguindo a ordenação decrescente do grau de saída: d, c, e b para este exemplo, durante a busca em largura.

Considere novamente o grafo da Figura 11. A Figura 19 mostra o posicionamento parcial partindo do vértice 3, que possui o maior grau de saída (*fanout*) com 6 arestas. A busca em largura irá posicionar todos os seus descendentes (6, 8, 9, 10, 11, 12) em nós LUTs próximas a entrada 3.

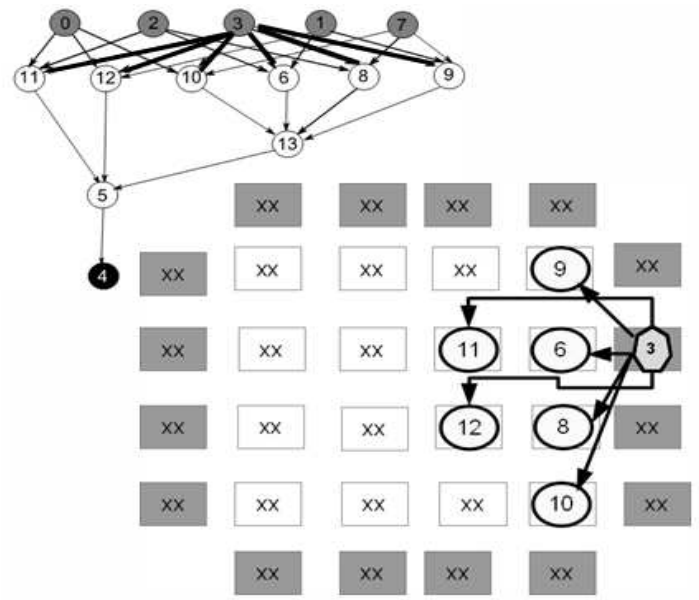


Figura 19: Etapa Parcial de BF: posicionamento do vértice entrada 3 e seus descendentes.

A travessia continua em largura. Como todos os descendentes do vértice 3 tem o mesmo grau, existem várias possibilidades. O algoritmo é guloso e só explora uma delas. Semelhante ao mapeamento DF, cada aresta $\boldsymbol{a} \rightarrow \boldsymbol{b}$ é visitada e a posição do vértice destino $\boldsymbol{b}$ é determinada pelo posicionamento do vértice $\boldsymbol{a}$. A primeira vez que o vértice é visitado que irá determinar sua posição. Por exemplo, se o descendente 9 do vértice 3 é o primeiro na busca em largura, a visita a aresta 9 → 13 irá determinar a posição do vértice 13. As outras arestas: 6→13, 8→13, 10→13 não influenciam no posicionamento de 13, como mostra a Figura 20.

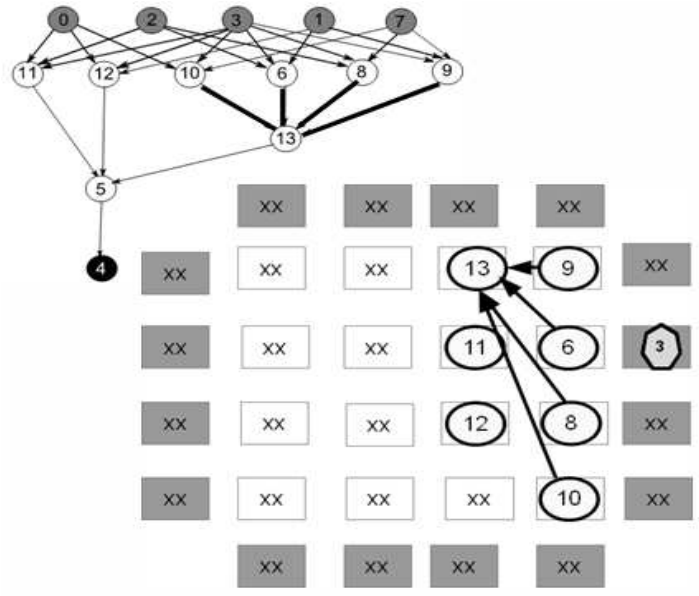


Figura 20: Etapa Parcial de BF: posicionamento do nó 13.

Seguindo em largura, a aresta 12→5 é visitada e posiciona o vértice 5 em uma LUT próxima do vértice 12. As arestas seguintes não influenciam no posicionamento pois o vértice 5 já está posicionado: 11 → 5 e 13→5 (veja a Figura 21). Resta apenas a aresta 5 → 4, que atinge o vértice de saída 4, conforme ilustra a Figura 22.

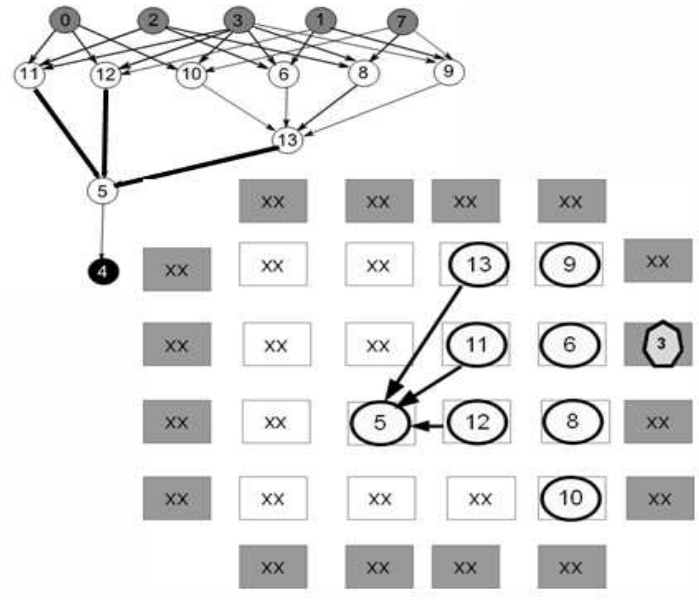


Figura 21: Etapa Parcial de BF: posicionamento do nó 5.

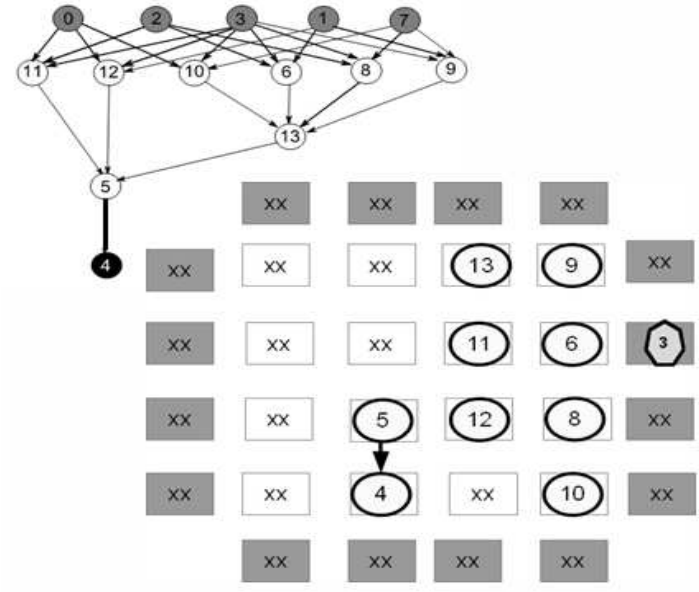


Figura 22: Etapa Parcial de BF: posicionamento do vértice de saída 4.

Continuando, a BF recomeça com o próximo vértice de entrada. Como os quatro vértices restantes têm o mesmo grau de saída (3 no exemplo), eles poderão ser

visitados em qualquer ordem. Suponha que o primeiro seja o vértice 7 que possui três descendentes: 8, 9 e10. O vértice de entrada 7 é posicionado em uma das bordas vazias e somente verifica que se seus nós descendentes já estão posicionados. A Figura 23 ilustra essa etapa de mapeamento.

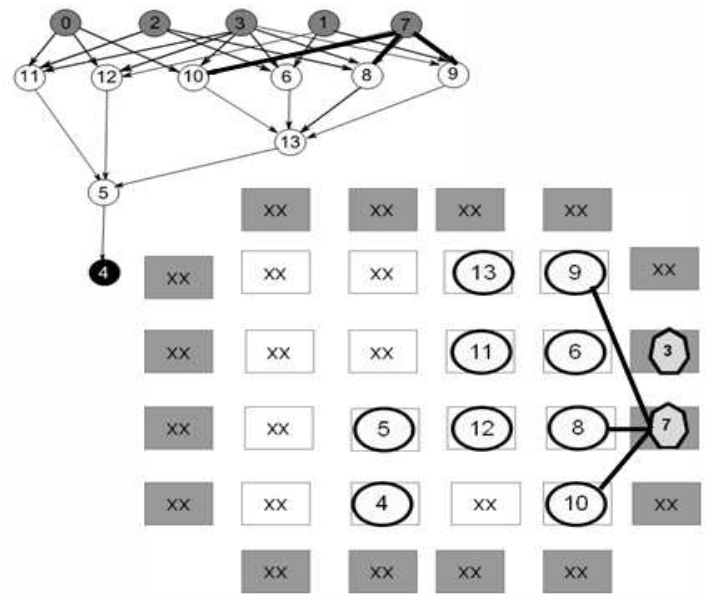


Figura 23: Etapa Parcial de BF: posicionamento do vértice 7.

Prosseguindo, a BF recomeça com o próximo vértice, por exemplo, pelo vértice 2 e que possui também três descendentes: 6, 8 e 11. O vértice 2 é apenas posicionado em uma das bordas vazias, já que seus descendentes já foram posicionados como ilustrado na Figura 24.

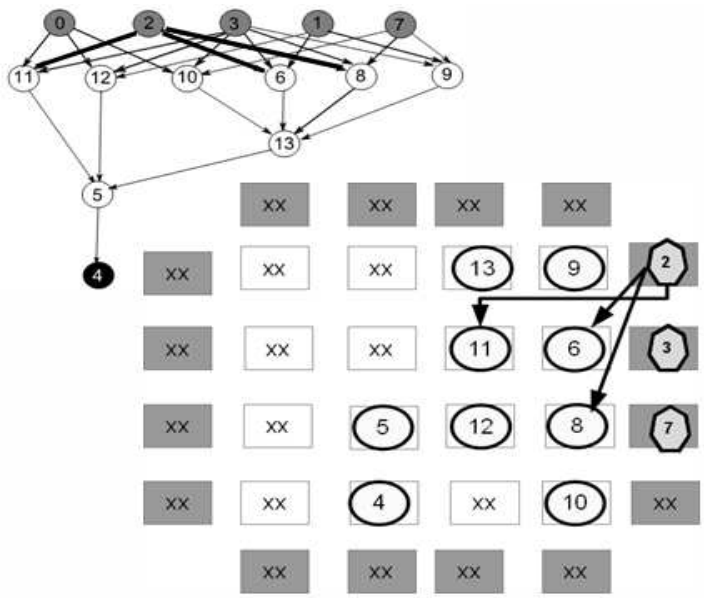


Figura 24: Etapa Parcial de BF: posicionamento do vértice de entrada 2.

Suponha que a próxima entrada seja o vértice 1, que possui também três descendentes já visitados: 6, 9 e 12. A Figura 25 ilustra essa etapa de mapeamento.

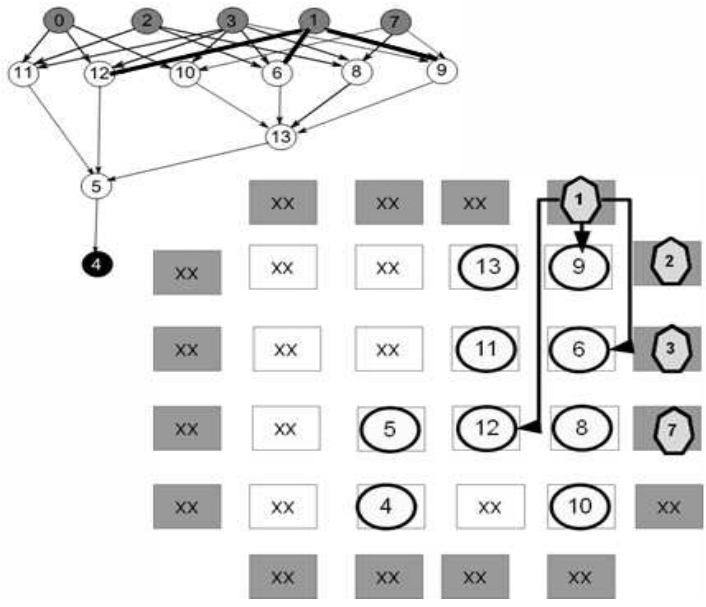


Figura 25: Etapa Parcial de BF: posicionamento do vértice de entrada 1.

Finalmente, o último vértice de entrada, o vértice 0, é visitado e posicionado como ilustra a Figura 26.

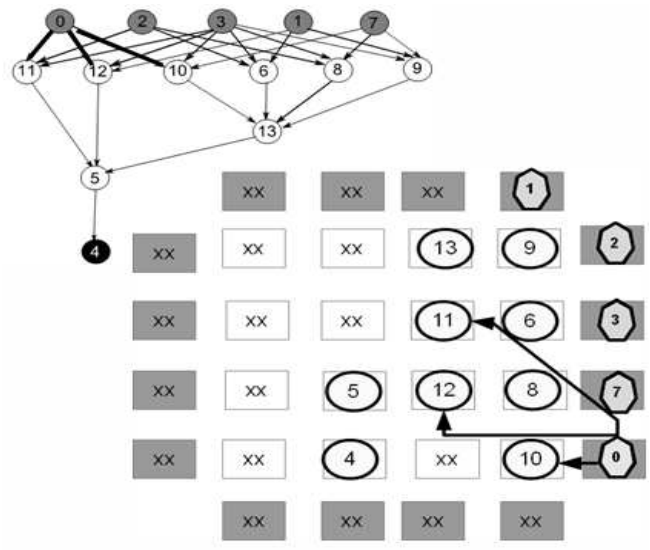


Figura 26: Etapa Final de BF: posicionamento do vértice 0.

A Figura 27 mostra o grafo de entrada com os custos das conexões nas arestas. O custo total para a travessia BF é 75, que é menor que o valor 79 obtido pela travessia DF. O custo médio por aresta é 2,88. O vértice 3 que tem o maior grau concentra 23,07% do total das arestas e 21,33% dos recursos de roteamento. Para a travessia em DF, o vértice 3 concentrava 31 % dos recursos de roteamento. Podemos observar que o BF reduziu o custo das conexões do vértice de alto grau. O caminho crítico para BF é 13 que é maior que o caminho crítico 12 gerado pela DF. Este exemplo é ilustrativo e não podemos tirar conclusões sobre o desempenho das duas travessias.

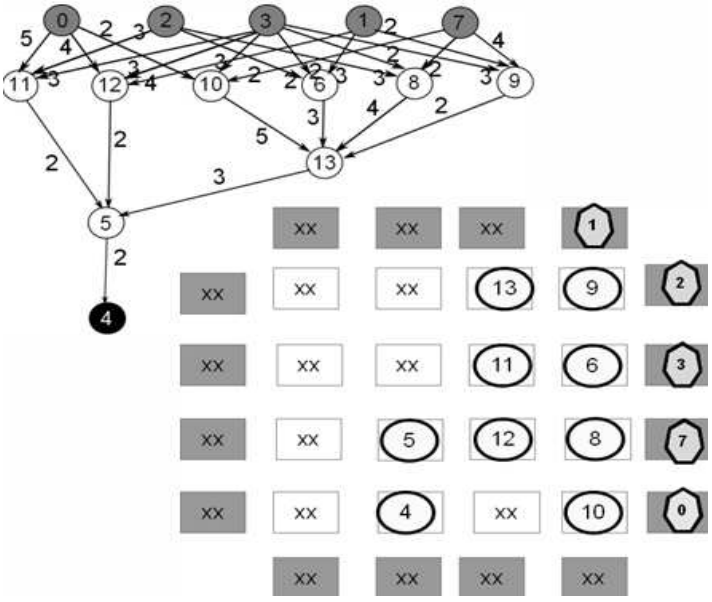


Figura 27: Custo Total Estimado para o Roteamento – após etapa de posicionamento baseado em BF.

3.3 Estratégias locais de otimização

3.3.1. Estratégia 1

A estratégia 1 tem por finalidade priorizar a reconvergência da classe de *fanout* 2 sem nenhuma reconvergência aninhada. O algoritmo percorre em DF, como ilustrado na Figura 28, mas ao encontrar um vértice com grau igual a 2, ele percorre no sentido oposto e os vizinhos são armazenados em uma lista antes de continuar a busca em profundidade. Enquanto o vértice possuir *fanout* 1, o algoritmo continua a busca até localizar um vértice já visitado, destacando uma reconvergência. Caso encontre um vértice que possua *fanout* maior que 1, a busca é interrompida.

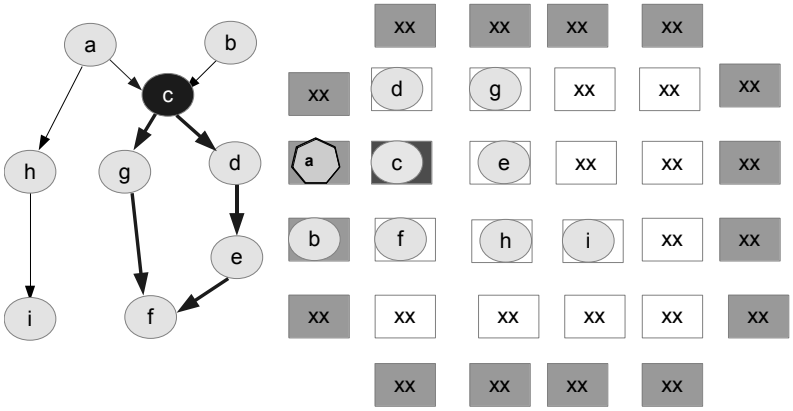


Figura 28: Estratégia 1.

Na Figura 28, a travessia começa, por exemplo, pelo vértice a, percorrendo aresta por aresta em profundidade. As seguintes arestas serão percorridas a→c, c→d, e→e e e→f. Ao encontrar o vértice f, verifica-se que esse possui *fanin* = 2, então pode existir uma reconvergência e inicia uma travessia no sentido oposto enquanto o par de aresta for *fanout* = 1 e ou encontrar um vértice já visitado. Assim, armazenam-se esses pares e depois os imprime, por exemplo: g→f e c→g. Ao concluir que por esse caminho já foram visitados todos os pares, o algoritmo recomeça pelo vértice a, e percorre a→h e h→i. O objetivo é agrupar as arestas dos caminhos reconvergentes e melhorar o posicionamento.

3.3.2. Estratégia 2

A estratégia 2 foi desenvolvida utilizando a abordagem de busca em largura (BF), como ilustrado na Figura 29. Nela as entradas são ordenadas pelo maior grau de saída (*fanout altos*), posteriormente, no nível seguinte (interno), também será ordenado pelo *fanout* de maior valor e assim sucessivamente. Todo esse processo é executado enquanto não for encontrada uma saída. O processo é repetido com todas as demais entradas.

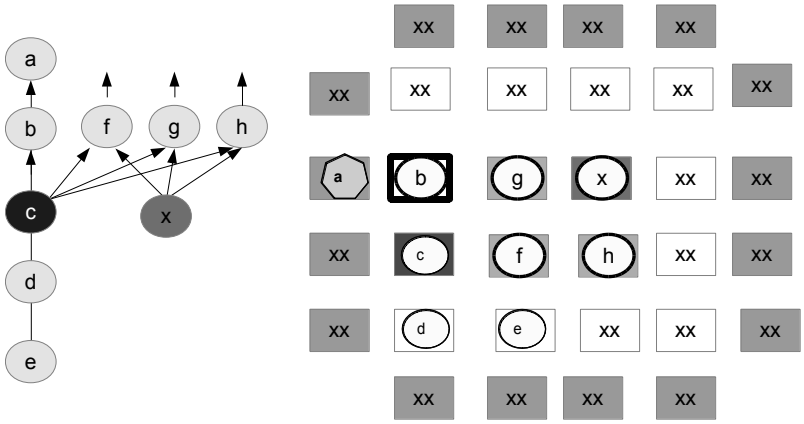


Figura 29: Estratégia 2: Ordenação da busca BF pelos *fanout.*

3.4 Algoritmo

Em tempo de síntese, as ferramentas de CAD são responsáveis por gerar o grafo do circuito que pode ser armazenado em um arquivo como uma lista de pares de arestas, na ordem de travessia (BF e/ou DF). Este é o ponto de partida para a execução do algoritmo proposto em [34]. Nesta dissertação usamos a mesma abordagem. O próximo passo é posicionamento dos vértices na matriz FPGA, onde usamos as heurísticas propostas nesta dissertação e comparamos os resultados com o algoritmo de P&R [34]. Para o roteamento usaremos a ferramenta VPR. O fluxo das ferramentas é apresentado na Figura 30.

No modelo proposto em [34], tanto o FPGA quanto o circuito a ser mapeado são representados por grafos. O posicionamento é implementado por uma função de mapeamento entre os dois grafos. Nesta dissertação usamos o mesmo modelo.

Como já mencionado, esta dissertação difere do algoritmo proposto em [34], ao explorar a travessia dos grafos no sentido inverso (entrada-saída), detectando durante a travessia os vértices com grau alto de *fanout* e propondo estratégias para priorizar o posicionamento deles e de seus descendentes. O caminho crítico e o comprimento total dos fios são usados para avaliar a qualidade das soluções propostas.

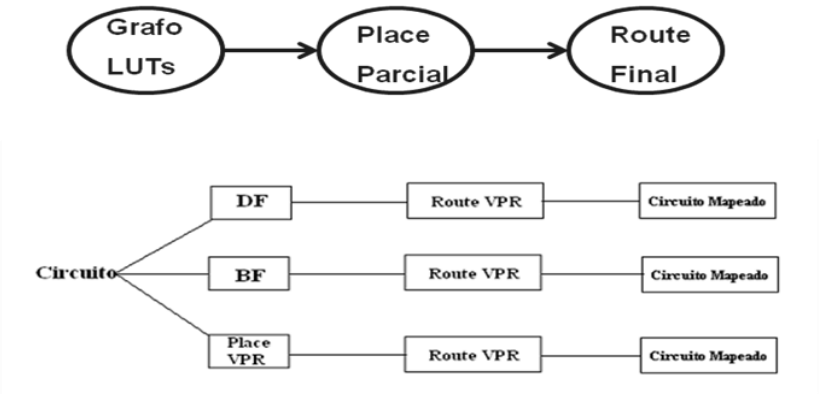


Figura 30: Tempo de síntese

O FPGA virtual que foi usado para avaliar o algoritmo proposto em [34], também foi usado nesta dissertação. Este FPGA foi proposto por [24, 28]. Vale ressaltar que o modelo pode ser personalizado para uma outra família FPGA específica, basta descrever as LUT, comutadores e canais para formar o grafo do FPGA.

Como já mencionado, o algoritmo de posicionamento é uma derivação do algoritmo proposto em [34] diferindo ao inverter o sentido do percurso e ao usar também a busca em largura. O posicionamento é ideal quando uma aresta A→B do circuito é mapeada em dois nós adjacentes x→y no grafo do FPGA. Porém, nem sempre é possível, seja devido uma limitação física no grafo FPGA ou da ordem na qual as arestas são visitadas, uma vez que é uma solução gulosa para um problema NP-completo. O conceito de nós adjacentes e nós não-adjacentes serão apresentados a seguir, semelhante à abordagem proposta em [34], porém no sentido inverso e com uma simplificação.

3.4.1 Nós adjacentes

Para o caso ideal, os vértices da aresta A→B devem ser posicionados o mais próximo possível um do outro. O caso ideal, onde os nós são adjacentes, tem distância 1 ou diretamente conectados com o custo mínimo (2 segmentos e um comutador). Mas nem sempre será possível obter o caso ideal. A abordagem proposta em [34] introduziu uma nova abstração para os nodos adjacentes no grafo FPGA, como a abordagem desta dissertação faz o percurso no sentido inverso, os conjuntos de nós adjacentes devem ser recalculados.

Uma contribuição desta dissertação, além do percurso no sentido inverso, foi à função de custo dos nós adjacentes no sentido inverso e sua simplificação. No modelo proposto em [34], existem 4 custos associados a cada entrada da LUT, como mostrado na Figura 31(b). Como nossa abordagem faz somente o posicionamento podemos simplificar o custo usando apenas o custo médio, reduzindo a busca por um fator 4x (Figura 31(c)). Considere o nodo $N_{i,j}$ na Figura 31(a), que mostra as posições dos 8 nodos na vizinhança do nodo $N_{i,j}$. A Figura 31(b) mostra, dentro de cada nodo, o custo de posicionamento para os nodos adjacentes como proposto em [34]. Os custos são calculados em função do roteamento. Cada nodo tem 4 entradas, uma em cada direção: sul, oeste, norte e leste. A saída do nodo $N_{i,j}$ é sempre pela direção sul. Por exemplo, o nodo do topo $N_{i-1,j}$ tem custo 3,2,3 e 2 se conectamos nas entradas oeste, norte, leste e sul, respectivamente. O custo médio será (3+2+3+2)/4 = 2. Para simplificar ainda mais o modelo, só usamos custos com números inteiros.

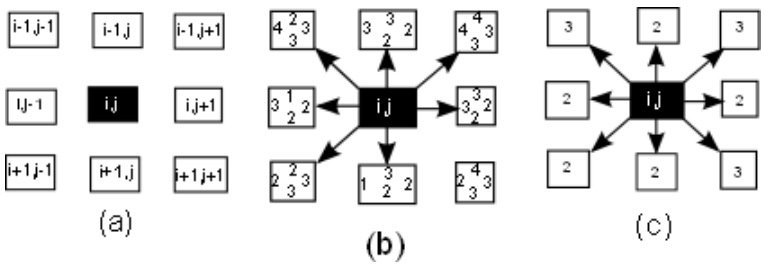


Figura 31: Vizinhança (a) LUTs FPGA (b) Custo LUT output

Primeiramente, o mapeamento tenta posicionar nós vizinhos com menor custo. Por exemplo, a Figura 11 considerando a travessia em DF de 0→10→13→5→4, será mapeada como ilustrado na Figura 12. Os cinco vértices são posicionados com um custo mínimo. O nodo 8 (8→13) também é posicionado com

um custo mínimo. Por exemplo, os nodos, 6 (6→13) e 9 (9→13) não podem ser posicionados em uma posição adjacente ao lado do vértice 13, como ilustrado na Figura 14, uma vez que o vértice 13 já esta posicionado e não há uma LUT vizinhança vazia.

Na abordagem BF, considerando a tentativa de posicionar nós vizinhos que possuem menor custo de posicionamento, para o caminho BF 3→6, 3→8, 3→9, 3→10, 3→11 e 3→12, os nós 11, 10 e 12 são posicionados com um custo mínimo de posicionamento, enquanto os nós restantes não são adjacentes ao nó 3.

Para resolver o problema de nós não-adjacentes foi proposto em [34] ampliar o conceito de custos de posicionamento com distância 2, 3 e 4. A Figura 32 descreve mais possibilidades de custo de posicionamento para conectar diferentes entradas a $N_{i,j}$, onde são mostrados vértices até com a distância 9. Como simplificamos a função custo para média dos custos, propomos realizar a busca em uma vizinhança maior, ampliando ainda mais o conceito de adjacência.

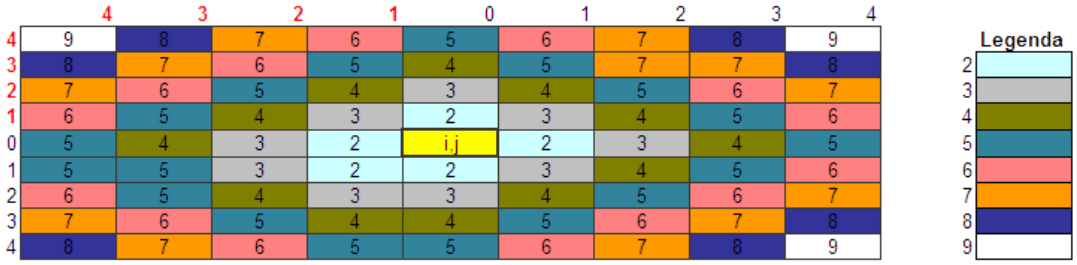


Figura 32: Tabela de Custos

3.4.2 Nós não-adjacentes

Alguns vértices não poderão ser posicionados nós adjacentes, pois não existem posições livres na vizinhança do vértice de origem da aresta, mesmo quando o conceito de nó adjacente é ampliado para custos maiores, estes casos podem ocorrer. Nesta situação, o algoritmo de busca em largura, procura a primeira posição vazia na vizinhança.

4. RESULTADOS

Os resultados foram medidos usando 15 *benchmarks* MCNC [6]. Este conjunto de *benchmark* é um padrão adotado para avaliar várias ferramentas de posicionamento e roteamento de FPGA. A ferramenta VPR também foi utilizada para roteamento e medida do caminho crítico e do número total de recursos de roteamento (segmentos e comutadores). O VPR está disponível em [41]. Maiores detalhes sobre as opções de configuração da ferramenta VPR, estão disponíveis em [5,6]. Os experimentos foram realizados em um Intel Core i3-2100, de 3,09 GHz, 3,41MB de cache LGA1155, 2 núcleos.

A proposta apresentada nesse trabalho é fazer uso de FPGA virtual semelhante ao proposto em [34], onde o FPGA virtual é composto por uma matriz 100x100, e cada canal contem 50 trilhas de roteamento. O objetivo principal é reduzir o tempo de execução do posicionamento, além de minimizar o comprimento total de fios e não deteriorar o caminho crítico. Além disso, avaliar também, qual será o tempo necessário para rotear o circuito com o algoritmo VPR. Para medir o tempo de posicionamento do VPR usamos os parâmetros usados foram*: -inner_num 1 –place_algorithm bounding_box*.

Nesta dissertação foram desenvolvidas e implementadas 4 variações do algoritmo proposto em [34]. Todas são no sentido entradas para saídas que difere da proposta de [34]. Duas são em largura, a solução BF e a solução em largura aliada a estratégia 2, que será referenciada por E2. As outras duas variações são em profundidade, a variação DF e a variação em profundidade aliada a estratégia 1, que será referenciada por E1. Para comparar a qualidade das soluções obtidas pelas variações propostas, usamos o algoritmo original proposto por [34], referenciado como P&R e o posicionamento do VPR.

4.1 Análise da execução de tempo de posicionamento

A Tabela 1 apresenta o tempo de posicionamento para as abordagens propostas DF, BF, VPR e para as estratégias E1 e E2, considerando 15 *benchmarks* MCNC [6]. As variações DF e BF são é 1002x e 1343x mais rápidas que o VPR. As

estratégias E1 e E2 são 432x e 1117x mais rápidas que o VPR. Comparando com os resultados do tempo de posicionamento para abordagem P&R [34], que foram disponibilizados apenas para 10 *benchmarks* (representamos com NA os benchmarks que não foram avaliados pelo P&R[34]), o P&R é 460x mais rápido que o VPR e em relação ao DF, BF e E1, obteve uma perda de 82,4%, 76,7% e 85,2%, respectivamente. Em relação a E2, o P&R obteve um ganho de 34,3%.

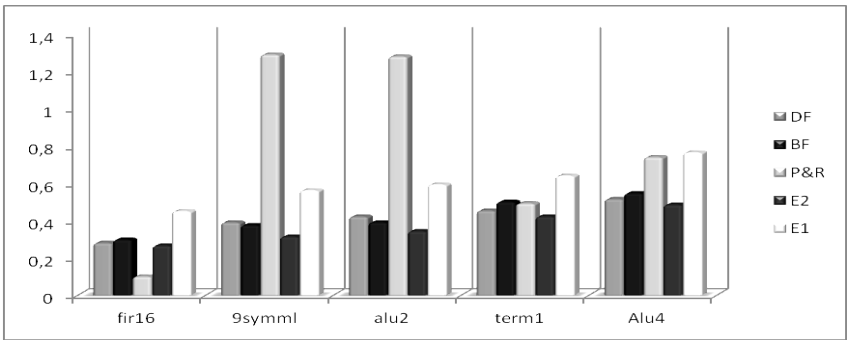


Gráfico 1: Tempo de execução do Posicionamento do fir 16 até o Alu4

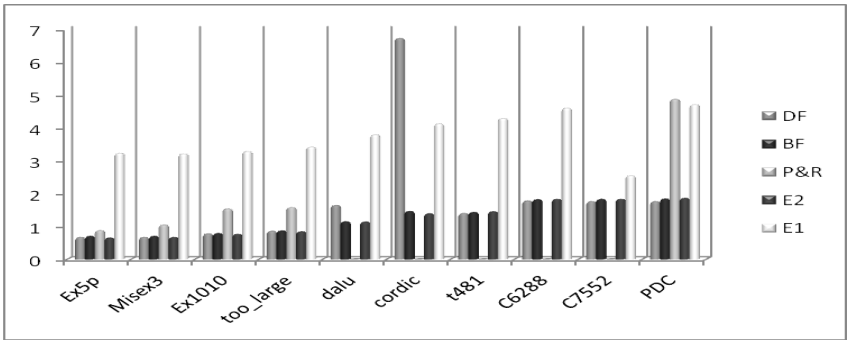


Gráfico 2: Tempo de execução do Posicionamento do Ex5p até o PDC.

Tabela 1: Tempo de execução do Posicionamento

				VPR	DF		BF		P&R		E2		E1	
Bench	CLB	Input	Output	sec	ms	ganho	ms	ganho	ms	ganho	ms	ganho	ms	ganho
fir16	47	1	1	0,0245	0,28	87,50	0,297	82,49	0,1	245	0,265	92,45	0,45	54,44
9symml	179	9	1	0,0947	0,39	242,82	0,375	252,53	1,29	73,41	0,313	302,56	0,562	168,51
alu2	241	10	6	0,135	0,421	320,67	0,39	346,15	1,28	105,47	0,343	393,59	0,593	227,66
Alu4	415	14	8	0,2721	0,515	528,35	0,546	498,35	0,74	367,70	0,484	562,19	0,765	355,69
term1	261	34	10	0,1685	0,453	371,96	0,5	337,00	0,492	342,48	0,421	400,24	0,64	263,28
C6288	2417	32	32	2,8288	1,781	1588,32	1,82	1554,29	NA	NA	1,828	1547,48	4,6	614,96
too_large	992	38	3	0,8764	0,859	1020,26	0,87	1007,36	1,58	554,68	0,843	1039,62	3,42	256,26
C7552	2536	207	108	3,0689	1,76	1743,69	1,83	1676,99	NA	NA	1,828	1678,83	2,546	1205,38
Misex3	771	14	14	0,6885	0,672	1024,55	0,703	979,37	1,05	655,71	0,671	1026,08	3,21	214,49
Ex5p	646	8	63	0,5892	0,672	876,79	0,703	838,12	0,88	669,55	0,655	899,54	3,23	182,41
Ex1010	964	10	10	0,9604	0,782	1228,13	0,79	1215,70	1,54	623,64	0,765	1255,42	3,28	292,80
PDC	2857	16	40	4,7344	1,76	2690,00	1,84	2573,04	4,87	972,16	1,859	2546,75	4,71	1005,18
cordic	2058	23	2	2,357	6,72	350,74	1,46	1614,38	NA	NA	1,391	1694,46	4,125	571,39
dalu	1618	75	16	1,6355	1,641	996,65	1,15	1422,17	NA	NA	1,141	1433,39	3,79	431,53
t481	2073	16	1	2,7445	1,4	1960,36	1,43	1919,23	NA	NA	1,453	1888,85	4,29	639,74

*NA – benchmarks não avaliados pelo algoritmo P&R

4.2 Análise da influência do posicionamento na qualidade do roteamento

A Tabela 2 apresenta o caminho crítico, ou seja, o caminho mais longo no grafo da FPGA para as abordagens propostas nesta dissertação, afim de comparar com os algoritmos de posicionamento P&R [24] e VPR [5]. Para medir o caminho crítico, a saída de cada posicionamento é roteada usando o algoritmo de roteamento VPR. A saída do roteamento informa o caminho crítico gerado. Foram avaliados 15 *benchmarks* do MCNC [6]. Em relação à ferramenta VPR como referência, o DF apresentou um aumento médio do caminho crítico em 47%, o BF gerou um aumento de 81%, enquanto que o P&R é equivalente ao VPR. E em relação às estratégias E1 e E2, estes tiveram um aumento de 13% e 49% respectivamente. Portanto as heurísticas baseadas em profundidade tem um comportamento melhor quando o critério de custo é o caminho crítico, pois uma das informações usadas no percurso é a priorização do caminho crítico inicial.

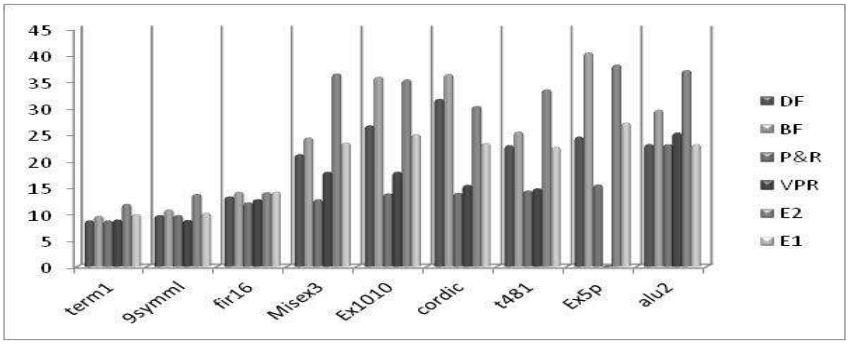


Gráfico 3: Caminho Crítico de term1 até o Alu2

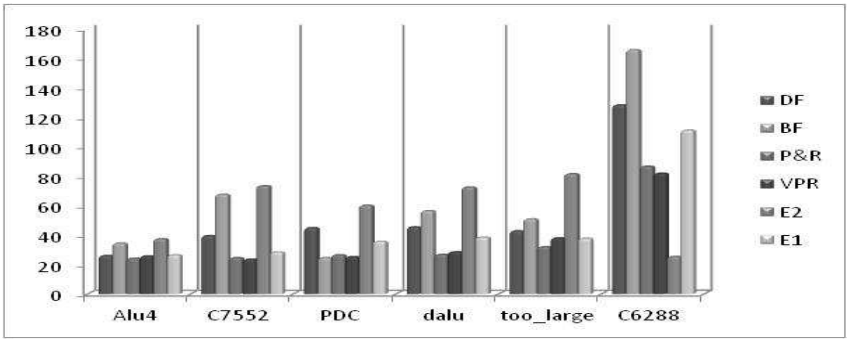


Gráfico 4: Caminho Crítico de Alu4 até C6288.

Tabela 2: Caminho Crítico

Bench	DF	BF	P&R	VPR	E2	E1
fir16	13,23	14,1	12,1	12,7	14,01	14,12
9symml	9,71	10,76	9,7	8,76	13,7	10,14
alu2	23,14	29,63	23,1	25,31	37,12	23,13
Alu4	26,11	34,56	24,2	25,8	37,47	26,51
term1	8,69	9,53	8,7	8,85	11,81	9,87
C6288	128,48	166	86,74	82	25,41	111,34
too_large	42,9	50,94	32,1	38,09	81,71	37,87
C7552	39,61	67,65	24,63	23,6	73,5	28,5
Misex3	21,22	24,38	12,7	17,89	36,51	23,44
Ex5p	24,55	40,48	15,5	14.43	38,16	27,2
Ex1010	26,67	35,88	13,8	17,89	35,36	25
PDC	44,96	24,65	26,6	25,3	60,25	35,76
cordic	31,71	36,41	13,92	15,4	30,33	23,33
dalu	45,56	56,55	26,81	28,61	72,66	38,57
t481	22,92	25,51	14,34	14,76	33,48	22,61

Observando alguns casos específicos na Tabela 2, podemos ver:

1. **Pdc** (que é o maior circuito) o caminho crítico no VPR, o P&R e BF se equiparam, mas em relação ao DF, E1 e E2, o caminho crítico foi 1-2x, 1x e 2-3x maior. Mostrou que apesar da BF, ser uma busca em largura que não avalia o caminho crítico na travessia, a BF pode gerar uma solução com custo baixo.
2. **Alu2**: DF, P&R e E1 valores próximos (apenas 10% acima) do VPR. O desempenho das soluções em largura BF e E2 obtiveram uma perda de 17% e 46%, respectivamente.
3. **Ex1010**: P&R teve um aumento de 22% em relação ao VPR. As soluções DF, BF, E1 e E2 tiveram um aumento de 49%, 100%, 40% e 98%, respectivamente.

O segundo aspecto avaliado para medir a qualidade do posicionamento é o tempo de execução do algoritmo de roteamento do VPR. Um posicionamento com vértices de alto grau distantes pode exigir um esforço maior do algoritmo de roteamento. Tabela 3 apresenta os tempos de execução para rotear cada *benchmark* após a etapa de posicionamento das abordagens DF, BF, E1 e E2 propostas nesta dissertação, do P&R proposto em [34] e da ferramenta VPR [5]. Para facilitar a comparação, uma coluna mostra o ganho em relação ao posicionamento gerado pelo VPR. O roteamento foi executado com os parâmetros*: -route_only* –

route_chan_witch 50. Na coluna representada pela sigla G/P indica G-ganho (valores acima de 1) e P-perda (valor abaixo de 1).

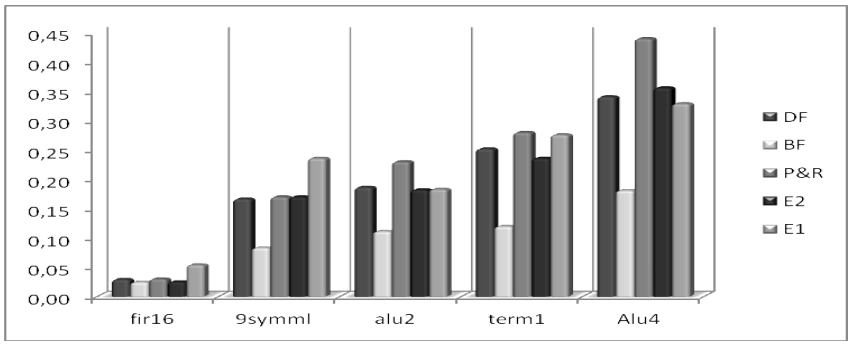


Gráfico 5: Tempo de Roteamento de fir16 até Alu4.

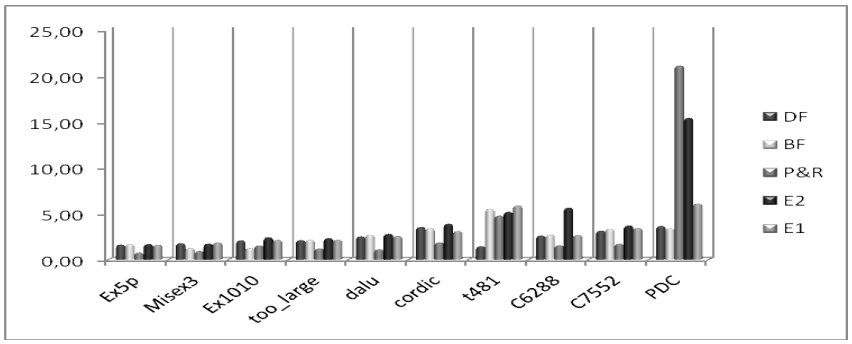


Gráfico 6: Tempo de Roteamento de Ex5p até PDC.

Tabela 3: Tempo de Roteamento

	DF	G/P	BF	G/P	P&R	G/P	VPR	E2	G/P	E1	G/P
Bench	sec		sec		sec		sec	sec		sec	
fir16	0,03	0,00	0,02	1,10	0,03	0,00	0,03	0,03	0,00	0,05	0,50
9symml	0,17	0,92	0,08	1,84	0,17	0,90	0,15	0,17	0,90	0,24	0,65
alu2	0,19	1,00	0,11	1,68	0,23	0,81	0,19	0,18	1,02	0,18	1,02
Alu4	0,34	0,87	0,18	1,63	0,44	0,67	0,30	0,36	0,83	0,33	0,90
term1	0,25	0,75	0,12	1,57	0,28	0,67	0,19	0,24	0,80	0,28	0,68
C6288	2,60	0,86	2,69	0,84	1,54	1,46	2,25	5,63	0,40	2,66	0,85
too_large	2,10	0,94	2,17	0,91	1,19	1,66	1,97	2,32	0,85	2,15	0,92
C7552	3,12	0,81	3,32	0,76	1,71	1,47	2,52	3,69	0,68	3,42	0,74
Misex3	1,77	0,82	1,26	1,15	0,92	1,58	1,45	1,71	0,85	1,85	0,79
Ex5p	1,62	0,90	1,67	0,87	0,75	1,94	1,46	1,67	0,87	1,61	0,91
Ex1010	2,07	0,89	1,25	1,47	1,52	1,20	1,83	2,42	0,76	2,14	0,86
PDC	3,65	1,18	3,41	1,26	21,10	0,20	4,31	15,40	0,28	6,08	0,71
cordic	3,53	0,80	3,41	0,83	1,85	1,53	2,84	3,88	0,73	3,10	0,92
dalu	2,51	0,89	2,68	0,83	1,10	2,02	2,23	2,78	0,80	2,56	0,87
t481	1,44	4,18	5,51	1,09	4,78	1,26	6,02	5,19	1,16	5,88	1,02

Em geral, o P&R gerou um desempenho mais próximo do VPR. As soluções DF, BF, E1 e E2 foram em média piores e apresentaram resultados semelhantes, exceto para o PDC e o T481. Para o PDC que é o maior circuito, o DF e o BF obtiveram um ganho de 19% e 21% em relação ao VPR, respectivamente, enquanto a

E1 obteve uma perda de 41%, e as abordagens E2 e P&R tiveram perda bem considerável, acima de 200%, em relação ao VPR.

O próximo aspecto avaliado é o número total de conexões ou recursos de roteamento usados pelos algoritmos de posicionamento. O processo foi o mesmo dos experimentos anteriores. Para cada posicionamento, o circuito é posteriormente roteado no VPR que fornece o consumo de recursos de roteamento através do número total de segmentos usados. A Tabela 4 apresenta o total de segmentos para as abordagens propostas DF e BF, as estratégias E1 e E2, e o P&R e o VPR, considerando 15 *benchmarks* MCNC. Na coluna representada pela sigla G/P indica G-ganho e P-perda, para facilitar a análise.

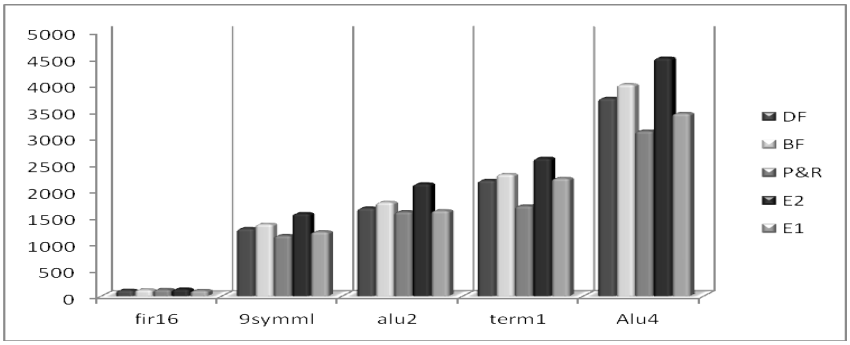


Gráfico 7: Segmentos de fiação de fir16 até Alu4.

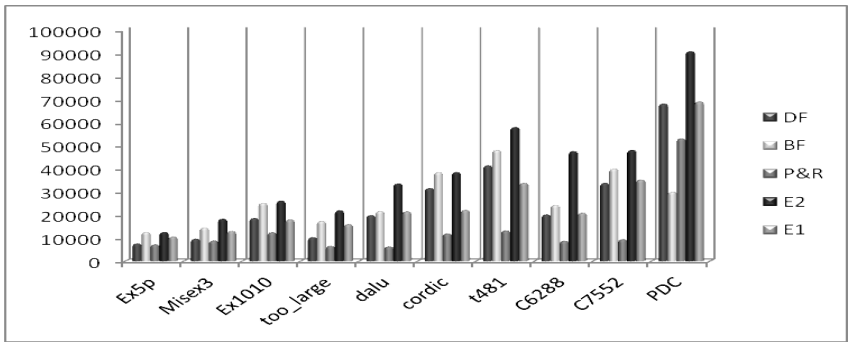


Gráfico 8: Segmentos de fiação de Ex5p até PDC.

Tabela 4: Segmentos de fiação

Bench	DF	G/P	BF	G/P	P&R	G/P	VPR	E2	G/P	E1	G/P
term1	109	28,24	118	38,82	123	44,71	85	128	50,59	103	21,18
9symml	1275	42,78	1359	52,18	1141	27,77	893	1559	74,58	1213	35,83
fir16	1662	41,57	1773	51,02	1593	35,69	1174	2121	80,66	1614	37,48
Misex3	3741	61,81	4001	73,05	3121	34,99	2312	4502	94,72	3454	49,39
Ex1010	2187	59,75	2302	68,15	1704	24,47	1369	2605	90,28	2225	62,53
cordic	20161	123,27	24248	168,53	8726	-3,37	9030	47491	425,92	20856	130,96
t481	10300	91,17	17381	222,59	6522	21,05	5388	21905	306,55	15954	196,10
Ex5p	33838	229,71	40020	289,94	9452	-7,90	10263	48053	368,22	35226	243,23
alu2	9654	58,11	14552	138,32	8845	44,86	6106	18285	199,46	12994	112,81
Alu4	7597	38,48	12600	129,68	7229	31,77	5486	12535	128,49	10638	93,91
C7552	18620	86,65	25214	152,75	12450	24,80	9976	26091	161,54	18026	80,69
PDC	68096	114,92	30001	-5,31	53045	67,42	31684	90713	186,31	69025	117,85
dalu	31519	164,87	38435	222,98	11919	0,16	11900	38466	223,24	22108	85,78
too_large	19825	180,53	21607	205,75	6268	-11,31	7067	33511	374,19	21483	203,99
C6288	41411	228,24	48071	281,03	13218	4,77	12616	57928	359,16	33779	167,75

Pode-se observar que o VPR é superior a todas as heurísticas, pois avalia diferentes opções de posicionamento com a técnica SA, enquanto que as outras soluções são gulosas e avaliam apenas 1 posicionamento, reduzindo significativamente o tempo de execução. O preço a ser pago é o uso de mais conexões. Podemos observar que para maioria dos circuitos, as soluções baseadas em profundidade P&R, DF e E1 tem o melhor desempenho. As soluções em largura usam, em geral, uma quantidade maior de fios, exceto para o circuito PDC que apesar de ser um caso específico é o maior circuito. Para o PDC em relação ao VPR, o BF obteve ganho de 5,3%, e DF, E1, E2 e P&R obtiveram perda de 114%, 117%, 186% e 67%, nesta ordem.

4.3 Análise de mapeamento de arestas em função do *fanout*

A qualidade do posicionamento afeta diretamente no custo de roteamento. Essa seção descreve as análises de ocupação dos recursos de roteamento (fios e comutadores) para o mapeamento das arestas em função dos *fanouts dos vértices.* O objetivo é visualizar o impacto do posicionamento na distribuição das conexões para as abordagens DF, BF e as estratégias E1 e E2.

Para facilitar a explicação iremos definir uma notação. Os vértices serão agrupados em classes em função do seu *fanout.* A classe f_i é formada pelo conjunto de vértices com *fanout igual a i*. Considere ainda a aresta $e = a \rightarrow b$, como sendo a primeira aresta visitada de *a*. Como já mencionado, o posicionamento é realizado por

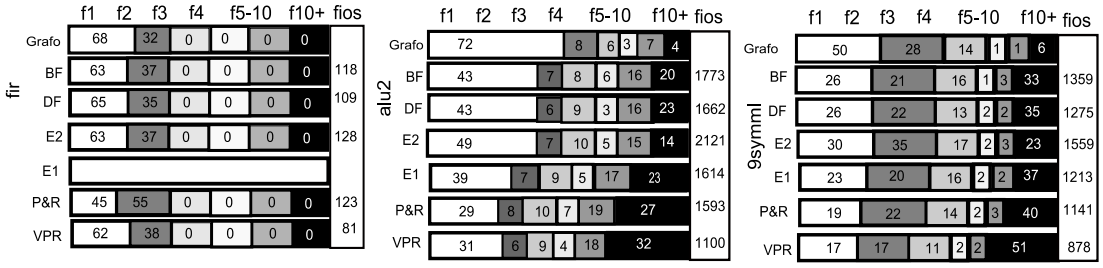
uma função de mapeamento entre dois grafos (do FPGA e do circuito a ser mapeado), explora a localidade destes vértices durante o processo. O posicionamento de *b* é definido quanto *e* é visitado durante a travessia de DF e/ou BF. Para explorar a localidade e otimizar o custo da aresta *e*, o vértice *b* é posicionado mais próximo quanto possível do vértice *b*. Cada aresta é mapeada em um ou mais segmentos de fios/comutadores no grafo FPGA, e seu mapeamento consome *n* fios e *n-1* comutadores.

As Figuras 33 à 37 apresentam em detalhes com as classes de *fanout* i nos 15 *benchmarks* avaliados. Os *benchmarks* foram agrupados de três em três para facilitar a visualização, gerando 5 figuras. Para cada benchmark a distribuição ou o histograma do uso das conexões em função das classes de *fanout* são apresentados.

A primeira linha mostra o histograma do *fanout* (grau de saída) do grafo de cada circuito em função da classe $f_{i.}$. O objetivo é visualizar qual o consumo de recursos de roteamento para cada classe. No grafo inicial, todas as arestas tem custo 1. Depois de posicionado por um algoritmo, cada aresta terá um custo medido pelo número de conexões que ela usa. Estes dados são obtidos após o roteamento do VPR, analisando seu arquivo de saída. Da segunda linha à sexta linha (índices BF, DF, E2 e E1, e P&R, respectivamente) a percentagem é mostrada por classe de *fanout* de cada uma das heurísticas gulosas avaliadas (todas avaliam apenas uma possibilidade de posicionamento); e a última linha, a sétima (índice VPR), apresenta os resultados do posicionamento do VPR que avalia muitas possibilidades de posicionamento para minimizar o caminho crítico.

Por exemplo, a ALU2 tem 72% de seus vértices com *fanout* 1 classe f_1, como ilustrado na primeira linha da Figura 34. As arestas de saída desta classe irão consumir 43% do total de recursos de roteamento. Enquanto que a classe f_3. Que corresponde a apenas 6% vértices irá consumir 8% dos recursos de roteamento.

A distribuição de *fanouts* em circuitos não é uniforme, apesar da média ser próxima de 3. Para facilitar a análise, agrupamos os *fanouts* maiores em classes com vários graus diferentes. O índice f5-10 representa as classes com os *fanouts* f5, f6, f7, f8, f9 e f10. A classe f10+ por sua vez representa todos os vértices com *fanout i* > 10. A maioria dos vértices possuem grau em f1 e f2, que justifica a média ser em torno de 3 como mostram as figuras a seguir. Por exemplo na Figura 34, os f1 + f2 para os *benchmarks* fir, alu2 e 9ymml, com graus 1 e 2 são responsáveis por 100%, 80% e 78% dos vértices, respectivamente.



a) b) c)

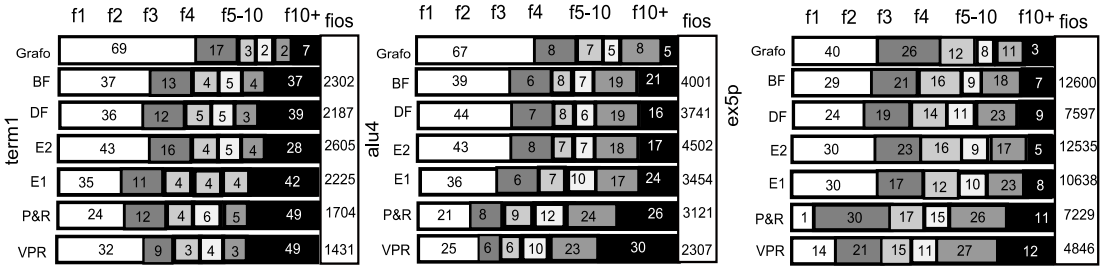
Figura 33: Disposição de vértices para os *benchs*: fir, alu2 e 9symml

Conforme mostrado na Figura 33 (a), o *benchmark* fir possui 68% dos vértices em f1 e 32% dos vértices *em* f2, e após realizar o posicionamento BF, os nós de f1 usam 63% dos fios e os nós em f2 usa 37%. Como o Fir é um circuito pequeno com *fanout* baixo, não podemos usá-lo como comparação. Podemos notar que todas as estratégias geram uma distribuição similar, exceto a estratégia P&R que prioriza o *fanout* f1. Como as abordagens baseadas em travessia são gulosas, o consumo de conexões é 30-50% maior do que o consumo gerado pelo VPR. Ao lado de cada barra temos o indicador do total de fios. A BF, por exemplo, usa 118 conexões em comparação com o VPR que usa apenas 81. A abordagem DF gasta apenas 109 conexões.

Outro exemplo mostrado na Figura 33 (b) é o *benchmark* alu2, onde para a classe de f10+, que indica a classe de i > 10, ocupa apenas 4% dos vértices, mas tem um impacto variando de 14% à 32% no consumo dos recursos de roteamento. As abordagens propostas nesta dissertação reduzem o consumo de conexões da classe f10+ para a ALU2 como podemos notar (14% para E2 à 23% para DF). Já os trabalhos anteriores P&R e VPR, a classe f10+ consome de 27% a 32% das conexões. Entretanto, o consumo total de conexões é maior nas abordagens DF, BF, E1 e E2. O pior caso é a estratégia E2 que consome 2121 fios. Embora f10+ represente apenas 14% das conexões, este valor corresponde a 296 fios comparando com o VPR que consome 354 fios ou 32% das conexões em f10+. Ao priorizar os vértices de alto grau, o consumo total e das outras classes pode aumentar como ilustra este exemplo. Vale ressaltar que as abordagens avaliadas aqui são estratégias gulosas que fazem o posicionamento apenas uma vez para um problema de otimização NP-completo. Portanto, uma simplificação adotada pelo algoritmo guloso irá melhorar algum aspecto, mas pode deteriorar outros e o impacto final ser

negativo. O importante é observar detalhamento onde está o impacto e por quê ele ocorre.

Comparando as estratégias que são baseadas em largura (BF e E2), com as baseadas em profundidade (DF e E1) observamos que as em profundidade são melhores, considerando-se o consumo total de fios. Entretanto, as estratégias BF e E2 tem foco nas ligações com múltiplos *fanout*. A classe f10+ para as quais estas estratégias tem foco, como observamos para circuito ALU2, onde BF e E2 usam 354 e 296 fios, respectivamente, em comparação com P&R que usa 430 fios. Como já mencionado, a melhora nos múltiplos *fanout* não compensa o número maior de fios que são usados pelas outras classes, que leva a um resultado geral pior se olharmos o total de fios usados pelas heurísticas DF e E1.



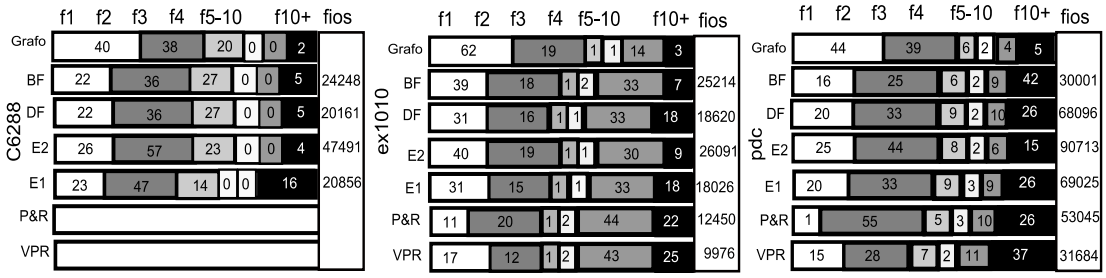
a) b) c)

Figura 34: Disposição de vértices para os *benchs*: term1, alu4 e ex5p.

Na Figura 34 (a), observamos que para *benchmark* term1, a heurística P&R tem um perfil semelhante ao VPR, diferindo apenas nas classes f1 e f2, e com um consumo maior de recursos (1704 contra 1431 do VPR). Das estratégias avaliadas nesta dissertação, o DF que também é baseado em profundidade e o E1 são melhores que as estratégias baseadas em largura. Apesar da estratégia E2 reduzir o consumo relativo da classe f10+ com 28% em relação às demais abordagens que consomem de 37% a 49% dos recursos com a classe f10+, a estratégia E2 é a pior no consumo total de fios, consumindo praticamente o dobro de fios em relação ao VPR.

Comparando as estratégias baseadas em largura (BF e E2) e profundidade (DF e E1), as últimas são melhores nos três *benchmark*s considerando o consumo total de fios. Novamente, como já mencionado, as estratégias BF e E2 tem foco nas ligações com múltiplos *fanout*. Essas estratégias priorizam a classe f10+. Para circuito term1, BF e E2 usam 851 e 729 fios, nesta ordem, em comparação com P&R que usa 834 fios. Porém a melhora nos múltiplos *fanout* não compensa o número

maior de fios que são usados pelas outras classes, que leva a um resultado pior se olharmos o total de fios usados pelas heurísticas DF e E1.

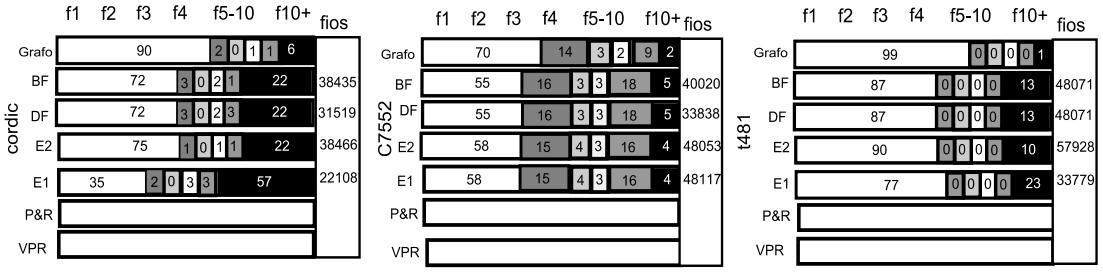


a) b) c)

Figura 35: Disposição de vértices para os *benchs*: C6288, ex1010 e pdc.

A Figura 35 apresenta circuitos maiores que a Figura 34. Enquanto as soluções do VPR para a Figura 34 consomem de 1431 à 4846, na Figura 35, os três *benhmarks* são bem maiores e consomem de 9976 a 31684 conexões. O *benchmark* C6288 não foi avaliado em [34] e por isto não aparece na Figura 35. Para o C6288 observamos novamente que a estratégia em profundidade DF é bem melhor que as em largura, se considerarmos o total de fios. O pior caso é a estratégia E2 que consome 47491 fios em comparação com os 20161 usados pela abordagem DF. O *benchmark* ex1010 é bem posicionado pelo P&R, porém as abordagens DF, BF, E1 e E2 não são bem sucedidas.

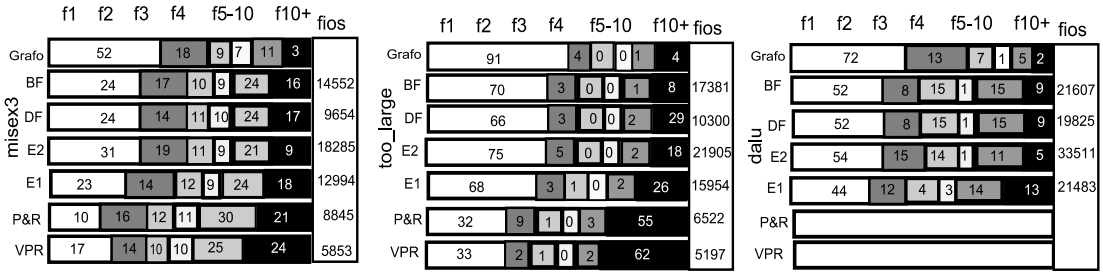
Um destaque deve ser dado ao PDC, onde a estratégia gulosa BF proposta nesta dissertação supera o VPR que avalia múltiplos posicionamentos e consome de 1000 a 10000x mais tempo de execução. O BF como as outras heurística realiza apenas um posicionamento. O destaque é válido, apesar de ser um caso individual, pois são mais de 3000 vértices e mais de 30.000 conexões. Em geral o BF é pior que as abordagens em profundidade. O PDC mostra que nem sempre é verdade e que existe uma grande variação por ser um problema de otimização. Soluções que usem ambas as abordagens (largura e profundidade) podem ser exploradas como trabalhos futuros com algum critério de busca local, aumentando um pouco o tempo de execução, mas reduzindo o consumo total.



a) b) c)

Figura 36: Disposição de vértices para os *benchs*: cordic, C7552 e t481.

A Figura 36 apresenta três casos que não foram avaliados em [34] pelo P&R. Observamos que para estes casos, a busca em profundidade também gera uma redução no consumo de fios significativa, mesmo que o consumo relativo de conexões da classe f10+ seja maior.



a) b) c)

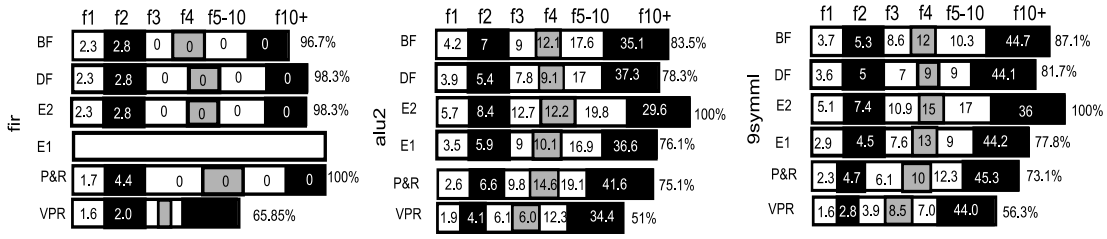
Figura 37: Disposição de vértices para os *benchs*: misex3, too_large e dalu.

Na Figura 37, o misex3 e o too_large tem um comportamento semelhante aos anteriores, onde o P&R gera o resultado mais próximo em relação ao VPR. Dentre as abordagens propostas aqui, a DF é que tem o melhor comportamento. Novamente a redução relativa do consumo em F10+ feito pela estratégia E2 não é compensado nas outras classes. No dalu, o comportamento de E2 foi o pior e as outras estratégias apresentaram resultados similares.

Os nós de *fanouts* elevados contribuem para o custo total de fios, mesmo que estes sejam poucos frequentes no grafo original. Por exemplo, considera-se o *benchmark* term1 (Figura 34 (a)) que possui o *fanout* alto na classe f10+ com apenas 6.5% dos nós de entrada, após o mapeamento, estes nós consumirão 36.9% (BF), 38.2% (DF), 27.8% (E2), 41.5% (E1) e 48.2% (P&R) do total de fios.

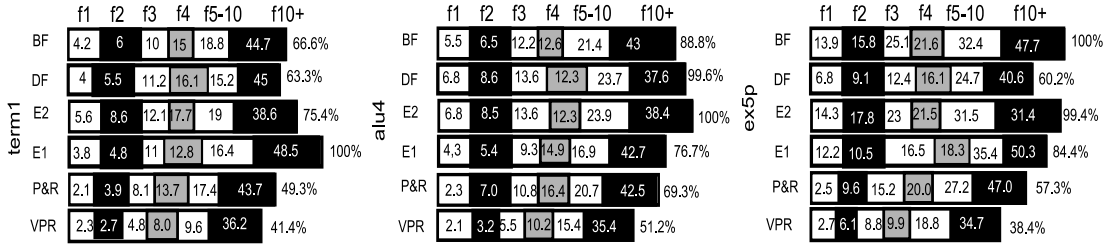
Nesta seção iremos avaliar qual é o comprimento médio das conexões em função de sua classe fi. As Figuras 39 à 42 irão detalhar o consumo médio de

comutadores (*switches*) para cada a classe *fanout* fi. Por exemplo, o term1 na Figura 39 usa 4.2, 4, 5.6, 3.8 e 2.1 conexões para cada aresta mapeadas pelas as abordagens, BF, DF, E2, E1 e P&R, nessa ordem. Já para a classe f2, cada vértice tem grau 2 de saída ou *fanout* 2. Então se o consumo para o term1 é de 6, 5.5, 8.6, 4.8, 3.9 e 2.7conexões para cada aresta mapeadas pelas as abordagens, BF, DF, E2, E1 e P&R, nessa ordem. Isto significa que para o BF, a classe f2 usa 6 comutadores ou 3 comutadores por aresta (já que cada vértice tem duas arestas) em comparação com o VPR que usa apenas 2.7 comutadores ou 1.35 comutadores por aresta.



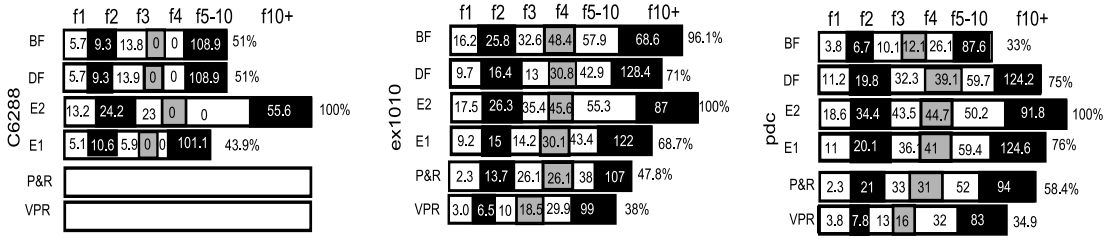
a) b) c)

Figura 38: Consumo médio de comutadores para os *benchs*: fir, alu2 e 9symml.



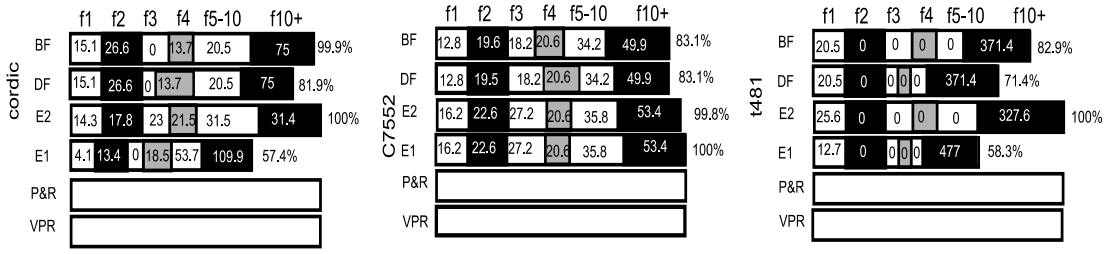
a) b) c)

Figura 39: Consumo médio de comutadores para os *benchs*: term1, alu4 e ex5p.



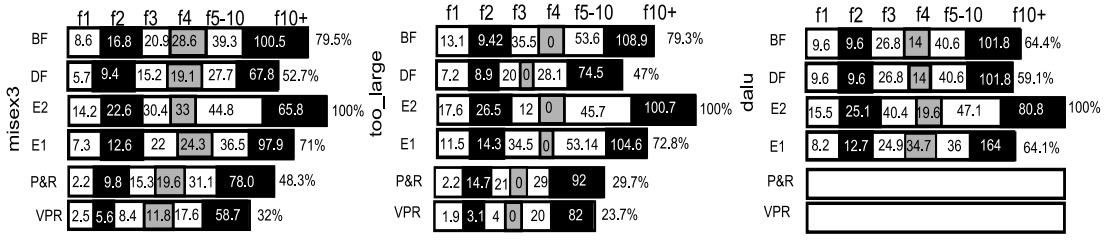
a) b) c)

Figura 40: Consumo médio de comutadores para os *benchs*: C6288, ex1010 e pdc.



a) b) c)

Figura 41: Consumo médio de comutadores para os *benchs*: cordic, C7552 e t481.



a) b) c)

Figura 42: Consumo médio de comutadores para os *benchs*: misex3, too_large e dalu.

O consumo de comutadores ilustrado nas Figuras 38-42 complementa as informações de distribuição vistas nas Figuras 33-37. Para facilitar as comparações, as barras tem o tamanho proporcional ao consumo total de comutadores. Sendo o pior caso com 100% e os outros são avaliados tendo como referência o pior caso. Podemos observar que a estratégia E2 é pior em 12 dos 15 casos avaliados se olharmos o custo total. Porém como ela prioriza o posicionamento dos vértices de alto grau, a E2 é a melhor de todas as soluções para reduzir o consumo de comutadores da classe F10+ em 8 dos 15 casos, sendo que em 7 casos é uma das melhoras. Ou seja, ela cumpre o seu objetivo de otimizar a classe f10+, porém a priorização destes vértices deteriora o custo total, pois gera um posicionamento ruim para as outras classes.

Além disso, o apêndice C apresenta um histograma detalhado da distribuição dos graus dos vértices, onde podemos observar as classes F5-10 e F10+ sem agrupamento. O maior grau e sua frequência também podem ser observados.

5. CONCLUSÕES E TRABALHOS FUTUROS

Esta dissertação explora uma nova técnica de posicionamento de FPGA, que foi proposta inicialmente em [34]. O trabalho anterior é avaliado com detalhes e novas variações são propostas nesta dissertação. O FPGA é modelado por um grafo com uma correspondência direta com sua estrutura física. Nesta dissertação o modelo é simplificado e gera resultados semelhantes em muitos casos quando comparado ao trabalho anterior [34]. O posicionamento consiste em mapear o grafo do circuito em um grafo FPGA. A localidade dos vértices foi explorada durante esse processo. Porém, diferente do trabalho anterior [34], buscamos explorar e avaliar outras estratégias de travessia. As abordagens em largura e profundidade foram avaliadas. O sentido do percurso também foi modificado. Além disso, duas estratégias locais para explorar o grau 2 (*fanout*=2) e o grau múltiplo de saída dos vértices (alto *fanout*) foram avaliadas.

Com base nos resultados experimentais, o resultado obtido pelo algoritmo de posicionamento proposto nesta dissertação foi em média de 1002x e 1343x, em profundidade e em largura, respectivamente, quando comparados à ferramenta VPR [5,6]. O VPR é o padrão acadêmico usado para avaliar a qualidade das soluções de roteamento e posicionamento.

O problema é NP-completo e estamos avaliando heurísticas gulosas que fazem apenas uma tentativa por vértice. Ou seja, o mais simples possível. O objetivo é ter heurísticas rápidas que sejam viáveis para serem usadas em ambientes de reconfiguração parcial durante a execução das aplicações. O espaço de soluções é exponencial e cresce muito rápido. Uma solução gulosa poderia gerar um posicionamento que não seja possível de rotear. Entretanto, mostramos nesta dissertação que os 4 algoritmos propostos sempre geram soluções que podem ser roteadas. Usamos como referência o posicionamento baseado em SA do VPR [5]. Esta técnica gera milhares ou até milhões de tentativas a mais que as abordagens propostas aqui. Assim, o algoritmo SA não é viável de ser usado em tempo de execução para reconfiguração parcial. Além disso, apesar de fazer apenas uma tentativa, as abordagens propostas aqui conseguem gerar um resultado que no pior caso é apenas 2x pior no consumo de recursos de roteamento. Como o número de fios é fixo, um consumo maior não afeta a funcionalidade do circuito. Em termos de

caminho crítico, os resultados mostram que as heurísticas gulosas aumentam em 30-60% o caminho crítico, dobrando em alguns casos. Como já ressaltado, o tempo de posicionamento é reduzido em 3 à 4 ordens de grandeza, para uma perda de 2x apenas.

A alternativa explorada aqui é a nova técnica proposta em [34] baseada apenas em profundidade. Podemos observar que se pode usar tanto travessia em largura quando em profundidade, como contribuição desta dissertação. O tempo de execução permanece baixo e os posicionamentos geram uma solução que podem ser roteada. Em geral, a profundidade é melhor. Apesar da largura reduzir os custos relativos dos vértices com alto grau, o consumo total e o caminho crítico em média, para os circuitos avaliados, é pior.

5.1 Trabalhos Futuros

Uma estratégia local pode ser usada em ambas as travessias (largura ou profundidade) para melhorar o posicionamento dos vértices. O trabalho anterior e o proposto aqui, usa apenas a primeira visita ao vértice para fixar sua posição. Uma vez posicionado, o vértice não muda de lugar. Uma alternativa seria deixar lacunas ou LUT não usadas (em preto) durante o posicionamento, como mostra a Figura 43. Elas não podem ser usadas durante a execução do posicionamento inicial. Após o posicionamento terminar, os vértices de alto grau ou de *fanout* 2 que serão revisitados, podem ter sua posição modificada, como por exemplo o vértice c na Figura 43. Assim pode-se mover o c para um local mais próximo a seus vizinhos.

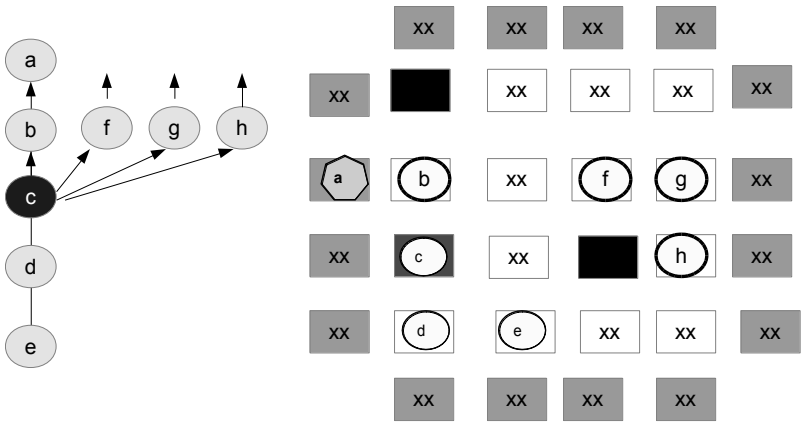


Figura 43: Estratégica 3: Lut vazia.

Podemos observar que na Figura 44, o vértice c agora é posicionado perto de f, g e h, reduzindo a tarefa do roteamento para o *multicast*, ficando longe apenas dos vértices b e d. A mudança de posição pode aumentar o caminho crítico, mas um estudo detalhado deve ser realizado com experimentos usando vários benchmarks com características diferente que representem o universo dos circuitos digitais como os *benchmarks* MCNC.

6. REFERÊNCIAS BIBLIOGRÁFICAS

[1] ALTERA CORPORATION - White Paper. **FPGA Run-Time Reconfiguration: Two Approaches.** San Jose, CA, 2008. Disponível em: http http://www.altera.co.jp/literature/wp/wp-01055-fpga-run-time-reconfiguration.pdf. Acesso: 10 de novembro de 2011.

[2] ALTERA CORPORATION - **FPGAs**. Disponível em: www.altera.com/products/fpga.html. Acesso: 12 de maio de 2012.

[3] ASSIS, Alex Damiany. **Um Algoritmo de Posicionamento e Roteamento Polinomial para Arquiteturas Reconfiguráveis de Grão Grosso com Redes Multiestágio**. Viçosa: MG, 2010. 14 p. Tese (Mestrado) – Programa de Pós-Graduação em Ciência da Computação, Universidade Federal de Viçosa, Viçosa, 2010.

[4] BETZ, Vaughn. **175. Vpr: SPEC CPU2000 Benchmark Description File**. Disponível no site: http://www.spec.org/cpu2000/CINT2000/175.vpr/docs/175.vpr.html. Acesso: 15 de setembro de 2011.

[5] BETZ, Vaughn; CAMPBELL, Ted; FANG, Wei Mark; JAMIESON, Peter; KUON, Ian; LUU, Jason; MARQUARDT, Alexander; ROSE, Jonathon; YE, Andy. **VPR and T-VPack1 User's Manual.** Summer 2008 VPR 5.0 Full Release, 2008.

[6] BETZ, Vaughn. **FPGA Architecture for the Challenge**. Disponível no site: http://www.eecg.toronto.edu/~vaughn/challenge/fpga_arch.html. Acesso: 15 de setembro de 2011.

[7] CARVALHO, Gustavo Rezende. **Heurísticas para a fase de Roteamento de Circuitos Integrados Baseados em FPGAs**. João Pessoa: PB, 2010. Tese (Mestrado) – Programa de Pós-Graduação em Informática (área de conhecimento:

Sistemas de Informação), UFPB: Universidade Federal da Paraíba, João Pessoa, 2010.

[8] GONZALEZ, Jose Artur Quilici. **Uma Metodologia de Projetos para Circuitos com Reconfiguração Dinâmica de Hardware aplicada a Support Vector Machines**. São Paulo: SP, 2006. Tese (Doutorado) – Programa de Pós-Graduação em Engenharia (área do conhecimento: Microeletrônica), USP: Escola Politécnica, São Paulo, 2006.

[9] LIMA, Ednaldo M. V. de. **Posicionamento Automático em Circuitos Implementados em FPGAs: Desenvolvimento de Rotinas de Aceleração Computacional**. João Pessoa: Paraíba, 2008. Tese (Mestrado) – Programa de Pós-Graduação em Informática, Universidade Federal da Paraíba, João Pessoa, 2008.

[10] MARQUES, Paulo César Furlanetto. **Algoritmo de Roteamento Maze para Dispositivos Programáveis FPGA**. Porto Alegre: RS, 2002. Tese (Mestrado) – Programa de Pós-Graduação em Ciência da Computação, Pontifícia Universidade Católica do Rio Grande do Sul, Porto Alegre, 2002.

[11] MINKOVICH, Kirill. **BLIF Benchmark suit.** Disponível no site: http://cadlab.cs.ucla.edu/~kirill/. Acesso: 16 de setembro de 2011.

[12] MOREANO, Nahri Balesdent. **Algoritmos para Alocação de Recursos em Arquiteturas Reconfiguráveis**. Campinas: SP, 2005. Tese (Doutorado) – Programa de Pós-Graduação em Ciência da Computação, Instituto de Computação Universidade Estadual de Campinas – UNICAMP, São Paulo, 2005.

[13] OLIVEIRA, Fábio Abreu Dias de Oliveira. **Arquiteturas reconfiguráveis de computadores: características gerais e um estudo de caso**. Tese (Mestrado) – Universidade Federal do Rio Grande do Sul – Instituto de Informática – Programa de Pós-Graduação em Computação.

[14] ROSE, Jonathan; BROWN, Stephen. **Flexibility of Interconnection Structures for Field-Programmable Gate Arrays**. IEEE Journal of Solid-State Circuits, vol. 26, n 03, march 1991.

[15] SARMENTO, Marcelo. **Arquiteturas Auto-reconfiguráveis em Sistemas Digitais**. Porto Alegre: RS, 2001. Monografia – Programa de Bacharelado em Informática, Pontifícia Universidade Católica do Rio Grande do Sul, Porto Alegre, 2001.

[16] SHARMA, Akshay. **Place and Route Techniques for FPGA Architecture Advancement**. Washington: EUA, 2005. Tese (Doutorado) – Programa de Pós-Graduação em Filosofia (Área de conhecimento: Engenharia Elétrica), University of Washington Graduate Scholl, Washington, 2005.

[17] STEVEN J.E. Wilton. **Architectures and Algorithms for Field-Programmable Gate Arrays with Embedded Memory**. Toronto: Canadá, 1997. Tese (Doutorado) – Programa de Pós-Graduação em Filosofia (Área de conhecimento: Engenharia Elétrica e de Computadores), University of Toronto, 1997.

[18] ZHUO, Yue. **Timing and Congestion Driven Algorithms for Fpga Placement**. 2006. Tese de Doutorado. University of North Texas.

[19] MCMURCHIE, Larry; EBELING, Carl. **PathFinder: a negotiation-based performance-driven router for FPGAs**. Conference: Symposium on Field Programmable Gate Arrays - FPGA , pp. 111-117, 1995.

[20] LEONHARDT, Charles Capella. **Roteamento de Circuitos VLSI**. Universidade Federal do Rio Grande do Sul. Porto Alegre, 2010.

[21] LEONHARDT, Charles Capella; JUNIOR, Adriel Mota Ziesemer; REIS, Ricardo Augusto da Luz. **Uma nova abordagem do Pathfinder utilizando o algoritmo iterated 1-steiner**. UFRRGS. Resumo publicado em evento. Repositório Digital. XX Salão de Iniciação Científica - 2008. Acesso: 16 de janeiro de 2013.

[22] HENTSCHKE, Renato Fernandes. **Algoritmos para posicionamento de células em circuitos VLSI**. Dissertação de Mestrado - Programa de Pós-Graduação de Pós-Graduação em Computação. (Área de conhecimento: Ciência da Computação), UFRRGS: Porto Alegre: 1997.

[23] FERREIRA, R. S.; GARCIA, A.; TEIXEIRA, T.; CARDOSO, J. M. P. **A Polynomial Placement Algorithm for Data Driven Coarse-Grained Reconfigurable Architectures**. in IEEE Computer Society Annual Symposium on VLSI (ISVLSI'07), 2007.

[24] H. Sidiropoulos, K. Siozios, P. Figuli, D. Soudris, and M. Hubner, **On supporting efficient partial reconfiguration with just-in-time compilation**, in PhD Forum (IPDPSW), IEEE, 2012.

[25] K. Papadimitriou, A. Dollas, S. Hauck. **Performance of partial reconfiguration in FPGA systems: A survey and a cost model**. ACM Trans. Reconf. Technol. Syst., vol. 4, Dec. 2011.

[26] M. Gericota, G. Alves, M. Silva, J. Ferreira. **Run-time management of logic resources on reconfigurable systems.** in Design, Automation and Test in Europe Conference and Exhibition, 2003, pp. 974–979.

[27] M. Handa and R. Vemuri. **An efficient algorithm for finding empty space for online FPGA placement**. in Proceedings of DAC, 2004.

[28] M. Hubner, P. Figuli, R. Girardey, D. Soudris, K. Siozios, and J. Becker. **A heterogeneous multicore system on chip with run-time reconfigurable virtual FPGA architecture**. In Workshops and Phd Forum (IPDPSW), 2011.

[29] A. Ludwin and V. Betz. **Efficient and Deterministic Parallel Placement for FPGAs**. ACM Trans. Des. Autom. Electron. Syst., vol. 16, no. 3, 2011.

[30] M. Lin and J. Wawrzynek **Improving FPGA placement with dynamically adaptive stochastic tunneling**. CAD of Integrated Circuits and Systems, IEEE Trans-actions on, vol. 29, no. 12, 2010.

[31] M. Xu, G. Grewal, and S. Areibi. **Starplace: A new analytic method for FPGA placement**. Integration, the VLSI Journal, vol. 44, no. 3, pp. 192 – 204, 2011.

[32] H. Bian, A. C. Ling, A. Choong, and J. Zhu. **Towards scalable placement for FPGAS**. in Proceedings of FPGA, 2010.

[33] S. Y. Chin and S. J. Wilton. **Towards scalable FPGA cad through architecture**. In Proceedings of FPGA, 2011.

[34] FERREIRA, Ricardo; ROCHA, Luciana; SANTOS, Andre Gustavo dos; NACIF, Jose Augusto; WONG, Stephan; CARR, Luigi. **A Run-time Graph-Based Polynomial Placement and Routing Algorithm for Virtual FPGAs**. 23rd International Conference on Field Programmable Logic and Applications. Porto, Portugal, 2013.

[35] STITT, G. **Are Field-Programmable Gate Arrays Ready for the Mainstream?** Publicada em IEEE Computer Society. Florida, USA, 2011.

[36] HAUCK, Scott; DEHON, Andre. **Reconfigurable Computing: The Theory and Practice Of FPGA-Based Computation**. Elsevier. 2008.

[37] SILVA, Marcos Vinícius da. **Exploração do Espaço de Projeto de Arquiteturas Reconfiguráveis em Arranjos**. Viçosa: MG, 2010. Dissertação de Mestrado – Programa de Pós-Graduação em Ciência da Computação, Universidade Federal de Viçosa, Viçosa, 2006.

[38] Y.L. Wu; D. Chang. **On the NP-completeness of regular 2-D FPGA routing architectures and a novel solution**. IEEE/ACM International Conference on Computer-aided design, pages 362-366. CA, USA, 1994.

[39] DEHYADGARI M., NICKRAY M., AFZALI-KUSHA, A. and NAVABI, Z. **Evaluation of pseudo adaptive XY routing using an object oriented model for NoC**, in Micro-electronics, International Conference on, 2005.

[40] LIN, X, MCKINLEY P, and NI, L. **Deadlock-free multicast wormhole routing in 2-d mesh multicomputers**, Parallel and Dist. Syst, IEEE Trans on, vol. 5, 1994.

[41] *Benchmarks*. **VPR and T-VPack: Versatile Packing, Placement and Routing for FPGAs**. Disponivel em: http://www.eecg.toronto.edu/~vaughn/vpr/vpr.html

APÊNDICE A – Exemplo circuito em Formato .net

Para alimentar o VPR, deve-se fornecer quatro parâmetros necessários, além de outros parâmetros opcionais, como se segue:

vpr netlist.net architecture.arch placement.p routing.r [-options]

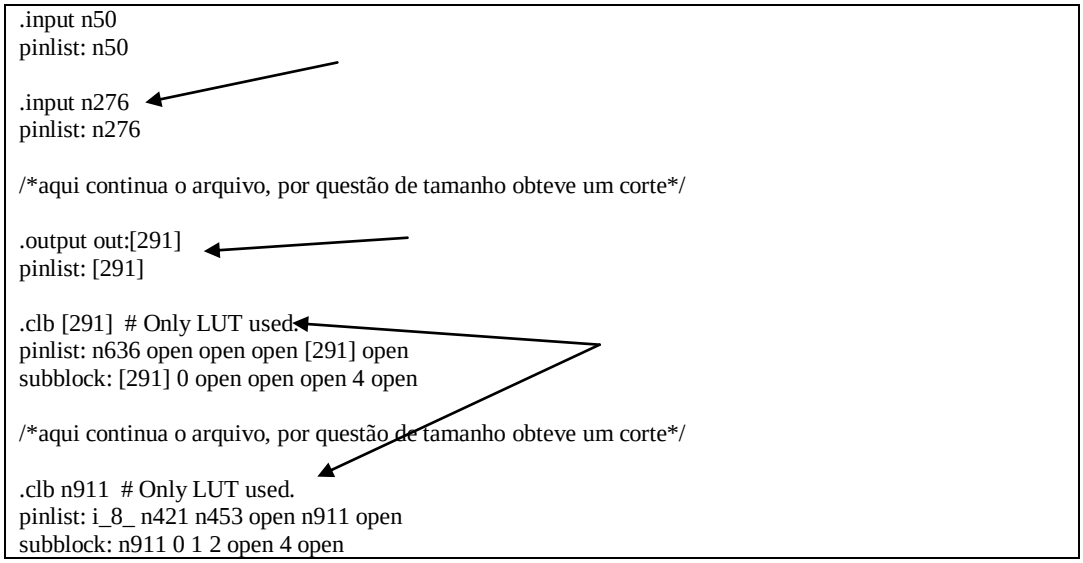


Figura 44: Exemplo de circuito em formato .net.
Fonte: Acervo Pessoal.

- Os IOBs (*.input* e *.output*): possuem apenas um pino;

- Os blocos lógicos (*.clb*): possuem quantos pinos que foram especificados no arquivo de arquitetura (*.xml*) especificado.

- O identificador *pinlist* contem uma lista de nomes de todas as redes (*nets*) conectadas a cada um dos pinos do *.clb* ou *.iob*. A conexão das *.clbs* listadas nos *pinlist*, se iniciam ao pino 0, e assim sucessivamente ao pino [1, 2, ... n], se caso algum pino do *.clb* não estiver conectado a entrada correspondente, este deve ser identificada pela palavra reservada *open*.

- A palavra reservada *subblock* representa o conteúdo interno dos *.clb* e especifica o nome do sub-bloco, e depois apresenta o pino da *.clb* ou o pino de saída do sub-bloco no qual o *.clb* está conectado. E cada *.clb* deve ter pelo menos uma linha de *subblocks*.

APÊNDICE B – VPR (*Versatile Placement and Routing Tool*)

O VPR[1] foi desenvolvido pelo grupo de pesquisa do Prof. Vaughn Betz [5] e é amplamente utilizado em pesquisas de algoritmos em FPGA por proporcionar flexibilidade na definição da arquitetura e possuir diversas opções de algoritmos de posicionamento e roteamento. O VPR evoluiu ao longo dos anos com novas versões, incluindo arquiteturas heterogêneas comuns nos FPGA atuais [8, 15, 16, 18]. O VPR é uma ferramenta CAD que faz uso do algoritmo *simulated annealing* para posicionamento e do algoritmo *pathfinder para* roteamento [4, 5, 6, 17].

O VPR posiciona inicialmente o circuito e através de iterações sucessivas do algoritmo SA busca melhorar a qualidade do posicionamento. Em seguida, o roteamento é executado. Ambos os algoritmos tem opções para priorizar tempo de execução ou caminho crítico. Além de buscar minimizar o número de trilhas nos canais.

A Figura 45 mostra o fluxo de execução da ferramenta CAD VPR. Os arquivos de entrada consistem em um *netlist.net* e *architecture.xml*. O arquivo de entrada deve ser um *netlist* formado por LUTs e *flip-flops* e um arquivo XML *(.xml*) que descreve a arquitetura do FPGA em que este circuito será implementado. Detalhes do arquivo *netlist* estão no Apêndice A. Nesta dissertação são considerados arquivos sem *flip-flops*, mas, com pequenas modificações no posicionamento que pode ser estendido para circuitos sequenciais.

[1] Mais informações sobre documentos e download da ferramenta VPR e *benchmarks* disponível em http://www.eecg.utoronto.ca/vpr

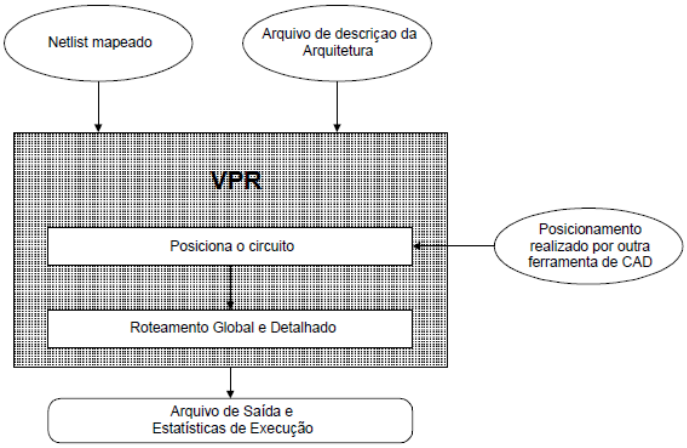
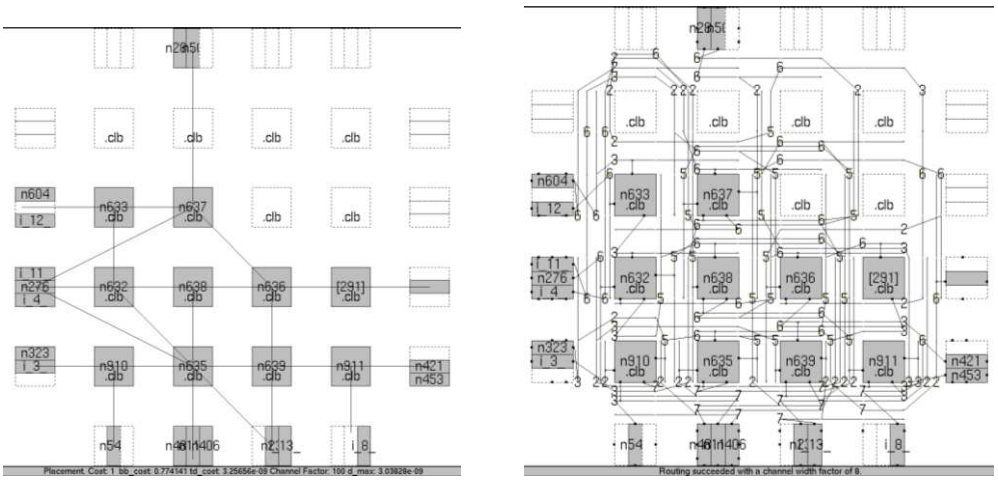


Figura 45: Fluxo de execução do roteador VPR
Fonte: Marques. Algoritmo de Roteamento Maze para Dispositivos programáveis FPGA.

A arquitetura do FPGA é especificada no formato XML. Este arquivo descreve os principais atributos do FPGA largura dos canais, especificações dos blocos lógicos, quantidade de pinos de I/O, quantidade de sub-blocos internos, quantidade de entradas das LUTs utilizadas nos sub-blocos, detalhes sobre as estruturas de roteamento, características como capacitâncias das conexões, tipos de blocos de comutadores, conjuntos de trilhas em que cada bloco CLB ou IOB estão conectados [5, 9, 10, 12].

A saída da VPR gera o posicionamento e/ou roteamento, apresentando estatísticas para avaliação da arquitetura FPGA, tais como o comprimento total dos fios do roteamento, contagem de nodos de I/O, contagem do máximo de trilhas (*tracks*) por canal, tempo de execução, entre outras informações.

A versão 5.0 do VPR [6] possui uma interface gráfica que permite visualizar passo a passo o posicionamento e roteamento. Este recurso é útil não só para circuitos pequenos nas quais se podem visualizar os detalhes, mas também para circuitos maiores, nos quais se pode ver a concentração de fios nos canais e nos comutadores, o caminho crítico, bem como detalhes da estrutura de roteamento (conexões locais das LUTs e os segmentos). A Figura 46 (a) apresenta o circuito *t_k4* depois da realização da etapa de posicionamento. É possível observar os identificadores dos blocos lógicos e as conexões ponto a ponto entre os blocos sem detalhes de roteamento. A Figura 46 (b) apresenta o roteamento do circuito *t_k4* pelo *software* VPR. Pode-se visualizar a concentração de fios em uso dos segmentos nos canais de roteamento



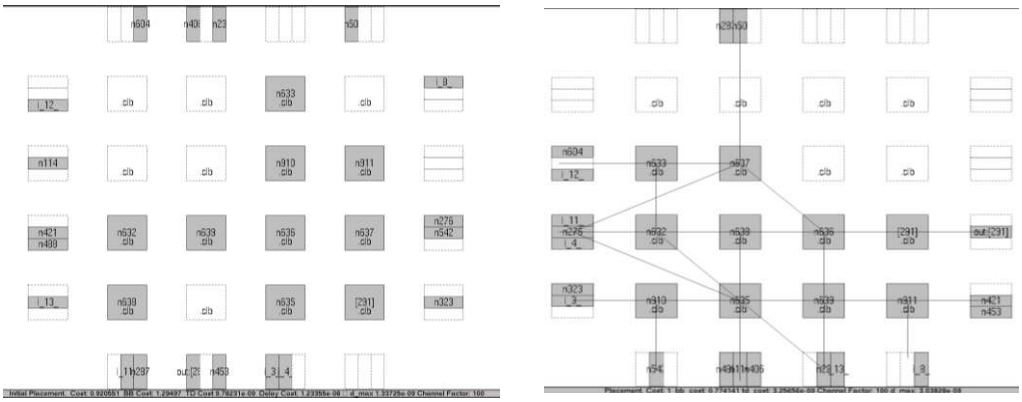
a) b)

Figura 46: Circuito t_k4: Posicionamento (a) e Roteamento (b) realizado pelo software VPR 5.0

Posicionamento do VPR

O posicionamento dos blocos lógicos no circuito é realizado através do algoritmo *Simulated Anneling* (SA). Devido à complexidade computacional do problema e a abordagem iterativa do algoritmo, se um grande número de iterações for realizado, o tempo de execução pode ser elevado [7, 9, 10, 22].

A interface gráfica do VPR permite visualizar, como ilustrado nas Figuras 47(a) e 47 (b) a etapa de posicionamento inicial e final do circuito *t_k4*, respectivamente.



a) b)

Figura 47: Circuito t_k4: (a) Posicionamento Inicial e (b) Posicionamento Final realizado pelo software VPR 5.0

O VPR pode realizar apenas o posicionamento e gerar um arquivo de saída, ou pode realizar ambos, posicionamento e roteamento. O VPR pode também realizar apenas o roteamento, sendo que o usuário pode fornecer o posicionamento, deixando apenas a etapa de roteamento para VPR. Este recurso foi utilizado nesta dissertação, já que, como mencionado o nosso objetivo é um algoritmo de posicionamento. Podendo então usar o VPR para a realização do roteamento.

Após a etapa final do posicionamento dos blocos lógicos, a etapa de roteamento consiste em realizar a conexão entre os blocos através da conexão direta entre blocos adjacentes ou indireta passando por um ou mais comutadores programáveis seguindo uma função de custo[2] (Figura 48).

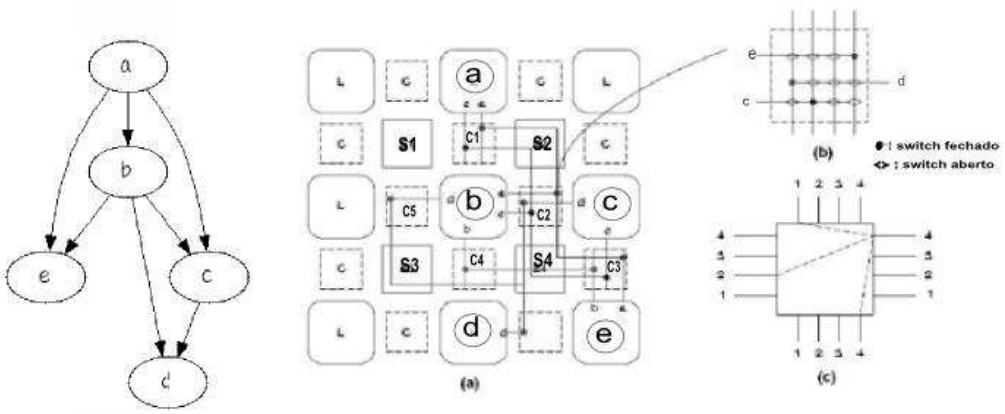


Figura 48: (a) Exemplo de roteamento. (b) Um bloco C2. (c) Um bloco S
Fonte: CARVALHO

O custo em segmentos para o grafo acima (Figura 48) é igual a 15, onde nas arestas a→e, a→c e b→d, usam 3 segmentos de fios cada um. Neste exemplo são usados 3 comutadores (S2, S3 e S4) sendo que os comutadores mais usados foram S2 e S4 para realizar as conexões entre vértices. São usados 5 canais de roteamento sendo os que apresentam maior taxa de utilização c2 e c3.

A Figura 49 representa o grafo do *benchmark* [6] *t.blif* de tamanho 4x4 e, a Figura 50 ilustra as etapas de posicionamento e roteamento

[2] Mais informações sobre posicionamento e roteamento da ferramenta VPR estão disponíveis em: http://www.eecg.toronto.edu/~vaughn/vpr/vpr.html

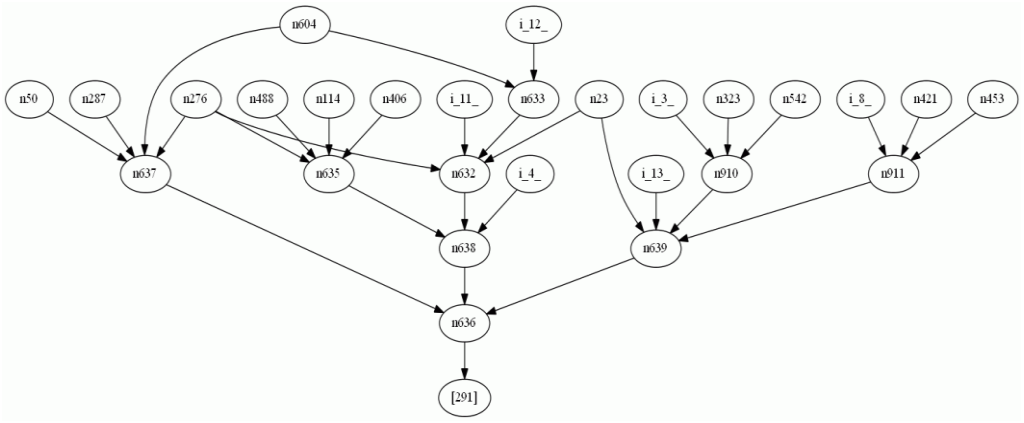
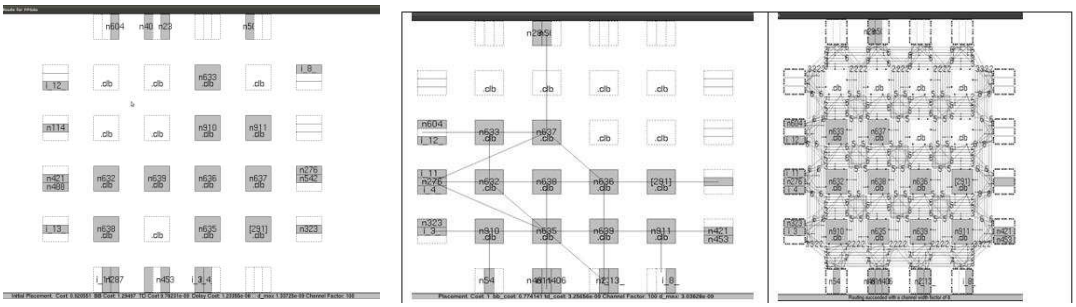


Figura 49: Grafo t.blif



a) b) c)

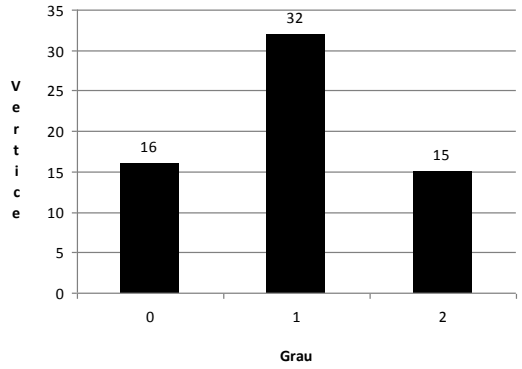
Figura 50: Circuito na rede VPR a) Posicionamento Inicial b) Posicionamento Final e c) Roteamento final no VPR

O roteamento depende da qualidade do posicionamento, uma vez que um posicionamento ruim pode impossibilitar o roteamento dependendo das restrições dos recursos de roteamento como a largura máxima do canal. Como já mencionado, ambos os problemas de posicionamento e roteamento em FPGAs são NP-completo [18].

APÊNDICE C – Histogramas

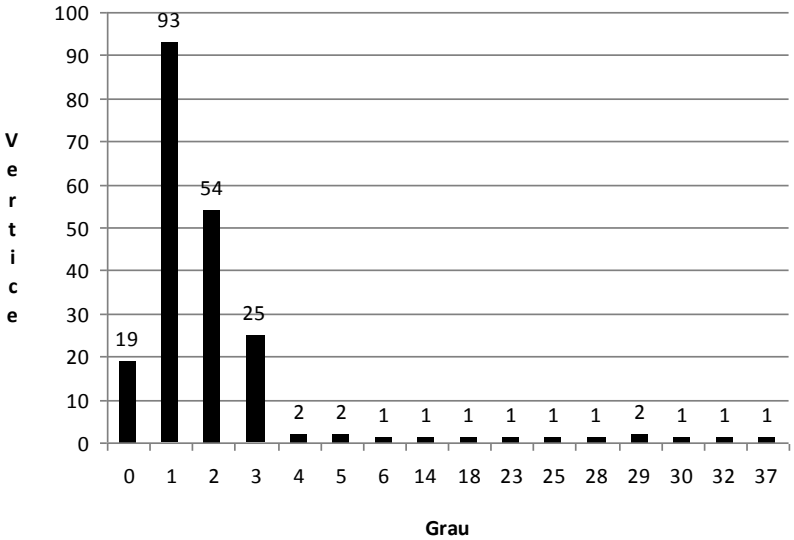
Fir16

Grau	Vertices	arestas	% aresta	% acc arestas
TOTAL vertices	63	62		
0	16	0		
1	32	32	0,516129032	0,516129032
2	15	30	0,483870968	1



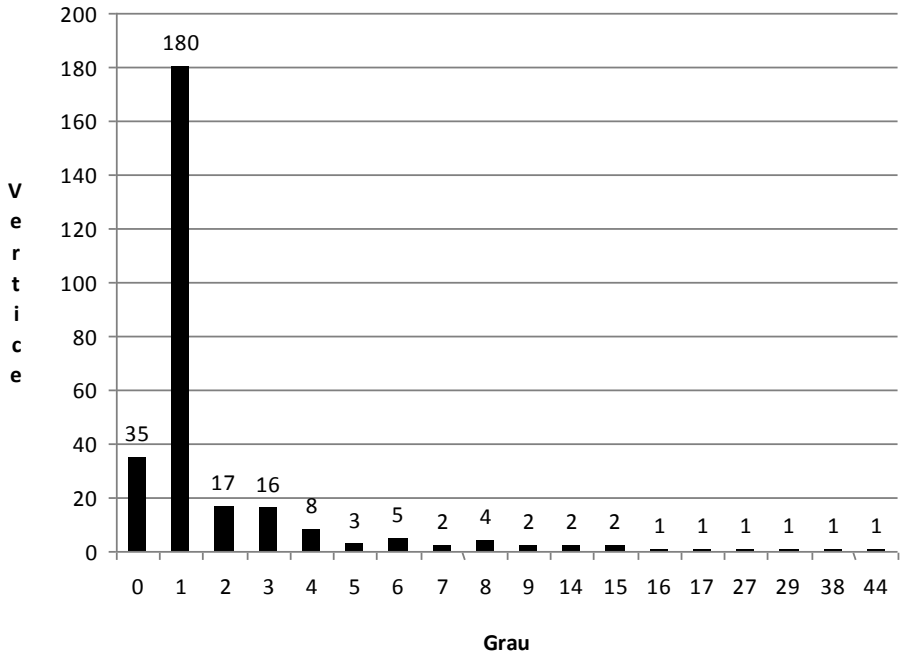
9symml

Grau	Vertices	arestas	% aresta	% acc arestas
TOTAL vertices	206	565		
0	19	0		
1	93	93	0,164602	0,16460177
2	54	108	0,19115	0,355752212
3	25	75	0,132743	0,488495575
4	2	8	0,014159	0,502654867
5	2	10	0,017699	0,520353982
6	1	6	0,010619	0,530973451
14	1	14	0,024779	0,555752212
18	1	18	0,031858	0,587610619
23	1	23	0,040708	0,628318584
25	1	25	0,044248	0,672566372
28	1	28	0,049558	0,722123894
29	2	58	0,102655	0,824778761
30	1	30	0,053097	0,877876106
32	1	32	0,056637	0,934513274
37	1	37	0,065487	1



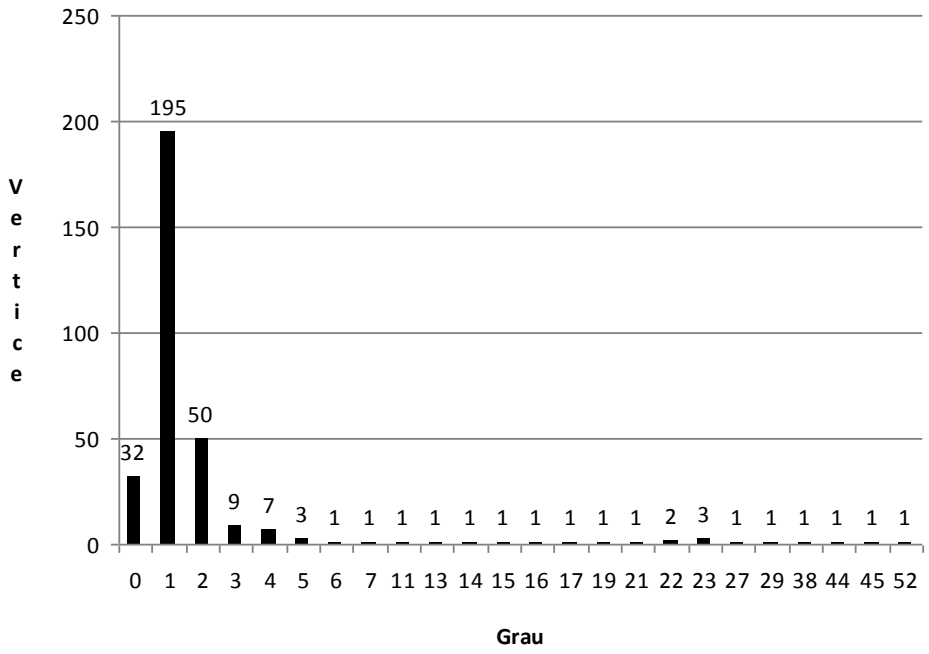
Alu2

Grau	Vertices	arestas	% aresta	% acc arestas
TOTAL vertices	282	632		
0	35	0		
1	180	180	0,28481	0,284810127
2	17	34	0,053797	0,338607595
3	16	48	0,075949	0,414556962
4	8	32	0,050633	0,465189873
5	3	15	0,023734	0,488924051
6	5	30	0,047468	0,536392405
7	2	14	0,022152	0,558544304
8	4	32	0,050633	0,609177215
9	2	18	0,028481	0,637658228
14	2	28	0,044304	0,681962025
15	2	30	0,047468	0,72943038
16	1	16	0,025316	0,754746835
17	1	17	0,026899	0,78164557
27	1	27	0,042722	0,824367089
29	1	29	0,045886	0,870253165
38	1	38	0,060127	0,930379747
44	1	44	0,06962	1



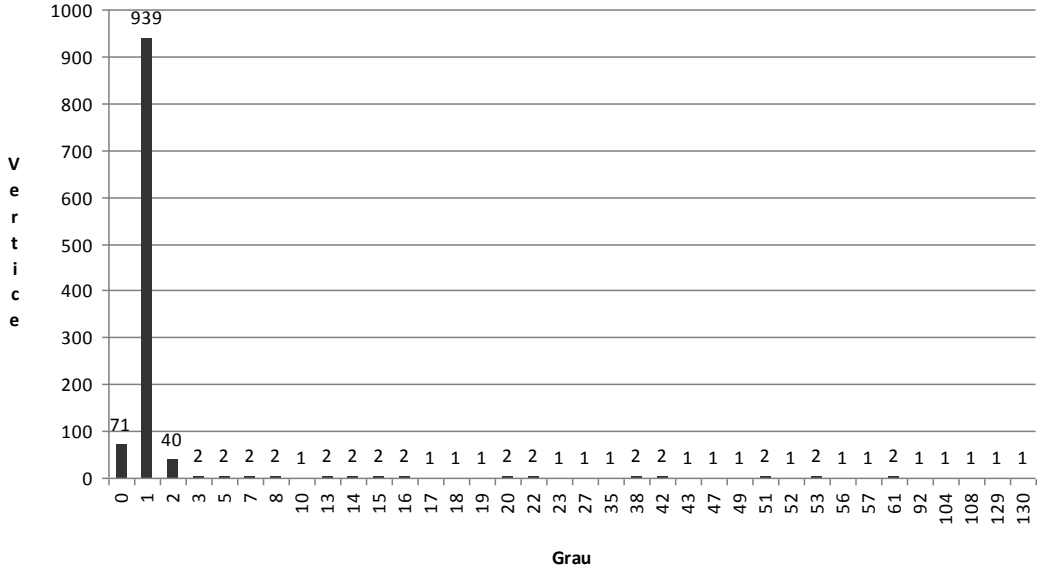
Term1

Grau	Vertices	arestas	% aresta	% acc arestas
TOTAL vertices	317	852		
0	32	0		
1	195	195	0,228873	0,228873239
2	50	100	0,117371	0,346244131
3	9	27	0,03169	0,377934272
4	7	28	0,032864	0,410798122
5	3	15	0,017606	0,428403756
6	1	6	0,007042	0,435446009
7	1	7	0,008216	0,443661972
11	1	11	0,012911	0,45657277
13	1	13	0,015258	0,471830986
14	1	14	0,016432	0,488262911
15	1	15	0,017606	0,505868545
16	1	16	0,018779	0,524647887
17	1	17	0,019953	0,544600939
19	1	19	0,0223	0,566901408
21	1	21	0,024648	0,591549296
22	2	44	0,051643	0,643192488
23	3	69	0,080986	0,724178404
27	1	27	0,03169	0,755868545
29	1	29	0,034038	0,789906103
38	1	38	0,044601	0,834507042
44	1	44	0,051643	0,886150235
45	1	45	0,052817	0,938967136
52	1	52	0,061033	1



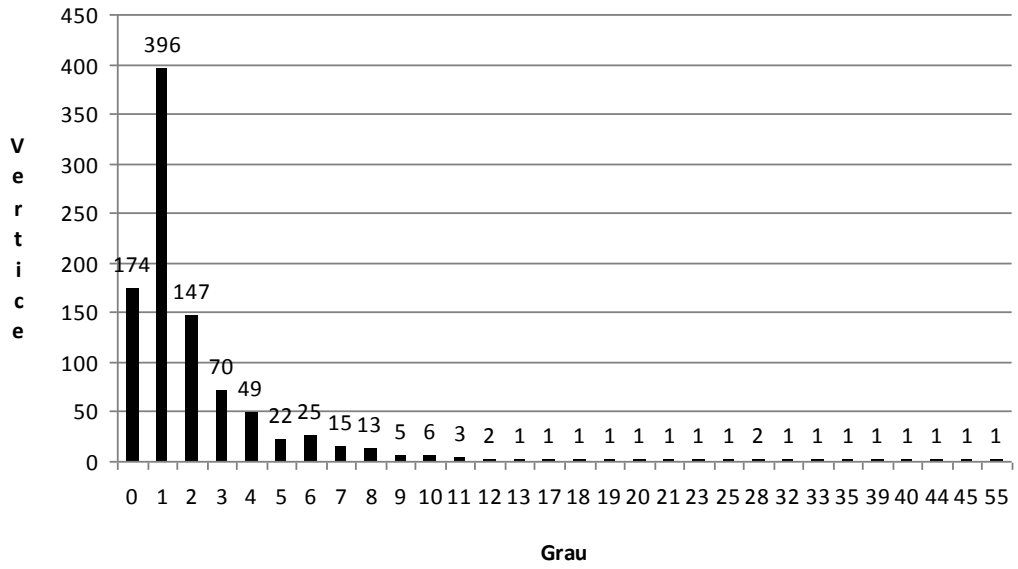
Too_large

Grau	Vertices	arestas	% aresta	% acc arestas
TOTAL vertices	1098	2771		
0	71	0		
1	939	939	0,338867	0,338866835
2	40	80	0,02887	0,367737279
3	2	6	0,002165	0,369902562
5	2	10	0,003609	0,373511368
7	2	14	0,005052	0,378563695
8	2	16	0,005774	0,384337784
10	1	10	0,003609	0,38794659
13	2	26	0,009383	0,397329484
14	2	28	0,010105	0,407434139
15	2	30	0,010826	0,418260556
16	2	32	0,011548	0,429808733
17	1	17	0,006135	0,435943703
18	1	18	0,006496	0,442439553
19	1	19	0,006857	0,449296283
20	2	40	0,014435	0,463731505
22	2	44	0,015879	0,479610249
23	1	23	0,0083	0,487910502
27	1	27	0,009744	0,497654276
35	1	35	0,012631	0,510285096
38	2	76	0,027427	0,537712017
42	2	84	0,030314	0,568025983
43	1	43	0,015518	0,583543847
47	1	47	0,016961	0,600505233
49	1	49	0,017683	0,61818838
51	2	102	0,03681	0,654998196
52	1	52	0,018766	0,673763984
53	2	106	0,038253	0,712017322
56	1	56	0,020209	0,732226633
57	1	57	0,02057	0,752796824
61	2	122	0,044027	0,796824251
92	1	92	0,033201	0,830025262
104	1	104	0,037532	0,867556839
108	1	108	0,038975	0,906531938
129	1	129	0,046554	0,953085529
130	1	130	0,046914	1



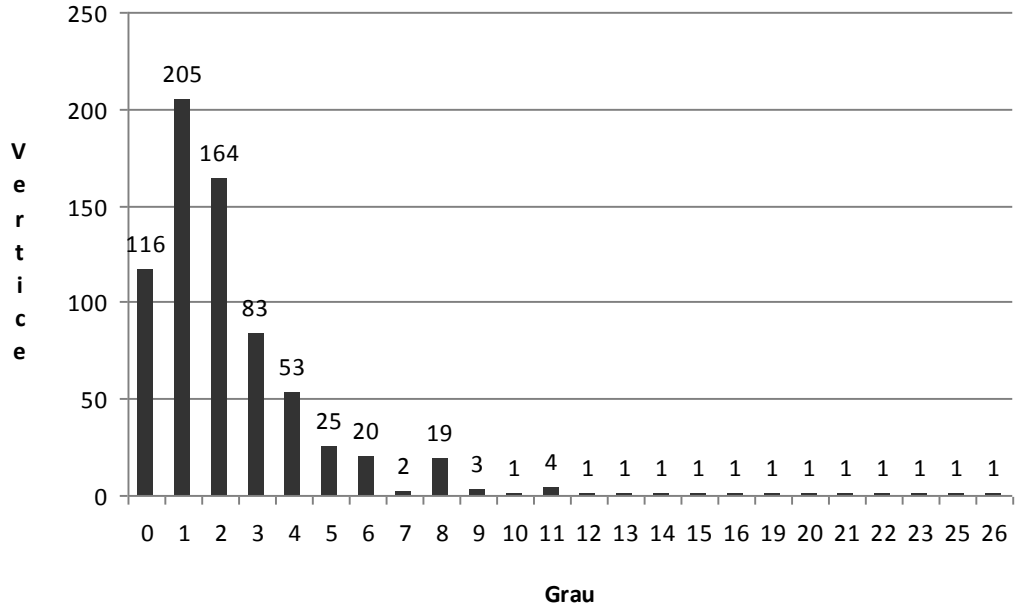
Misex3

Grau	Vertices	arestas	% aresta	% acc arestas
TOTAL vertices	945	2262		
0	174	0		
1	396	396	0,175066	0,175066313
2	147	294	0,129973	0,305039788
3	70	210	0,092838	0,397877984
4	49	196	0,086649	0,484526967
5	22	110	0,04863	0,533156499
6	25	150	0,066313	0,599469496
7	15	105	0,046419	0,645888594
8	13	104	0,045977	0,691865606
9	5	45	0,019894	0,711759505
10	6	60	0,026525	0,738284704
11	3	33	0,014589	0,752873563
12	2	24	0,01061	0,763483643
13	1	13	0,005747	0,769230769
17	1	17	0,007515	0,776746242
18	1	18	0,007958	0,784703802
19	1	19	0,0084	0,793103448
20	1	20	0,008842	0,801945181
21	1	21	0,009284	0,811229001
23	1	23	0,010168	0,821396994
25	1	25	0,011052	0,83244916
28	2	56	0,024757	0,857206012
32	1	32	0,014147	0,871352785
33	1	33	0,014589	0,885941645
35	1	35	0,015473	0,901414677
39	1	39	0,017241	0,918656057
40	1	40	0,017683	0,936339523
44	1	44	0,019452	0,955791335
45	1	45	0,019894	0,975685234
55	1	55	0,024315	1



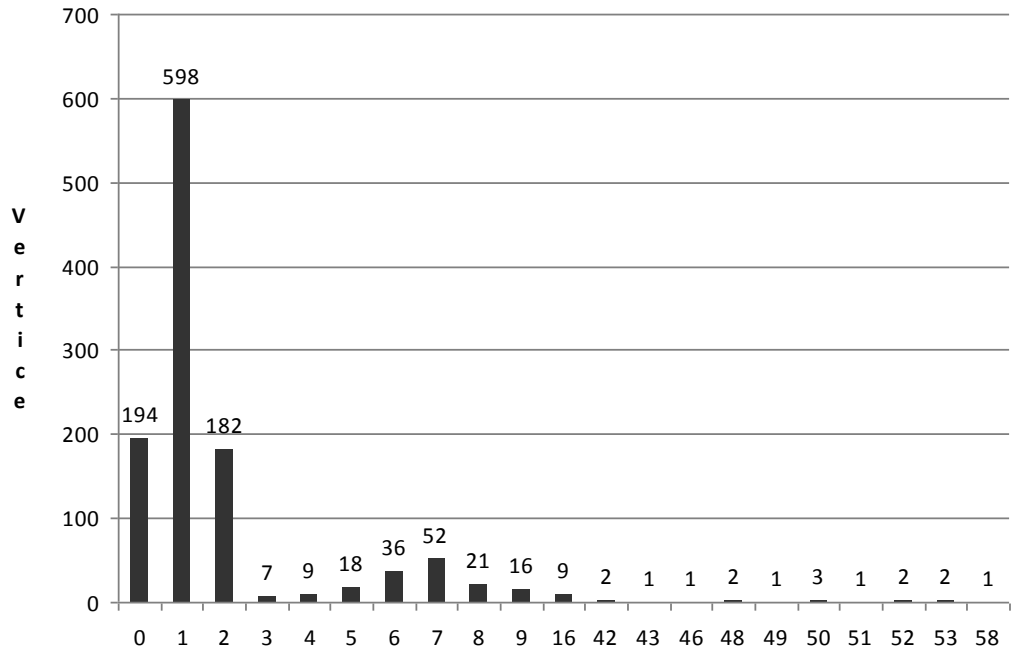
Ex5p

Grau	Vertices	arestas	% aresta	% acc arestas
TOTAL vertices	707	1712		
0	116	0		
1	205	205	0,119743	0,119742991
2	164	328	0,191589	0,311331776
3	83	249	0,145444	0,456775701
4	53	212	0,123832	0,580607477
5	25	125	0,073014	0,653621495
6	20	120	0,070093	0,723714953
7	2	14	0,008178	0,731892523
8	19	152	0,088785	0,82067757
9	3	27	0,015771	0,836448598
10	1	10	0,005841	0,84228972
11	4	44	0,025701	0,867990654
12	1	12	0,007009	0,875
13	1	13	0,007593	0,882593458
14	1	14	0,008178	0,890771028
15	1	15	0,008762	0,89953271
16	1	16	0,009346	0,908878505
19	1	19	0,011098	0,919976636
20	1	20	0,011682	0,931658879
21	1	21	0,012266	0,943925234
22	1	22	0,01285	0,956775701
23	1	23	0,013435	0,97021028
25	1	25	0,014603	0,984813084
26	1	26	0,015187	1



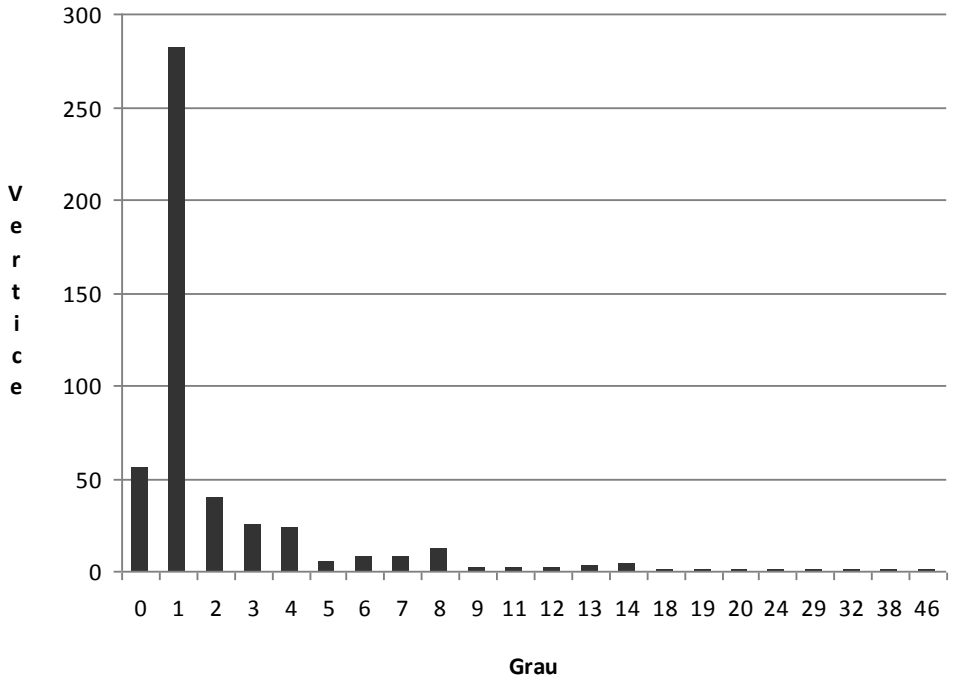
Ex1010

Grau	Vertices	arestas	% aresta	% acc arestas
TOTAL vertices	1158	2932		
0	194	0		
1	598	598	0,203956	0,203956344
2	182	364	0,124147	0,328103683
3	7	21	0,007162	0,33526603
4	9	36	0,012278	0,347544338
5	18	90	0,030696	0,378240109
6	36	216	0,07367	0,451909959
7	52	364	0,124147	0,576057299
8	21	168	0,057299	0,633356071
9	16	144	0,049113	0,682469304
16	9	144	0,049113	0,731582538
42	2	84	0,028649	0,760231924
43	1	43	0,014666	0,774897681
46	1	46	0,015689	0,79058663
48	2	96	0,032742	0,823328786
49	1	49	0,016712	0,840040928
50	3	150	0,05116	0,891200546
51	1	51	0,017394	0,908594816
52	2	104	0,035471	0,944065484
53	2	106	0,036153	0,980218281
58	1	58	0,019782	1



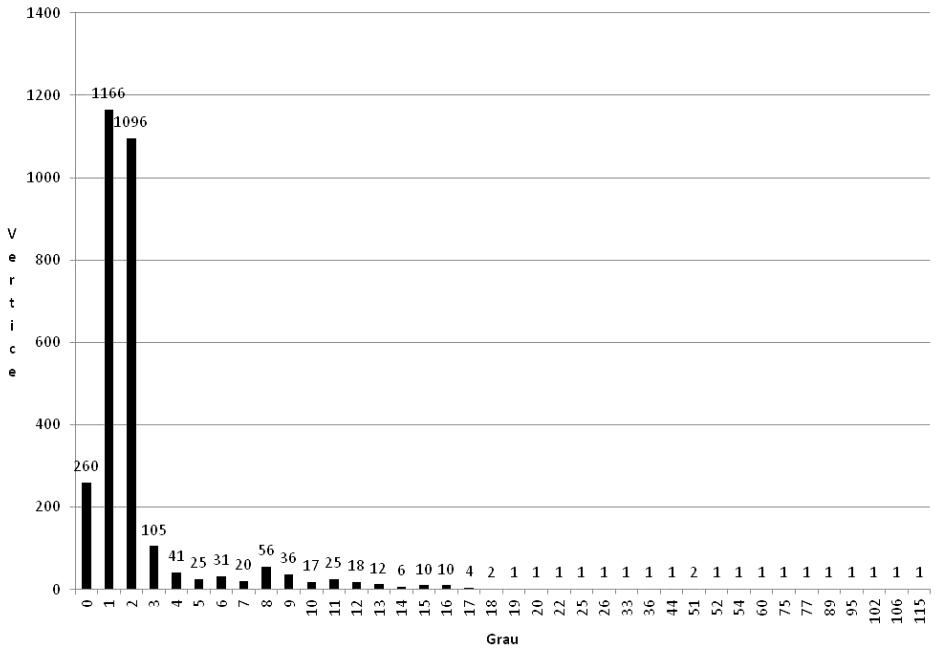
Alu4

Grau	Vertices	arestas	% aresta	% acc arestas
TOTAL vertices	479	1137		
0	56	0		
1	282	282	0,248021	0,248021108
2	39	78	0,068602	0,316622691
3	25	75	0,065963	0,382585752
4	23	92	0,080915	0,46350044
5	5	25	0,021988	0,485488127
6	8	48	0,042216	0,527704485
7	8	56	0,049252	0,576956904
8	12	96	0,084433	0,661389622
9	2	18	0,015831	0,677220756
11	2	22	0,019349	0,696569921
12	2	24	0,021108	0,7176781
13	3	39	0,034301	0,751978892
14	4	56	0,049252	0,80123131
18	1	18	0,015831	0,817062445
19	1	19	0,016711	0,833773087
20	1	20	0,01759	0,851363237
24	1	24	0,021108	0,872471416
29	1	29	0,025506	0,897977133
32	1	32	0,028144	0,926121372
38	1	38	0,033421	0,959542656
46	1	46	0,040457	1



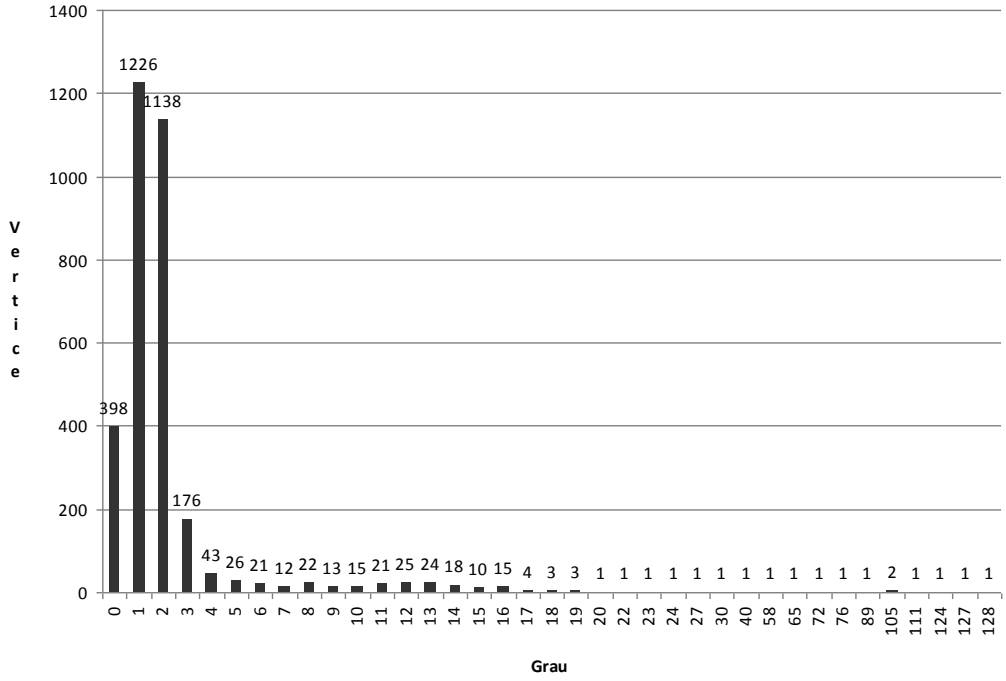
Spla

Grau	Vertices	arestas	% aresta	% acc arestas
TOTAL vertices	2878	7527		
0	260	0		
1	1166	1166	0,154909	0,154908994
2	1096	2192	0,291218	0,446127275
3	105	315	0,041849	0,487976618
4	41	164	0,021788	0,509764847
5	25	125	0,016607	0,526371728
6	31	186	0,024711	0,551082769
7	20	140	0,0186	0,569682476
8	56	448	0,059519	0,629201541
9	36	324	0,043045	0,672246579
10	17	170	0,022585	0,694831938
11	25	275	0,036535	0,731367079
12	18	216	0,028697	0,76006377
13	12	156	0,020725	0,780789159
14	6	84	0,01116	0,791948984
15	10	150	0,019928	0,811877242
16	10	160	0,021257	0,833134051
17	4	68	0,009034	0,842168194
18	2	36	0,004783	0,846950976
19	1	19	0,002524	0,849475223
20	1	20	0,002657	0,852132324
22	1	22	0,002923	0,855055135
25	1	25	0,003321	0,858376511
26	1	26	0,003454	0,861830743
33	1	33	0,004384	0,866214959
36	1	36	0,004783	0,870997741
44	1	44	0,005846	0,876843364
51	2	102	0,013551	0,89039458
52	1	52	0,006908	0,897303042
54	1	54	0,007174	0,904477215
60	1	60	0,007971	0,912448519
75	1	75	0,009964	0,922412648
77	1	77	0,01023	0,932642487
89	1	89	0,011824	0,944466587
95	1	95	0,012621	0,957087817
102	1	102	0,013551	0,970639033
106	1	106	0,014083	0,984721669
115	1	115	0,015278	1



PDC

Grau	Vertices	arestas	% aresta	% acc arestas
TOTAL vertices	3111	7895		
0	398	0		
1	1226	1226	0,155288	0,155288157
2	1138	2276	0,288284	0,443571881
3	176	528	0,066878	0,510449652
4	43	172	0,021786	0,532235592
5	26	130	0,016466	0,54870171
6	21	126	0,015959	0,564661178
7	12	84	0,01064	0,575300823
8	22	176	0,022293	0,597593414
9	13	117	0,01482	0,61241292
10	15	150	0,018999	0,631412286
11	21	231	0,029259	0,660671311
12	25	300	0,037999	0,698670044
13	24	312	0,039519	0,738188727
14	18	252	0,031919	0,770107663
15	10	150	0,018999	0,78910703
16	15	240	0,030399	0,819506016
17	4	68	0,008613	0,828119063
18	3	54	0,00684	0,834958835
19	3	57	0,00722	0,842178594
20	1	20	0,002533	0,844711843
22	1	22	0,002787	0,847498417
23	1	23	0,002913	0,850411653
24	1	24	0,00304	0,853451552
27	1	27	0,00342	0,856871438
30	1	30	0,0038	0,860671311
40	1	40	0,005066	0,865737809
58	1	58	0,007346	0,873084231
65	1	65	0,008233	0,881317289
72	1	72	0,00912	0,890436985
76	1	76	0,009626	0,900063331
89	1	89	0,011273	0,911336289
105	2	210	0,026599	0,937935402
111	1	111	0,01406	0,951994934
124	1	124	0,015706	0,967701077
127	1	127	0,016086	0,983787207
128	1	128	0,016213	1


1400
1200
1000
800
600
400
200
0
Vertice
398
1226
1138
176
43
26
21
12
22
13
15
21
25
24
18
10
15
4
3
3
1
1
1
1
1
1
1
1
1
1
2
1
1
1
1
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
22
23
24
27
30
40
58
65
72
76
89
105
111
124
127
128
Grau