**UNIVERSIDADE FEDERAL DE SANTA CATARINA**
**PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO**

Renan Oliveira Netto

# ACELERAÇÃO DA LEGALIZAÇÃO INCREMENTAL MEDIANTE O USO DE ÁRVORES ESPACIAIS

Florianópolis

2017

Renan Oliveira Netto

# ACELERAÇÃO DA LEGALIZAÇÃO INCREMENTAL MEDIANTE O USO DE ÁRVORES ESPACIAIS

Dissertação submetida ao Programa de Pós-Graduação em Ciência da Computação para a obtenção do Grau de Mestre em Ciência da Computação. Orientador: Prof. Dr. José Luis Almada Güntzel

Florianópolis

2017

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Netto, Renan
Aceleração da Legalização Incremental Mediante o Uso de Árvores Espaciais / Renan Netto ; orientador, José Luís Almada Güntzel - Florianópolis, SC, 2017.
81 p.

Dissertação (mestrado) - Universidade Federal de Santa Catarina, Centro Tecnológico. Programa de Pós-Graduação em Ciência da Computação.

Inclui referências

1. Ciência da Computação. 2. Electronic Design Automation. 3. Síntese física. 4. Legalização. 5. Árvores espaciais. I. Almada Güntzel, José Luís . II. Universidade Federal de Santa Catarina. Programa de Pós-Graduação em Ciência da Computação. III. Título.

Renan Oliveira Netto

**ACELERAÇÃO DA LEGALIZAÇÃO INCREMENTAL MEDIANTE O USO DE ÁRVORES ESPACIAIS**

Esta dissertação foi julgada adequada para obtenção do título de mestre e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação.

Florianópolis, 20 de Fevereiro de 2017.

_____
Profª. Carina Friedrich Dorneles, Drª.
Coordenadora do Programa

**Banca Examinadora:**

_____
Prof. José Luis Almada Güntzel, Dr.
Universidade Federal de Santa Catarina
Orientador

_____
Prof. Marcelo Soares Lubaszewski, Dr.
Universidade Federal do Rio Grande do Sul

_____
Prof. Mario Antonio Ribeiro Dantas, Dr.
Universidade Federal de Santa Catarina

_____
Prof. Laércio Lima Pilla, Dr.
Universidade Federal de Santa Catarina

## AGRADECIMENTOS

Agradeço ao meu orientador José Luis Almada Güntzel, por todo o auxílio oferecido na execução deste trabalho e na escrita deste texto.
Agradeço aos membros da banca, por cederem seu tempo para a avaliação deste trabalho.
Agradeço aos meus colegas de trabalho Vinicius dos Santos Livramento e Chrystian de Sousa Guth, pela ajuda durante a realização deste trabalho.
Agradeço à minha namorada Taciane Martimiano, por me acompanhar nestes dois anos de mestrado.
Por fim, agradeço à minha família Carlos Alberto Barbosa Netto, Loreni Oliveira Netto e Caio Oliveira Netto, por todo apoio e dedicação oferecidos até hoje, pois sem eles, nada disso seria possível.

## RESUMO

Na síntese física de circuitos integrados, a etapa de legalização é responsável por remover sobreposições de células e alinhá-las com as linhas e colunas do circuito, enquanto minimiza o deslocamento das células. Esta etapa é aplicada não somente após o posicionamento global, mas também após etapas de otimização incremental tais como posicionamento incremental guiado por atraso, *gate sizing* e inserção de *buffers*. Quando utilizada em técnicas de otimização incremental, a legalização pode ser aplicada como um passo final, após cada iteração da otimização, ou de maneira incremental, após cada transformação no posicionamento. Infelizmente, técnicas recentes de legalização incremental utilizam estruturas de dados que não são adequadas para o armazenamento de informações sobre geometrias. Além disso, apesar de diferentes estratégias de legalização serem utilizadas por diferentes trabalhos de otimização incremental, estes trabalhos não apresentam resultados quantitativos do impacto destas estratégias no tempo de execução e qualidade da solução final. Este trabalho propõe uma técnica de legalização incremental utilizando uma estrutura de dados chamada R-tree, projetada para o armazenamento de informações sobre geometrias, permitindo buscas espaciais rápidas. A técnica proposta foi comparada a técnicas do estado da arte em legalização incremental, assim como às estratégias de legalização final e iterativa. Os resultados experimentais mostram que a técnica proposta é pelo menos 6 vezes mais rápida e realiza o mesmo número de legalizações quando comparado a outras técnicas de legalização incremental do estado da arte. Além disso, o algoritmo proposto é mais rápido que as estratégias de legalização final e iterativa, enquanto resulta em uma solução com perfil de densidade e comprimento das interconexões semelhante.


**Palavras-chave:** Síntese física, otimização incremental, estratégias de legalização, legalização incremental, buscas espaciais, R-tree

## ABSTRACT

In the physical synthesis of digital circuits, circuit legalization removes overlaps and keeps cell alignment with circuit rows and sites while minimizing total cell displacement. Legalization is applied not only after global placement, but also after incremental optimization steps like incremental timing-driven placement, gate sizing, and buffer insertion. In the case of incremental optimization techniques, the legalization step can be applied as a final step, after each optimization iteration or incrementally, after each cell movement. Unfortunately, recent incremental legalization techniques employ data structures that are not suitable for handling geometry information. In addition, despite different legalization strategies are used by different works on incremental optimization, those works do not present quantitative results on how those strategies impact on the runtime and quality of the final solution. This work proposes a new legalization technique that relies on an R-tree, a data structure tailored to efficient geometry information storage, which allows for fast spatial search. The proposed technique was compared to state-of-the-art incremental legalization techniques, as well as to the final and iterative legalization strategies. Experimental results show that the proposed technique is at least 6 times faster and performs as many successful legalizations when compared to the related work on incremental legalization. In addition, it is faster than both the other two legalization strategies, while resulting in a solution with similar density profile and circuit wirelength.


**Keywords:** Physical design, incremental optimization, legalization strategies, incremental legalization, spatial search, R-tree

## LISTA DE FIGURAS

## LISTA DE TABELAS

## LISTA DE ABREVIATURAS E SIGLAS

## LISTA DE SÍMBOLOS

**SUMÁRIO**

# 1 INTRODUÇÃO

A evolução do processo de fabricação de circuitos integrados e o constante aperfeiçoamento das ferramentas de *Eletronic Design Automation* (EDA) têm viabilizado o projeto de circuitos integrados cada vez mais complexos, os quais possuem restrições de projeto difíceis de serem satisfeitas e usam um número de transistores cada vez maior (KEATING et al., 2007).

A grande complexidade dos circuitos contemporâneos e o curto prazo entre projeto e lançamento do produto no mercado (*time-to-market*) fazem com que a maioria dos projetos de circuitos digitais siga uma metodologia de projeto denominada *standard cell*. Esta metodologia se caracteriza pela adoção de uma biblioteca de células padrão pré-projetadas e pré-caracterizadas, que descrevem as informações lógicas, físicas e elétricas dos elementos combinacionais e sequenciais a serem utilizados na síntese (KAHNG et al., 2011).

Um fluxo *standard cell* típico é dividido em várias etapas como apresentado pela Figura 1, onde cada etapa pode ser subdividida em outras. Particularmente, a etapa de síntese física (*physical design*) destacada na Figura 1 deve considerar as geometrias dos componentes do circuito para posicioná-los e criar os segmentos de fio que os conectam.

Devido à sua complexidade, a síntese física é subdividida em etapas menores, tais como: planejamento topológico, posicionamento global, síntese da árvore de relógio e roteamento (KAHNG et al., 2011). Entre algumas destas etapas são realizadas etapas de otimização incremental para refinar a solução encontrada até então. O escopo deste trabalho está limitado a tais etapas de otimização incremental.

A etapa de posicionamento global determina posições iniciais para todas as células do circuito de forma a otimizar uma função custo, tipicamente o comprimento das interconexões (ALPERT et al., 2012). Para poder lidar com circuitos contemporâneos, constituídos de milhões de células, o posicionamento global ignora algumas restrições de legalidade do circuito e portanto, podem ocorrer sobreposição de células e desalinhamentos em relação às linhas e colunas predeterminadas pelas regras de desenho. Assim, o passo seguinte, denominado de legalização, é responsável por remover as sobreposições entre células e alinhá-las corretamente. Como há um grande número de células em posições ilegais, o algoritmo de legalização empregado nesta etapa move potencialmente todas as células do circuito em um processo chamado de **legalização completa**. Para preservar a qualidade do posiciona-

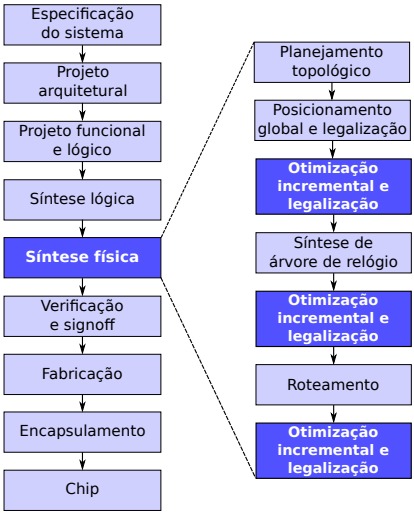


Figura 1 – Exemplo das etapas de um fluxo *standard cell*, com ênfase na etapa de otimização incremental e legalização. Figura adaptada de (KAHNG et al., 2011)

mento global, o algoritmo de legalização deve minimizar o movimento das células do circuito (MARKOV; HU; KIM, 2012).

Técnicas de otimização incremental tais como inserção de *buffers*, *gate sizing* ou posicionamento incremental guiado por atraso são comumente usadas durante o fluxo para otimizar diferentes figuras de mérito, como o comprimento das interconexões, densidade do circuito, potência e atraso (KAHNG et al., 2011). Estas técnicas recebem como entrada um posicionamento legal e, após realizarem a otimização, devem manter a legalidade do mesmo aplicando uma das seguintes estratégias de legalização: **legalização final**, **legalização iterativa**, ou **legalização incremental**.

A primeira estratégia é equivalente à legalização completa realizada após o posicionamento global, e consiste em mover todas as células que ocupam posições ilegais em um único passo, ao final de uma otimização. Esta estratégia é usada pelas técnicas apresentadas em Ren, Pan e Kung (2005) e Luo, Newmark e Pan (2006). A legalização iterativa, por sua vez, realiza o mesmo procedimento após cada iteração da otimização em curso, sendo utilizada pelas técnicas de Choi e Bazargan (2003), Chowdhary et al. (2005) e Guth et al. (2015). Como em ambas estratégias de legalização muitas células são movidas, técnicas

de legalização completa são normalmente utilizadas, tais como: heurísticas gulosas, como em Hill (2002), programação dinâmica, como em Spindler, Schlichtmann e Johannes (2008) e Puget et al. (2015), técnicas baseadas em difusão, como em Ren et al. (2005) ou resolvendo um problema de fluxo em redes, tal como em Cho et al. (2010) e Brenner (2012).

No entanto, legalizar o circuito inteiro várias vezes durante otimizações demanda um grande esforço computacional. Devido a isto, técnicas recentes têm utilizado uma estratégia de **legalização incremental**, que move um subconjunto das células do circuito durante cada transformação do posicionamento em uma otimização incremental (tal como movimento ou redimensionamento de uma célula) (ALPERT; CHU; VILLARRUBIA, 2007). Esta estratégia é utilizada, por exemplo, nos trabalhos de Ren et al. (2007), Papa et al. (2008), Popovych et al. (2014) e Chow et al. (2014). Note que cada transformação no posicionamento pode exigir que um subconjunto de células seja movido para permitir que a transformação como um todo seja realizada de forma legal. Como estas operações devem ser repetidas várias vezes durante uma otimização, a legalização incremental deve ser rápida o bastante para não comprometer o tempo de execução da otimização. Esta estratégia de legalização pode ser utilizada adaptando um algoritmo de legalização completa existente para agir de maneira incremental, tal como feito por Puget et al. (2015) e Popovych et al. (2014), ou utilizando um algoritmo de legalização especializado, como em Chow et al. (2014).

## 1.1 JUSTIFICATIVA

Apesar de vários trabalhos encontrados na literatura fazerem uso das diferentes estratégias de legalização mencionadas na seção anterior, nenhum deles avalia o impacto destas diferentes estratégias quando aplicadas em uma dada técnica de otimização incremental.

Além disso, as técnicas do estado da arte em legalização incremental utilizam estruturas de dados que não foram projetadas para a manipulação de objetos geométricos, tais como os layouts das células de um circuito. O uso de tais estruturas de dados exige que um gerenciamento ou adaptação seja feito para permitir a manipulação de objetos geométricos, o que resulta em um maior tempo de execução.

Portanto, é desejável o desenvolvimento de novas técnicas de legalização incremental que utilizem estruturas de dados especializadas na organização de objetos geométricos. Também é de grande interesse

para os desenvolvedores de ferramentas de síntese física uma comparação quantitativa entre as três estratégias de legalização utilizadas durante a otimização incremental, sobretudo uma comparação que faça uso de uma infraestrutura realista.

## 1.2 OBJETIVOS

Este trabalho tem como objetivo a avaliação quantitativa do impacto de diferentes estratégias de legalização no contexto de otimizações incrementais, assim como a proposta e a avaliação de uma nova técnica de legalização incremental rápida.

Os objetivos específicos deste trabalho são:

- Avaliar o algoritmo de legalização Abacus descrito em (SPINDLER; SCHLICHTMANN; JOHANNES, 2008), no contexto das estratégias de legalização final e iterativa, a partir de uma implementação própria do mesmo;
- Propor e implementar uma técnica de legalização incremental que use eficientemente uma estrutura de dados projetada para armazenamento e manipulação de objetos geométricos;
- Investigar o impacto de cada estratégia de legalização, quando incorporadas na técnica de posicionamento incremental guiado por atraso de Guth et al. (2015). A avaliação das estratégias de legalização é feita em termos de tempo de execução e qualidade da solução final. Esta última será medida em termos de densidade do circuito, comprimento das interconexões e violações de atraso;
- Avaliar a técnica proposta comparando-a com algoritmos do estado da arte em legalização incremental. A comparação é feita considerando o tempo de execução das técnicas, assim como a perturbação gerada no posicionamento inicial e número de legalizações realizadas.

## 1.3 CONTRIBUIÇÕES CIENTÍFICAS E TECNOLÓGICAS

Este trabalho traz as seguintes contribuições científicas e tecnológicas:

- Uma técnica de legalização incremental utilizando uma estrutura de dados especializada na organização dinâmica de dados geomé-

tricos, chamada R-tree. Esta estrutura representa conjuntos de objetos através do menor envoltório em forma de retângulo, permitindo buscas espaciais rápidas, essenciais para o problema de legalização incremental. A técnica proposta adapta um algoritmo do estado da arte em legalização incremental, proposto em Chow et al. (2014);

- Resultados experimentais utilizando uma infraestrutura baseada em circuitos industriais, mostrando que o algoritmo proposto é pelo menos 6 vezes mais rápido que os algoritmos do estado da arte, enquanto legaliza uma quantidade similar de células;

- Comparação quantitativa das três estratégias de legalização presentes na literatura, para uso em técnicas de otimização incremental. Os resultados experimentais mostram que a legalização incremental é a mais rápida das três estratégias, e atinge resultados semelhantes de densidade e comprimento de interconexões, mas resulta em mais violações de atraso quando aplicada no contexto do posicionamento incremental guiado por atraso. A estratégia iterativa, por sua vez, leva a um maior tempo de execução, que não é compensado pela qualidade de sua solução final, a qual é semelhante àquela atingida pela legalização final.

## 1.4 ORGANIZAÇÃO DESTE TRABALHO

O Capítulo 2 descreve os conceitos fundamentais necessários para a compreensão destre trabalho, assim como formula o problema de legalização. O Capítulo 3 apresenta os trabalhos correlatos em legalização, abrangendo tanto algoritmos incrementais como algoritmos que legalizam todo o circuito, enquanto que o Capítulo 4 descreve a técnica de legalização incremental proposta neste trabalho. O Capítulo 5 apresenta os resultados experimentais. Por fim, o Capítulo 6 apresenta as conclusões obtidas com a realização deste trabalho.

# 2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta, inicialmente, os conceitos básicos necessários para a compreensão deste trabalho, tais como aqueles relacionados ao layout e ao posicionamento de circuitos integrados digitais. Ele apresenta as três possíveis estratégias de legalização a serem empregadas no contexto de otimizações incrementais. Finalmente, este capítulo formaliza os problemas de legalização para cada estratégia.

## 2.1 CONCEITOS SOBRE LAYOUT E POSICIONAMENTO DE UM CIRCUITO DIGITAL

Em um fluxo de projeto *standard cell*, a etapa de síntese física denominada posicionamento determina a posição dos layouts dos componentes (portas lógicas e elementos sequenciais), layouts estes que são referidos por células.

O layout de um circuito integrado digital é composto de células e macroblocos, os quais devem ser dispostos em uma superfície bidimensional de modo a ficarem alinhados com as linhas e colunas de uma grade, a fim de evitar erros de desenho. Além disso, a superfície bidimensional possui uma área máxima delimitada por $M = (X_{left}, X_{right}, Y_{bottom}, Y_{top})$. A grade, por sua vez, é formada pelas linhas e pelas colunas que definem as posições válidas, sendo representadas pelos conjuntos $\mathcal{R} = \{r_1, r_2, ..., r_n\}$ e $\mathcal{S} = \{s_1, s_2, ..., s_n\}$, respectivamente. Cada linha $r_i$ e coluna $s_j$ possui altura e largura definidos. Para efeito de simplicidade, este trabalho assume que todas as linhas possuem a mesma altura $H_{row}$, enquanto que todas as colunas possuem a mesma largura $W_{site}$. As células e macroblocos, por sua vez, são definidos por um conjunto $\mathcal{C} = \{c_1, c_2, ...c_n\}$. Por fim, a superfície de um circuito digital é definida pela tupla $U = (M, \mathcal{R}, \mathcal{S}, \mathcal{C})$. Como células e macroblocos podem ser representados pelo mesmo conjunto, para efeito de conveniência, este trabalho irá usar apenas o termo **célula** para se referir a ambos células e macroblocos, no contexto de layout de um circuito digital.

Cada célula $c_i \in \mathcal{C}$ do circuito possui um layout $L(c_i) = (w(c_i), h(c_i), x(c_i), y(c_i))$ onde o par $(w(c_i), h(c_i))$ define a largura e altura da célula, enquanto que o par $(x(c_i), y(c_i))$ define sua posição. O conjunto de layouts de todas as células é dado por $L(C) = \{L(c_1), L(c_2), ..., L(c_n)\}$. Uma transformação no posicionamento $\mathcal{T}$ é uma operação que mapeia um layout de célula $L(c_i)$ para um novo layout $L'(c_i)$. Exemplos

de transformações são o redimensionamento de uma célula e o deslocamento da mesma. Por fim, um posicionamento $\mathcal{P}$ é uma função $\mathcal{P} : \mathcal{C} \rightarrow \mathbb{R}^2$ que mapeia cada célula $c_i$ para uma posição $(x(c_i), y(c_i))$.

A Figura 2 (a) ilustra um posicionamento ilegal de um circuito digital com 12 células ($c_1$ a $c_{12}$), assim como seus respectivos elementos de layout. As células são representadas pelos retângulos coloridos, enquanto que as regiões em branco correspondem a regiões vazias no circuito. Observe que a célula $c_5$ é um macrobloco e por isso, está destacada das demais. Neste exemplo, é possível observar três tipos de violações de restrições de legalidade: 1) sobreposições de células ($c_1$ e $c_3$, $c_4$ e $c_6$); 2) células desalinhadas com as linhas e colunas do circuito ($c_7$, $c_8$, $c_9$ e $c_{10}$); 3) células fora dos limites do circuito ($c_9$ e $c_{12}$).

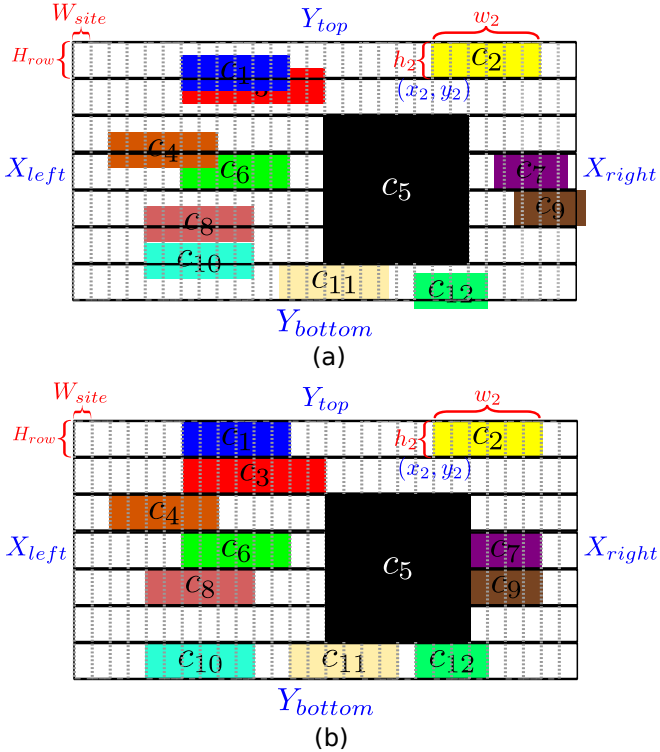


Figura 2 – Exemplos de posicionamentos ilegal (a) e legal (b) de um circuito digital com 12 células

A legalização é responsável por mover as células dos circuitos de

forma a remover estas violações. Este processo pode ser feito usando diferentes estratégias de legalização. A Figura 2 (b) apresenta um exemplo de solução obtida por um algoritmo de legalização que recebe como entrada o posicionamento ilegal da Figura 2 (a).

## 2.2 ESTRATÉGIAS DE LEGALIZAÇÃO DURANTE OTIMIZAÇÕES INCREMENTAIS

Otimizações incrementais recebem como entrada um posicionamento legal e realizam uma série de transformações no posicionamento com o objetivo de otimizar diferentes figuras de mérito, tais como comprimento das interconexões, densidade do circuito e atraso. Como estas técnicas recebem como entrada um posicionamento legal, elas devem manter a legalidade do circuito, o que é feito usando uma das seguintes estratégias de legalização: legalização final, legalização iterativa, legalização incremental.

A Figura 3 apresenta os fluxogramas de uma técnica genérica de otimização incremental utilizando cada uma das estratégias de legalização. Observe que para as três estratégias, a técnica de otimização realiza uma série de transformações no posicionamento por um certo número de iterações, eventualmente legalizando o circuito. A única diferença entre cada estratégia é o momento em que a legalização é feita e, consequentemente, os pontos de entrada e saída do algoritmo de legalização variam conforme o caso.

Utilizando a estratégia de legalização final a legalização é feita após todas as iterações da otimização, conforme apresentado no fluxograma da Figura 3 (a). A estratégia de legalização iterativa, por sua vez, realiza a legalização ao fim de cada iteração, conforme mostra o fluxograma da Figura 3 (b). Vale observar que, como as duas estratégias (final e iterativa) legalizam o circuito após várias transformações no posicionamento, a entrada do algoritmo é um posicionamento ilegal resultante desta sequência de transformações. Por outro lado, a estratégia de legalização incremental legaliza o circuito após cada transformação no posicionamento, conforme mostrado no fluxograma da Figura 3 (c). Desta forma, a legalização incremental recebe como entrada um posicionamento legal, realiza uma transformação e em seguida, legaliza o posicionamento.

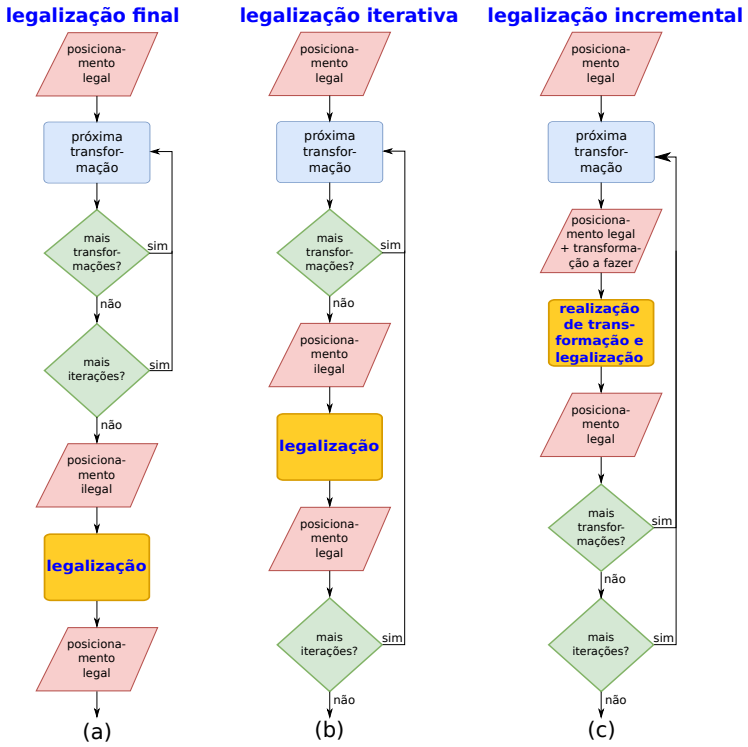


Figura 3 – Fluxogramas de uma técnica genérica de otimização incremental utilizando cada uma das três estratégias de legalização.

## 2.3 FORMULAÇÃO DOS PROBLEMAS DE LEGALIZAÇÃO FINAL E ITERATIVA

Dados a superfície $U$ de um circuito e o conjunto de layouts $L(C)$ de suas células, um posicionamento $\mathcal{P}$ é legal se ele satisfaz as restrições de legalidade definidas pelas Equações (2.1) a (2.6). As restrições das Equações (2.1) a (2.4) são definidas para cada célula $c_i \in \mathcal{C}$ e mantêm as células dentro dos limites do circuito e alinhadas com as linhas e colunas. Além disso, cada par de células na mesma linha deve satisfazer a restrição da Equação (2.5) que evita sobreposições entre células. O mesmo se aplica para as células na mesma coluna, como definido na Equação (2.6).

$$X_{left} \leq x(c_i) \leq X_{right} - w(c_i) \tag{2.1}$$
$$Y_{bottom} \leq y(c_i) \leq Y_{top} - h(c_i) \tag{2.2}$$
$$x(c_i) = n \times W_{site}, n \in \mathbb{N} \tag{2.3}$$
$$y(c_i) = m \times H_{row}, m \in \mathbb{N} \tag{2.4}$$
$$(x(c_k) + w(c_k) < x(c_j)) \vee (x(c_j) + w(c_j) < x(c_k)) \tag{2.5}$$
$$(y(c_k) + h(c_k) < y(c_j)) \vee (y(c_j) + h(c_j) < y(c_k)) \tag{2.6}$$

Dado um posicionamento ilegal $\mathcal{P}$, tal como o posicionamento inicial obtido a partir da etapa de posicionamento global, um algoritmo de legalização $\mathcal{L}$ gera um novo posicionamento $\mathcal{P}'$ que satisfaz as restrições de legalidade do circuito. O posicionamento $\mathcal{P}'$ é gerado através do movimento de um subconjunto de células $\mathcal{C}'$ buscando minimizar uma função custo. A Equação 2.7 descreve uma função custo comumente utilizada, que mede a perturbação no posicionamento através da distância Manhattan[1] entre as posições das células antes e depois da legalização, representadas por $(x(c_i), y(c_i))$ e $(x'(c_i), y'(c_i))$, respectivamente.

$$cost(\mathcal{L}) = \sum_{c_i \in \mathcal{C}} |x(c_i) - x'(c_i)| + |y(c_i) - y'(c_i)| \tag{2.7}$$

Observe que esta formulação assume que o algoritmo de legalização recebe como entrada um posicionamento inicial ilegal e potencialmente move todas as células com o objetivo de minimizar a função custo definida. Desta forma, esta formulação é utilizada nas estratégias de legalização final e legalização iterativa.

## 2.4 FORMULAÇÃO DO PROBLEMA DE LEGALIZAÇÃO INCREMENTAL

A estratégia de legalização incremental difere das outras duas estratégias pelo fato de ser aplicada a cada transformação do posicionamento que ocorre durante uma otimização incremental. Assim, a legalização incremental é usada para manter a legalidade do posicionamento imediatamente após a realização de uma transformação. Por exemplo, suponha que uma técnica de otimização deseje mover uma

[1] Dados dois pontos $(x, y)$ e $(x', y')$, a distância Manhattan é definida como $|x - x'| + |y - y'|$

célula $c_i$ para uma região que já está ocupada por outras células. O algoritmo de legalização incremental deve então mover as células desta região de forma a permitir que $c_i$ seja movida sem violar as restrições de legalidade do circuito.

Desta forma, um algoritmo de legalização incremental $\mathcal{L}_{inc}$ recebe como entrada um posicionamento legal $\mathcal{P}$ e uma transformação $\mathcal{T}$ a ser realizada. O algoritmo, então, gera um novo posicionamento $\mathcal{P}'$ que realiza a transformação desejada sem violar as restrições de legalidade das Equações (2.1) a (2.6), enquanto minimiza uma função custo, tal como a da Equação (2.7). Entretanto, para permitir que uma transformação seja realizada, pode ser necessário mover outras células, de forma a manter a legalidade do circuito. Portanto, um subconjunto de células $\mathcal{C}''$ é movido durante cada legalização incremental. Por outro lado, é possível especular que normalmente $|\mathcal{C}''| \ll |\mathcal{C}'|$, dado que apenas uma transformação é realizada no posicionamento $\mathcal{P}$.

# 3 TRABALHOS CORRELATOS

Este capítulo apresenta os principais trabalhos correlatos no contexto do problema de legalização. Inicialmente, são revisados os principais algoritmos de legalização completa. Em seguida, são descritos os principais algoritmos propostos para resolver o problema de legalização incremental. É importante ressaltar que este capítulo não faz uma análise exaustiva dos algoritmos de legalização, mas busca apresentar os trabalhos mais relevantes relacionados às diferentes abordagens utilizadas para resolver o problema.

## 3.1 TRABALHOS CORRELATOS EM LEGALIZAÇÃO COMPLETA

### 3.1.1 Hill (2002)

O algoritmo Tetris, de Hill (2002), utiliza uma heurística gulosa para legalizar um circuito. Inicialmente, o algoritmo ordena as células a partir de suas coordenadas no eixo x e as legaliza uma de cada vez. Para cada célula $c_i$, o algoritmo avalia o custo de posicioná-la em cada linha $r_j$ do circuito. Para cada linha, Tetris identifica a posição alvo $(x'(c_i), y'(c_i))$ mais próxima da posição $(x(c_i), y(c_i))$ da célula antes da legalização. O custo de posicionar $c_i$ em $r_j$ é dado pela distância Manhattan entre $(x'(c_i), y'(c_i))$ e $(x(c_i), y(c_i))$. Por fim, o algoritmo posiciona a célula na linha que resulta em menor custo.

Após ser posicionada por Tetris em uma certa linha, a célula $c_i$ não é mais movida durante o restante da legalização. Como cada célula é inserida em uma posição legal e depois não é mais movida, ao final do algoritmo o circuito inteiro é legalizado. No entanto, ao não permitir que uma célula seja movida após ser legalizada, Tetris limita as soluções possíveis para as células seguintes, o que consiste na principal limitação do algoritmo. Como resultado, Tetris encontra soluções subótimas que podem levar a um maior deslocamento das células.

### 3.1.2 Spindler, Schlichtmann e Johannes (2008)

O algoritmo Abacus, de Spindler, Schlichtmann e Johannes (2008), busca superar as limitações de Tetris utilizando uma formulação baseada em programação dinâmica para escolher a posição ótima de cada

célula. Abacus mantém a estratégia de legalizar uma célula por vez e, para cada célula, ele avalia o custo de posicioná-la em cada linha do circuito. A principal diferença em relação ao Tetris reside no fato de que, ao avaliar o custo de posicionar uma célula em uma determinada linha, Abacus reposiciona todas as células daquela linha, de forma a minimizar a função custo expressa na Equação (3.1). Esta função é definida como a soma ponderada das distâncias euclidianas entre a posição legalizada e a posição inicial de cada célula. Observe que o conjunto de células em uma linha $r_j$ é dado por $C(r_j)$, e cada célula $c_i$ possui um peso $\omega(c_i)$.

$$cost(L) = \sum_{r_j \in \mathcal{R}} \sum_{c_i \in C(r_j)} \omega(c_i) \sqrt{(x(c_i) - x'(c_i))^2 + (y(c_i) - y'(c_i))^2} \tag{3.1}$$

Como Abacus minimiza a soma ponderada dos deslocamentos, é necessário atribuir um peso para cada célula do circuito. O trabalho de Spindler, Schlichtmann e Johannes (2008) propõe utilizar o número de pinos (entradas e saídas) da célula como seu peso. No entanto, é possível utilizar métricas de criticalidade de *timing* ou potência para otimizar diferentes figuras de mérito.

Ao avaliar a posição de todas as células em uma linha para cada legalização, Abacus reduz o deslocamento das células quando comparado ao Tetris. A principal limitação do Abacus é o seu tempo de execução elevado. Devido a isso, normalmente o número de linhas do circuito consideradas para reposicionar uma dada célula é limitado às linhas mais próximas da posição original da célula.

### 3.1.3 Puget et al. (2015)

O algoritmo Jezz, de Puget et al. (2015), utiliza uma formulação baseada em programação dinâmica semelhante ao Abacus. A principal diferença entre os dois algoritmos é que, ao avaliar o custo de posicionar uma célula $c_i$ em uma determinada linha, Jezz considera diferentes deslocamentos de $c_i$ com relação à posição alvo, e avalia o deslocamento necessário das outras células como custo. O trabalho de Puget et al. (2015) ainda propõe um critério de desempate quando linhas diferentes resultam no mesmo custo.

Como resultado, Jezz produz um deslocamento médio de células semelhante ao algoritmo Abacus, sendo melhor para alguns circuitos.

A principal limitação do algoritmo é o seu tempo de execução elevado, que é superior ao do Abacus.

### 3.1.4 Ho e Liu (2010)

O algoritmo proposto em Ho e Liu (2010) também é bastante semelhante ao Abacus, mas com uma série de otimizações para torná-lo mais rápido e para reduzir o deslocamento das células. Para reduzir o tempo de execução, o algoritmo agrupa linhas próximas em *clusters* e as armazena em uma árvore binária. Desta forma, é possível obter rapidamente o conjunto de linhas para as quais é necessário avaliar o custo de posicionar uma determinada célula. Além disso, como as células não são posicionadas durante a avaliação de uma linha, esta etapa é otimizada para que seja mais rápida.

Para diminuir o deslocamento das células durante a legalização, o trabalho de Ho e Liu (2010) não utiliza a mesma função custo do Abacus, baseada na distância euclidiana das posições das células antes e depois da legalização. Ao invés disso, o trabalho propõe uma função custo utilizando distância Manhattan, descrita pela Equação (3.2). O uso desta função custo reduz a perturbação causada pela legalização, porém exige mudanças no algoritmo, as quais são detalhadas em Ho e Liu (2010). Como resultado, o algoritmo de Ho e Liu (2010) é mais rápido e resulta em menor deslocamento das células quando comparado com Abacus.

$$cost(L) = \sum_{r_j \in \mathcal{R}} \sum_{c_i \in C(r_j)} w_i(|x_i - x'_i| + |y_i - y'_i|) \tag{3.2}$$

### 3.1.5 Kennings, Darav e Behjat (2014)

O trabalho de Kennings, Darav e Behjat (2014) propõe um algoritmo de posicionamento detalhado considerando restrições adicionais de roteamento. Desta forma, os autores adaptaram Abacus para considerar tais restrições.

O algoritmo de Kennings, Darav e Behjat (2014) trata os obstáculos de roteamento como macro blocos, e armazena todos eles em uma estrutura de dados chamada R-tree (MANOLOPOULOS et al., 2010). Esta estrutura de dados consiste em uma árvore espacial especializada no armazenamento de dados multidimensionais, e será melhor detalhada na

Seção 4.2. Utilizando esta estrutura de dados, a legalização consegue rapidamente identificar sobreposições entre células e obstáculos.

Para reduzir a perturbação no posicionamento inicial ao aplicar a legalização, o algoritmo identifica regiões sem sobreposição do circuito que sejam capazes de acomodar todas as células contidas nelas. Em seguida, Abacus é aplicado em cada região de forma independente. Ao final da legalização, um passo extra é responsável por deslocar as células de forma a satisfazer restrições de roteamento.

A principal limitação deste trabalho é que ele utiliza a árvore espacial apenas para armazenar os layouts dos macro blocos, e não de todas as células do circuito. Como o número de células de um circuito é muito maior que o número de macro blocos, é possível acelerar a legalização utilizando a árvore espacial também para armazenar seus layouts, conforme será detalhado no Capítulo 4.

### 3.1.6 Ren et al. (2005)

O algoritmo de Ren et al. (2005) baseia-se no processo físico de difusão para resolver o problema da legalização. Neste processo físico, cada ponto $(x, y)$ em um plano possui velocidade determinada pelo gradiente da densidade neste ponto e pela própria densidade $d_{x,y}(t)$ em um instante de tempo $t$. As Equações (3.3) e (3.4) descrevem essa relação, onde $v^H_{x,y}(t)$ e $v^V_{x,y}(t)$ descrevem as velocidades do ponto $(x, y)$ para um instante de tempo $t$, nos eixos horizontal e vertical, respectivamente.

$$v^H_{x,y}(t) = \frac{-\frac{\partial d_{x,y}(t)}{\partial x}}{d_{x,y}(t)} \tag{3.3}$$

$$v^V_{x,y}(t) = \frac{-\frac{\partial d_{x,y}(t)}{\partial y}}{d_{x,y}(t)} \tag{3.4}$$

A partir das posições de todos os pontos no instante de tempo $t = 0$, é possível calcular suas posições em um novo instante $t \neq 0$ utilizando as Equações (3.5) e (3.6).

$$x(t) = x(0) + \int_0^t v^H_{x(t'),y(t')}(t')dt' \tag{3.5}$$

$$y(t) = y(0) + \int_0^t v^V_{x(t'),y(t')}(t')dt' \tag{3.6}$$

Para utilizar esta formulação no problema de legalização é necessário discretizá-la. O trabalho de Ren et al. (2005) propõe dividir a área do circuito em *bins* e calcular a densidade de um *bin* $(j, k)$ em um instante $n$ como $d_{j,k}(n)$. Desta forma, a velocidade de uma célula pertencente a um *bin* $(j, k)$ é dada pelas Equações (3.7) e (3.8) e depende apenas das densidades do *bin* ao qual a célula pertence, e aos *bins* vizinhos.

$$v^H_{j,k}(n) = -\frac{d_{j+1,k}(n) - d_{j-1,k}(n)}{2d_{j,k}(n)} \tag{3.7}$$

$$v^V_{j,k}(n) = -\frac{d_{j,k+1}(n) - d_{j,k-1}(n)}{2d_{j,k}(n)} \tag{3.8}$$

Para permitir que células dentro de um mesmo *bin* tenham velocidades diferentes, assim como evitar mudanças bruscas de velocidade entre células de diferentes *bins*, o algoritmo interpola as velocidades das células dentro de cada *bin*. Desta forma, a legalização gradativamente movimenta as células no circuito de forma a remover suas sobreposições. A principal limitação do algoritmo é que sozinho ele não é capaz de legalizar todo o circuito. Desta forma, ao final da sua execução é necessário utilizar outro algoritmo de legalização para legalizar as células que permanecem em posições ilegais.

### 3.1.7 Cho et al. (2010)

O trabalho de Cho et al. (2010) modela o problema de legalização como um problema de fluxo máximo de custo mínimo em grafos. Inicialmente, a área do circuito é dividida em regiões sem intersecções com obstáculos. Uma região é considerada sobrecarregada se a sua área é menor do que o necessário para acomodar todas as células contidas nela. Desta forma, o posicionamento é modelado como um grafo, onde cada região sobrecarregada corresponde a um nodo fonte (*source*)

$s_i \in \mathcal{S}$ e cada região não sobrecarregada corresponde a um nodo destino (*sink*) $t_i \in \mathcal{T}$.

O grafo ainda possui um nodo para cada célula $c_i$ do circuito, que é conectado ao nodo $s_i$ correspondente a sua região atual. Além disso, o nodo de uma célula possui arestas para cada nodo destino $t_i$, de forma que uma célula pode ser movida para qualquer região não sobrecarregada. O custo destas arestas é proporcional ao movimento necessário para transferir $c_i$ para a região representada por $t_i$.

Após construir o grafo, as células são assinaladas para as regiões do circuito, resolvendo o problema de fluxo máximo de custo mínimo. Após encontrar a região para cada célula, as células são devidamente posicionadas nestas regiões, em um processo chamado de **realização de fluxo**. Vale observar que o modelo de grafo utilizado não possui informações das dimensões das células e regiões. Estas informações devem ser consideradas na etapa de realização do fluxo, e este pode não ser realizável. Neste caso, um novo grafo é construído para uma próxima iteração. Este novo grafo é construído em um processo de aprendizado histórico, com o objetivo de aumentar a probabilidade de que o novo grafo seja realizável.

Como resultado, o algoritmo de Cho et al. (2010) atinge um menor deslocamento das células para legalizar o circuito, porém com duas principais limitações. A primeira limitação é que a solução do problema de fluxo máximo de custo mínimo ainda exige que algoritmos adicionais sejam executados para legalizar o circuito. A segunda limitação é o elevado tempo de execução. Para contornar a segunda limitação, o trabalho de Cho et al. (2010) propõe algumas heurísticas para acelerar a execução do algoritmo.

### 3.1.8 Brenner (2012)

O algoritmo de Brenner (2012) também modela o problema de legalização como um problema de fluxo máximo de custo mínimo, porém com algumas melhorias. A principal contribuição deste trabalho é a proposta de uma modificação no algoritmo que resolve o problema de fluxo máximo de custo mínimo. Enquanto assinala as células para as regiões do circuito, o algoritmo modificado considera as dimensões das mesmas, garantindo que o fluxo encontrado é realizável. A consequência desta modificação é que a solução encontrada não garante custo mínimo.

Ao garantir que o fluxo encontrado é sempre realizável, não são

necessárias novas iterações, reduzindo assim o tempo de execução. A principal limitação do algoritmo é que ainda é necessário o uso de outro algoritmo para posicionar as células nas regiões selecionadas, após o assinalamento encontrado pelo grafo.

## 3.2 TRABALHOS CORRELATOS EM LEGALIZAÇÃO INCREMENTAL

### 3.2.1 Popovych et al. (2014)

O trabalho de Popovych et al. (2014) propõe um algoritmo de posicionamento detalhado que tem como objetivo minimizar o comprimento das interconexões e a densidade do circuito. Isto é atingido trocando células de posição, ou movendo células para espaços livres do circuito. A legalização é feita de maneira incremental, após cada troca ou movimento. Para realizar a legalização incremental, os autores adaptaram o algoritmo Abacus para que este seja executado apenas em uma linha do circuito. Enquanto que Abacus avalia o custo de posicionar uma dada célula $c_i$ em cada linha $r_j \in \mathcal{R}$, a versão incremental avalia apenas a linha contendo a posição alvo da célula. A célula é legalizada utilizando a mesma formulação em programação dinâmica proposta em Spindler, Schlichtmann e Johannes (2008).

Ao limitar a legalização para apenas uma linha do circuito, a adaptação reduz drasticamente seu tempo de execução, permitindo que esta possa ser invocada várias vezes em uma otimização incremental. Para gerenciar as posições das células, o algoritmo armazena cada linha do circuito como uma lista encadeada de *clusters* de células e espaços em branco. Para acessar elementos nesta lista de forma eficiente o algoritmo mantém, para cada célula, um ponteiro da posição da mesma na lista. Desta forma, uma troca de células pode ser feita em tempo constante. Além disso, o algoritmo mantém, para cada espaço livre, suas células vizinhas, de forma que o movimento de uma célula para um espaço livre também pode ser feito em tempo constante.

No entanto, é importante notar que a estrutura de dados utilizada pelo algoritmo de Popovych et al. (2014) só garante tempo de busca constante para trocas de células, e não para qualquer transformação no posicionamento. Além disso, é necessário armazenar ponteiros para várias posições da lista, introduzindo um sobrecusto de memória.

### 3.2.2 Puget et al. (2015)

O trabalho de Puget et al. (2015) também propõe uma versão incremental de seu algoritmo de legalização Jezz. De maneira similar à adaptação feita por Popovych et al. (2014), a versão incremental de Jezz limita a legalização a apenas uma linha do circuito, reduzindo drasticamente o tempo de execução.

Assim como o algoritmo de Popovych et al. (2014), Jezz armazena cada linha do circuito em uma lista encadeada de *clusters* de células e espaços em branco. No entanto, Jezz utiliza uma estratégia diferente para acessar estra estrutura de dados de forma eficiente. Inicialmente, o algoritmo divide as linhas do circuito em regiões de mesma largura. Cada região mantém um ponteiro para uma célula dentro dela, que é atualizado quando a célula é movimentada. Desta forma, Jezz usa este ponteiro como ponto de início para buscas sequenciais, reduzindo o tempo de execução. Como resultado, Jezz reduz o *overhead* de memória, pois não precisa armazenar ponteiros para cada célula do circuito. Porém, o tempo de busca deixa de ser constante.

### 3.2.3 Chow et al. (2014)

Diferentemente dos algoritmos de Spindler, Schlichtmann e Johannes (2008) e Puget et al. (2015), que adaptam algoritmos de legalização completa para serem utilizados de forma incremental, o trabalho de Chow et al. (2014) propõe um algoritmo específico para legalização incremental, chamado de *Legalized Cell Move*. Dado um movimento de uma célula para uma posição alvo, *Legalized Cell Move* desloca as células ocupando esta posição para permitir que o movimento seja realizado sem violar a legalidade do circuito. O deslocamento aplicado para cada célula é o mínimo necessário para acomodar a nova transformação no posicionamento. Caso isso não seja possível, o movimento não é realizado.

Por ter um comportamento mais simples, *Legalized Cell Move* é mais rápido que os outros algoritmos de legalização incremental, porém resulta em um maior deslocamento das células. Além disso, o algoritmo modela cada linha do circuito como uma lista encadeada de células. Desta forma, a complexidade da busca de uma célula na lista é linear, dado que é necessário percorrer a lista para encontrar as células que intersectam a posição alvo. Este tempo de busca linear constitui a principal limitação do algoritmo.

## 3.3 ANÁLISE QUALITATIVA DOS TRABALHOS CORRELATOS

A Tabela 1 resume os trabalhos correlatos, classificando-os como legalização completa ou legalização incremental, apresentando a abordagem utilizada para resolver o problema de legalização e a estrutura de dados empregada. Note que diversas abordagens são utilizadas para resolver o problema de legalização completa, porém nenhum destes algoritmos pode ser diretamente aplicado para a legalização incremental. Por outro lado, um número pequeno de trabalhos resolve a legalização incremental. Alguns deles, como os trabalhos de Puget et al. (2015) e Spindler, Schlichtmann e Johannes (2008), adaptam um algoritmo de legalização completa para utilizá-lo na estratégia incremental, enquanto que o trabalho de Chow et al. (2014) propõe um algoritmo especializado para legalização incremental.

A principal limitação dos algoritmos de legalização incremental citados reside no fato de utilizarem estruturas de dados (tais como listas encadeadas) que não foram projetadas para armazenar objetos geométricos. Embora seja possível contornar esta limitação utilizando um sistema de cache, como feito em Puget et al. (2015) ou armazenando ponteiros para posições na lista, como proposto em Popovych et al. (2014), tais soluções introduzem um sobrecusto para gerenciar a estrutura de dados. Portanto, este trabalho de mestrado propõe uma técnica de legalização incremental que adapta a heurística gulosa de Chow et al. (2014), mas fazendo uso de uma estrutura de dados mais adequada, chamada de R-tree. Esta estrutura de dados é a mesma utilizada em Kennings, Darav e Behjat (2014). Porém, diferentemente daquele trabalho, o presente trabalho propõe armazenar os layouts de todas as células na R-tree, e não apenas os layouts dos macro blocos. Além disso, optou-se por utilizar a heurística gulosa de Chow et al. (2014) pois esta resulta em um menor tempo de execução.

| | Completa/Incremental | Abordagem | Estrutura de dados |
|---|---|---|---|
| Hill (2002) | Completa | Heurística gulosa | Lista encadeada |
| Spindler, Schlichtmann e Johannes (2008) | Completa | Programação dinâmica | Lista encadeada |
| Puget et al. (2015) | Completa e incremental | Programação dinâmica | Lista encadeada + cache |
| Kennings, Darav e Behjat (2014) | Completa | Programação dinâmica | Lista encadeada + R-tree |
| Ren et al. (2005) | Completa | Difusão | Não informada |
| Cho et al. (2010) | Completa | Fluxo em redes | Grafo |
| Brenner (2012) | Completa | Híbrida | Grafo |
| Popovych et al. (2014) | Incremental | Programação dinâmica | Lista encadeada + cache |
| Chow et al. (2014) | Incremental | Heurística gulosa | Lista encadeada |
| Este trabalho | Incremental | Heurística gulosa | R-tree |

Tabela 1 – Resumo dos trabalhos correlatos

# 4 LEGALIZAÇÃO INCREMENTAL UTILIZANDO R-TREE

Este capítulo apresenta a técnica proposta para resolver o problema de legalização incremental formulado na Seção 2.4, a qual adapta o algoritmo de Chow et al. (2014) para utilizar uma estrutura de dados especializada no armazenamento de dados geométricos chamada R-tree. Inicialmente, este capítulo descreve, por meio de um exemplo, o processo de legalização incremental. Em seguida, apresenta como a R-tree realiza operações espaciais de forma rápida, para enfim apresentar os detalhes da adaptação proposta.

## 4.1 PROCESSO DE LEGALIZAÇÃO INCREMENTAL

Antes de descrever a técnica proposta de legalização incremental utilizando R-tree, faz-se necessário detalhar alguns aspectos relevantes da execução da legalização incremental. Como apresentado na Figura 3, a legalização incremental é aplicada durante cada transformação no posicionamento. Este processo será ilustrado através do exemplo da Figura 4, que apresenta um posicionamento legal e uma transformação a ser realizada. Neste exemplo, a transformação realizada é o movimento da célula $c_8$ para a região tracejada em vermelha. Esta região, que representa a posição para a qual a célula deve ser movida, é denominada posição alvo de $c_8$.

Apesar de ser possível aplicar a legalização incremental para outras transformações no posicionamento, por simplicidade, este capítulo se limita ao uso desta estratégia para mover células no circuito, sem perda de generalidade da técnica apresentada. Além disso, o exemplo apresentado e a técnica proposta neste capítulo tratam apenas o problema de sobreposições de células. As outras restrições de legalidade (tais como alinhamento com linhas e colunas, e células dentro dos limites do circuito) são mais fáceis de serem satisfeitas e, portanto, não costumam ser tratadas por algoritmos de legalização incremental. No entanto, os algoritmos apresentados neste capítulo podem ser facilmente adaptados para serem utilizados com outras transformações no posicionamento, assim como para resolver outras violações de legalidade.

Na Figura 4 (a), a posição alvo de $c_8$ resulta em sobreposições com outras células do circuito ($c_{10}$ e $c_{11}$). O algoritmo de legalização

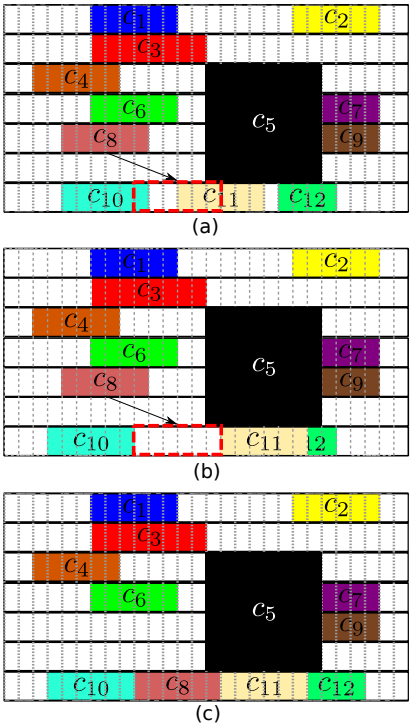


Figura 4 – Exemplo de legalização incremental. (a) A célula $c_8$ deve ser movida para uma nova posição que intersecta com $c_{10}$ e $c_{11}$. (b) As células $c_{10}$ e $c_{11}$ são deslocadas para remover sobreposições com $c_8$. (c) A célula $c_{12}$ é deslocada e $c_8$ é movida para a posição alvo.

incremental deve identificar estas sobreposições e mover $c_{10}$ e $c_{11}$ de forma a remover tais sobreposições. O movimento realizado em $c_{10}$ e $c_{11}$ depende do algoritmo utilizado. Porém para reduzir o espaço de solução, todos os algoritmos presentes na literatura realizam apenas movimentos horizontais nas células da mesma linha da posição alvo, conforme ilustrado na Figura 4 (b). O movimento realizado em $c_{11}$ resulta em uma nova sobreposição, com a célula $c_{12}$. Assim, o algoritmo de legalização incremental deve mover $c_{12}$ de forma a remover todas as sobreposições, resultando no posicionamento da Figura 4 (c).

Para identificar sobreposições entre posições alvo e células do circuito, é necessário armazenar os layouts das células do circuito. Uma forma possível de armazenar estas informações é utilizando uma lista

encadeada, conforme feito nos algoritmos de legalização incremental de Popovych et al. (2014), Puget et al. (2015) e Chow et al. (2014). Para identificar as sobreposições entre a posição alvo de $c_8$ e as células $c_{10}$ e $c_{11}$ utilizando este tipo de estrutura de dados, é necessário percorrê-la e, para cada célula $c_i$ na lista, verificar se existe intersecção entre o layout de $c_i$ e a posição alvo. Esta operação de busca possui complexidade $\mathcal{O}(n)$, já que a lista encadeada não é uma estrutura de dados especializada no armazenamento de objetos geométricos.

Os trabalhos de legalização incremental presentes na literatura armazenam os layouts das células do circuito em listas encadeadas. No entanto, é possível identificar sobreposições entre posições alvo e células de forma mais rápida utilizando uma estrutura de dados mais adequada, tal como uma árvore espacial. A técnica apresentada no presente trabalho faz uso de uma estrutura de dados chamada R-tree, a qual será apresentada na Seção 4.2.

## 4.2 FUNCIONAMENTO DA ESTRUTURA DE DADOS R-TREE

Uma R-tree é uma árvore não binária que armazena objetos geométricos de forma hierárquica, onde cada nodo representa o menor retângulo que envolve todos os seus filhos (MBR - minimum bounding rectangle). Além disso, os nodos intermediários possuem ponteiros para os seus filhos, ao passo que os nodos folha apontam para os objetos armazenados na árvore. Para ilustrar estes conceitos, a Figura 5 (b) mostra uma possível R-tree de grau três que armazena os layouts das células da Figura 5 (a). Observe que as dimensões das células são armazenadas nos nodos folha, enquanto que o nodo intermediário (neste exemplo, apenas o nodo raiz) armazena os MBRs de seus filhos, assim como ponteiros para os mesmos. A Figura 5 (c) mostra estes MBRs em um plano 2D.

Por armazenar informações geométricas em sua estrutura, a R-tree permite realizar operações de busca por intersecções (chamadas de buscas espaciais) de forma mais rápida. No entanto, a organização dos objetos na árvore depende de como os mesmos são inseridos/removidos. Portanto, as próximas seções apresentam os algoritmos de inserção, busca e remoção da R-tree. Para a análise de complexidade dos algoritmos apresentados nas Subseções 4.2.1 a 4.2.3, assume-se que $n$ representa o número de elementos em uma R-tree. A altura máxima de uma R-tree, por sua vez, é logarítmica com relação ao número de objetos armazenados ($\log n$).

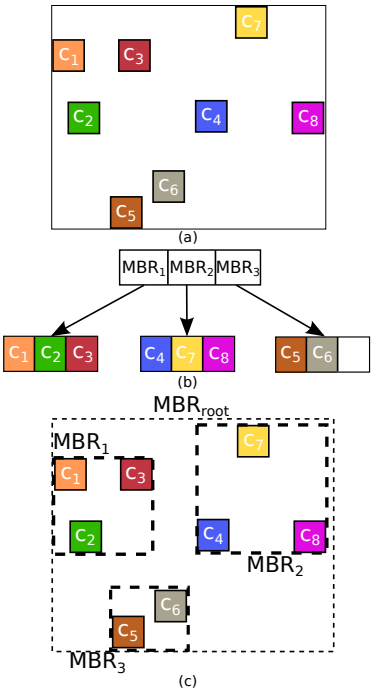


Figura 5 – (a) Exemplo de células em um circuito (quadrados coloridos). (b) Uma R-tree possível, gerada a partir dos dados de (a). (c) Representação geométrica da R-tree em (b).

A fim de facilitar a compreensão dos algoritmos apresentados nas Sub-seções 4.2.1 a 4.2.3, a Tabela 2 apresenta a terminologia utilizada. Além disso, a função $intersects(o_i, o_j)$, que aparece em tais algoritmos, retorna um valor *true* se existe intersecção entre dois objetos geométricos ou *false*, caso contrário.

### 4.2.1 Operação de inserção da R-tree

O Algoritmo 1 descreve a operação de inserção em uma R-tree. O algoritmo inicia inserindo $o_i$ na folha $n_i$ que resulta em menor aumento de área do MBR (linhas 1-2). Se após a inserção, o número de objetos armazenados em $n_i$ ultrapassar $M$, é necessário dividi-lo em novos nodos, de forma que nenhum nodo ultrapasse o número máximo

| Símbolo | Significado |
|---|---|
| $n_i$ | Nodo de uma R-tree |
| $o_i$ | Objeto geométrico armazenado (ou a ser armazenado) em uma R-tree |
| $MBR_i$ | MBR de um nodo $n_i$ |
| $leaves(R)$ | Conjunto de nodos folha de uma R-tree $R$ |
| $O(n_i)$ | Conjunto de objetos geométricos armazenados em um nodo $n_i$ |
| $size(n_i)$ | Número de elementos em um nodo $n_i$ (filhos ou objetos geométricos) |
| $m$ | Número mínimo de elementos em um nodo |
| $M$ | Número máximo de elementos em um nodo |
| $parent(n_i)$ | Nodo pai de um nodo $n_i$ |
| $children(n_i)$ | Conjunto de nodos filhos de um nodo $n_i$ |
| $root(R)$ | Raiz de uma R-tree $R$ |
| $d(o_i, o_j)$ | Distância entre dois objetos geométricos $o_i$ e $o_j$ |

Tabela 2 – Terminologia utilizada pelos algoritmos da R-tree

de objetos armazenados (linhas 3-4). Se $n_i$ é a raiz da árvore, então uma nova raiz é criada, e os novos nodos são adicionados como filhos dela (linhas 5-8). Caso contrário, o nodo $n_i$ é substituído pelos novos nodos (linhas 10-12).

**Algorithm 1:** R-TREE_INSERTION

```
   Input  : R-tree R e objeto geométrico o_i a ser inserido
   Output: R-tree R com objeto inserido
 1 n_i ← nodo n_i ∈ leaves(R) que resulta em menor aumento
     de área do MBR para inserir o_i;
 2 O(n_i) ← O(n_i) ∪ {o_i};
 3 if size(n_i) > M then
 4     (n_j, n_k) ← SPLIT_NODE(n_i);
 5     if n_i = root(R) then
 6         cria novo nodo r sem elementos;
 7         children(r) ← {n_j, n_k};
 8         root(R) ← r;
 9     else
10         p_i ← parent(n_i);
11         children(p_i) ← children(p_i) − {n_i};
12         children(p_i) ← children(p_i) ∪ {n_j, n_k};
13     end
14 atualiza MBRs dos nodos no caminho de root(R) até n_i,
     dividindo nodos quando necessário;
15 return R;
```

É importante observar que, após inserir $o_i$ na árvore, é necessário atualizar os MBRs de todos os nodos no caminho da raiz $root(R)$ até $n_i$, dado que estes podem ter sido modificados. Além disso, quando o nodo $n_i$ é dividido, o número de filhos do seu nodo pai muda. Portanto, se necessário, os nodos no caminho percorrido também devem ser divididos.

Existem diversos algoritmos utilizados para dividir um nodo da R-tree, conforme indicado na linha 4 do Algoritmo 1. O Algoritmo 2 descreve uma abordagem comumente usada, chamada de divisão linear (*linear split*). O algoritmo inicia criando dois novos nodos vazios $n_j$ e $n_k$ (linha 1). Em seguida, identifica o par de objetos $o_i$ e $o_j$ mais distantes entre si, os remove de $n_i$ e os armazena em $n_j$ e $n_k$, respectivamente (linhas 2-5). Os outros objetos de $n_i$ são inseridos em ordem, sendo que cada objeto é inserido no nodo (entre $n_j$ e $n_k$) que resulta em menor aumento de área de seu MBR (linhas 6-9).

**Algorithm 2:** SPLIT_NODE

**Input :** Nodo $n_i$ a ser dividido
**Output:** Novos nodos $n_j$ e $n_k$

1 cria novos nodos $n_j$ e $n_k$ sem elementos;
2 $(o_i, o_j) \leftarrow$ par de objetos $o_i, o_j \in O(n)$ com maior distância $d(o_i, o_j)$;
3 $O(n_j) \leftarrow \{o_i\}$;
4 $O(n_k) \leftarrow \{o_j\}$;
5 $O(n_i) \leftarrow O(n_i) - \{o_i, o_j\}$;
6 **foreach** $o_i \in O(n)$ **do**
7 $\quad n_l \leftarrow$ nodo $n_l \in \{n_j, n_k\}$ que resulta em menor aumento de área do MBR para inserir $o_i$;
8 $\quad O(n_l) \leftarrow O(n_l) \cup \{o_i\}$;
9 **end**
10 **return** $n_j, n_k$

A Figura 6 ilustra como a R-tree da Figura 5 (b) é construída através da inserção dos layouts das células da Figura 5 (a). A ordem de inserção dos objetos depende da aplicação. No exemplo da Figura 6, assumiu-se que os layouts das células são inseridos na ordem de $c_1$ a $c_8$. Cada subfigura apresenta a configuração da árvore (e a representação dos seus nodos no plano 2D) após cada inserção. Inicialmente, a árvore está vazia, portanto os três primeiros objetos são inseridos na própria raiz (Figuras 6 (a), (b) e (c)). Como a R-tree deste exemplo possui

grau 3, ao inserir o layout da célula $c_4$, é necessário dividir o nodo raiz.

Utilizando o Algoritmo 2, os layouts de célula $c_1$ e $c_4$ são armazenados em nodos diferentes pois são o par de objetos mais distantes entre si nesta etapa (linhas 2-5). Em seguida, as células $c_2$ e $c_3$ são inseridas no primeiro nodo pois este resulta em menor aumento de área do MBR (linhas 6-9). Por fim, um novo nodo raiz é criado, onde dois dos seus filhos apontam para os novos nodos criados (Figura 6 (d)).

Os layouts das células $c_5$ e $c_6$ são inseridas na folha mais à direita, pois esta resulta em menor aumento da área de seu MBR (Figuras 6 (e) e (f)). Por fim, os layouts das células $c_7$ e $c_8$ são inseridas na segunda folha (Figuras 6 (g) e (h)).

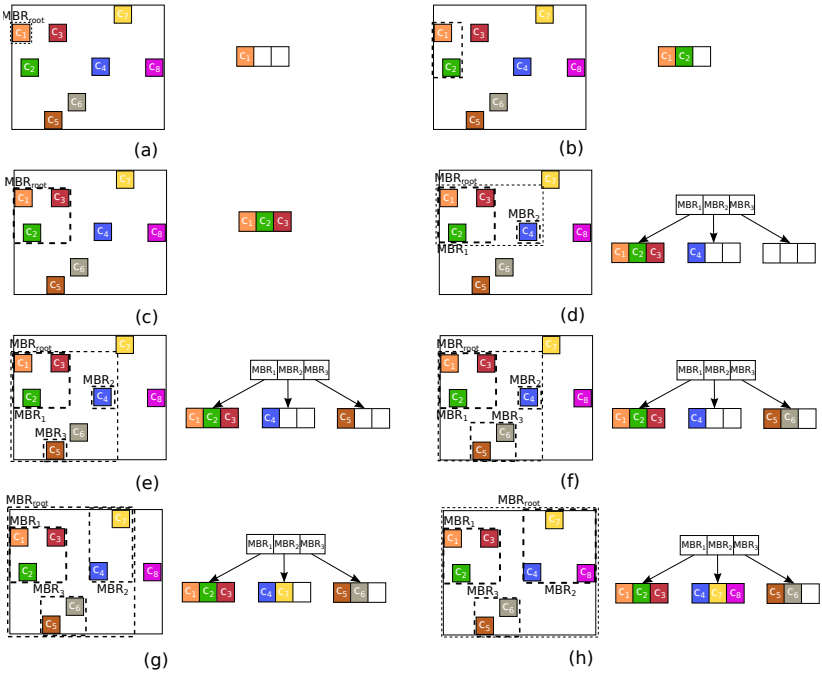


Figura 6 – Exemplo de construção de uma R-tree de grau 3 inserindo layouts de células. Cada subfigura apresenta a estrutura da árvore após cada inserção.

Para avaliar a complexidade da operação de inserção é necessário analisar as linhas 1 e 4 do Algoritmo 1. A linha 1 percorre a árvore em profundidade escolhendo sempre o nodo que resulta em menor aumento de área do MBR. Logo, o tempo de execução deste passo é limitado pela

altura da árvore, resultando em uma complexidade $\mathcal{O}(\log n)$. A linha 4 por sua vez, faz uso do Algoritmo 2, cuja complexidade é proporcional ao número de elementos em um nodo. Como o número de elementos por nodo é constante (limitado pelo grau da árvore), esse passo possui complexidade $\mathcal{O}(1)$. Logo, a complexidade da operação de inserção é dada pela linha 1 e é, portanto, $\mathcal{O}(\log n)$.

### 4.2.2 Operação de busca na R-tree

O Algoritmo 3 descreve a operação de busca em uma R-tree, o qual será utilizado para identificar sobreposições de células durante a legalização incremental. O algoritmo recebe como entrada uma R-tree $R$, um nodo $n_i$, e realiza a busca de um objeto geométrico $o_i$ na subárvore que possui $n_i$ como raiz. Esta operação é realizada de forma recursiva, tendo como condição de parada o momento em que um nodo folha é atingido (linha 1). Nesse caso, o algoritmo retorna os elementos armazenados em $n_i$ que possuam intersecção com $o_i$ (linha 2). Se $n_i$ não for um nodo folha, o algoritmo é chamado recursivamente para todos os filhos de $n_i$ cujo MBR possui intersecção com $o_i$ (linhas 4-9).

**Algorithm 3:** R-TREE_SEARCH

**Input :** R-tree $R$, nodo $n_i$ e objeto geométrico $o_i$ a ser buscado
**Output:** Nodos intersectando com $o_i$

```
1  if n_i ∈ leaves(R) then
2  |  return n_i;
3  else
4  |  N ← {n_j ∈ children(n_i) | intersects(o_i, MBR_j)};
5  |  I ← {};
6  |  foreach n_j ∈ N do
7  |  |  I ← I ∪ R-TREE_SEARCH(R, n_j, o_i);
8  |  end
9  |  return I;
10 end
```

Observe que, no pior caso o algoritmo visita todos os nodos da árvore e possui complexidade $\mathcal{O}(n)$. No entanto, a complexidade de pior caso raramente é atingida, dado que na prática as regiões buscadas não possuem intersecção com todos os nodos da árvore.

A Figura 7 ilustra, através de exemplo, como na prática não é

necessário visitar todos os nodos de uma R-tree durante uma operação de busca. Esta figura mostra o processo de busca dos nodos que intersectam uma região (retângulo tracejado em verde). Cada subfigura corresponde a uma chamada do Algoritmo 3 para uma determinada subárvore. A Figura 7 (a) ilustra a chamada do algoritmo de busca a partir da raiz da R-tree. Nesse caso, o Algoritmo 3 executa as linhas 4-9, encontrando os nodos cujos MBRs intersectam com a região de busca (neste caso $MBR_2$ e $MBR_3$). As Figuras 7 (b) e (c), por sua, vez, mostram o algoritmo de busca a partir dos nodos encontrados no passo anterior. Como estes nodos são folhas, o algoritmo executa a linha 2 e retorna os objetos que intersectam com a região de busca (neste caso $c_4$ e $c_6$).

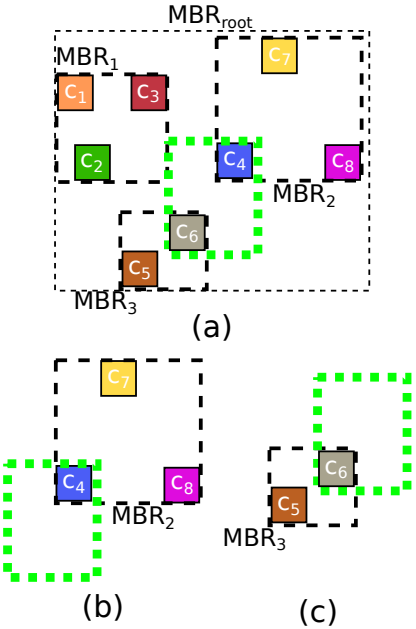


Figura 7 – Exemplo de busca espacial na R-tree da Figura 5. (a) Operação de busca a partir da raiz da árvore. (b) Operação de busca a partir do nodo $n_2$. (c) Operação de busca a partir do nodo $n_3$

Note que o nodo representado pelo $MBR_1$ não foi visitado, dado que não possui intersecção com a região de busca. Desta forma, a R-tree age como um mecanismo de filtro, que evita a busca em nodos que garantidamente não possuem intersecção com a região de busca.

Esse mecanismo de filtro possibilita que a R-tree realize consultas espaciais rápidas (por exemplo, para identificar sobreposições de células). Quanto menor for a intersecção entre MBRs, mais eficientes são as buscas, pois diminui a probabilidade de uma região de busca intersectar com múltiplos nodos. Algoritmos eficientes podem ser usados para melhor organizar os dados na árvore inserindo todos os objetos de uma só vez, o que melhora o tempo de execução das buscas espaciais (MANOLOPOULOS et al., 2010).

### 4.2.3 Operação de remoção da R-tree

O Algoritmo 4 descreve a operação de remoção de um objeto geométrico $o_i$ de uma R-tree. No contexto de legalização incremental, esta operação é utilizada para atualizar a estrutura de dados quando uma célula é movida.

**Algorithm 4:** R-TREE_DELETION

```
   Input  : R-tree R, e objeto geométrico o_i a ser removido
   Output: R-tree R com objeto removido
 1 N ← R-TREE_SEARCH(R, root(R), o_i);
 2 foreach n_i ∈ N do
 3 |  if o_i ∈ O(n_i) then
 4 |  |  O(n_i) ← O(n_i) − {o_i};
 5 |  |  if size(n_i) < m then
 6 |  |  |  p_i ← parent(n_i);
 7 |  |  |  children(p_i) ← children(p_i) − {n_i};
 8 |  |  |  foreach o_j ∈ O(n_i) do
 9 |  |  |  |  R-TREE_INSERTION(R, o_j);
10 |  |  |  end
11 |  |  atualiza MBRs dos nodos no caminho de root(R) até
   |  |   n_i;
12 end
13 return R;
```

Inicialmente, o algoritmo busca o conjunto de nodos $N$ que possuem intersecção com o objeto $o_i$ usando o Algoritmo 3 (linha 1). Note que mais de um nodo pode ser encontrado nesta busca, já que pode haver intersecção entre MBRs de nodos diferentes. Em seguida, o algoritmo verifica quais nodos realmente possuem o objeto $o_i$, e então o

remove dos mesmos (linha 4). Se após a remoção o nodo possuir menos elementos do que o mínimo $m$, ele é removido da árvore e seus elementos são reinseridos (linhas 5-10). Após cada remoção, o algoritmo deve atualizar os MBRs de todos os nodos no caminho de $root(R)$ até o nodo modificado (linha 11).

A complexidade da operação de remoção é dominada pelas operações de busca e inserção necessárias. Como a região de busca corresponde a um objeto da árvore (e portanto, possui intersecção com uma pequena parte da árvore), a operação de busca possui complexidade logarítmica, em média. A operação de inserção, por sua vez, possui complexidade $\mathcal{O}(\log n)$. No entanto, a inserção é realizada apenas para reinserir os objetos de um nodo removido. Assumindo uma R-tree sem objetos repetidos, a operação de inserção é realizada no máximo $m-1$ vezes. Portanto, a complexidade da operação de remoção é dominada pela operação de busca, sendo logarítmica, em média.

## 4.3 ALGORITMO DE LEGALIZAÇÃO INCREMENTAL UTILIZANDO R-TREE

A capacidade de uma R-tree de realizar buscas espaciais rápidas é particularmente útil para identificar sobreposições de células durante um algoritmo de legalização. No entanto, apenas o trabalho de Kennings, Darav e Behjat (2014) propõe o uso desta estrutura de dados, e apenas para determinar intersecções com macro blocos. Para utilizar esta estrutura de dados de forma eficaz, é necessário armazenar os layouts de todas as células do circuito em uma ou mais R-trees. Portanto, este trabalho propõe uma adaptação do algoritmo de legalização de Chow et al. (2014) para fazer uso da R-tree. A estratégia utilizada neste trabalho consiste em dividir o circuito em um conjunto de sublinhas $\Sigma = \{\sigma_1, \sigma_2, ..., \sigma_n\}$ de forma que nenhuma sublinha intersecte macro blocos. Assim, como todas as células devem pertencer a uma sublinha, é possível garantir que não exista intersecção entre células e macro blocos. A partir do conjunto de sublinhas, uma R-tree $R_i$ é criada para cada sublinha $\sigma_i$. Uma R-tree adicional $R_\Sigma$ é criada com os limites de cada sublinha, e permite identificar a qual sublinha uma determinada região pertence. Note que as células do circuito poderiam ser armazenadas de forma diferente utilizando R-trees. Por exemplo, todas as células poderiam ser armazenadas em uma única árvore. Entretanto, optou-se por construir uma R-tree para cada linha de forma a reduzir o número de elementos em cada árvore e, consequentemente,

reduzir o tempo de execução de suas operações.

A Figura 8 apresenta um exemplo de conjunto de R-trees criadas para o posicionamento apresentado no exemplo da Figura 4 (a). Observe que, devido ao macro bloco $c_5$, quatro linhas do circuito precisam ser divididas, resultando num total de 11 sublinhas para este exemplo ($R_{\sigma_1}$ a $R_{\sigma_{11}}$).

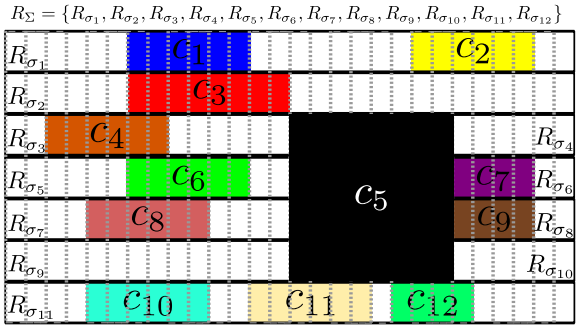


Figura 8 – Exemplo de sublinhas para parte de um circuito, apontando suas respectivas R-trees.

O Algoritmo 5 apresenta os passos da técnica proposta. Para simplificar a notação, o Algoritmo 5 assume que cada sublinha possui uma largura $w(\sigma_j)$ e estende a notação da seção 4.2 para representar o conjunto de elementos de uma R-tree $R$ como $O(R)$. Dado o conjunto de células de uma sublinha $O(R_{\sigma_i})$, a capacidade dessa linha é calculada pela Equação (4.1) como sendo o espaço livre na mesma.

$$cap(\sigma_j) = w(\sigma_j) - \sum_{c_i \in O(R_{\sigma_j})} w(c_i) \tag{4.1}$$

Dado o conjunto de R-trees e uma posição alvo $(x'(c_i), y'(c_i))$, o algoritmo de legalização executa os seguintes passos: primeiro, usa $R_\Sigma$ para encontrar a sublinha $\sigma_i$ contendo a posição alvo (linhas 1-2). Observe que a variável $o_i$ representa um objeto geométrico na posição alvo $(x'(c_i), y'(c_i))$ e com as dimensões de $c_i$. Assim, é possível identificar a sublinha $\sigma_i$ a qual esta posição pertence buscando em $R_\Sigma$ por uma R-tree cujo MBR da raiz intersecte com $o_i$. Se a capacidade da sublinha alvo $\sigma_i$ é menor do que a largura de $c_i$, não é possível posicionar $c_i$ em $(x'(c_i), y'(c_i))$ e nenhuma célula é movida (linhas 3-4). Caso contrário, o algoritmo procede com a legalização utilizando $R_{\sigma_i}$ para identificar sobreposições de células com a posição alvo de $c_i$, e armazenando-as

**Algorithm 5:** R-TREE-BASED_INCREMENTAL_LEGALIZATION

**Input :** R-tree de sublinhas $R_\Sigma$, R-tree de células $R_{\sigma_i}$ para cada sublinha $\sigma_i$, e posição alvo $(x'(c_i), y'(c_i))$ para uma célula $c_i \in \mathcal{C}$

**Output:** *true* se é possível posicionar $c_i$, $false$ caso contrário

1 $o_i \leftarrow (w(c_i), h(c_i), x'(c_i), y'(c_i))$;
2 $R_{\sigma_i} \leftarrow \{R_{\sigma_i} \in O(R_\Sigma) \mid intersects(MBR_{root(R_{\sigma_i})}, o_i)\}$;
3 **if** $cap(\sigma_i) < w(c_i)$ **then**
4 | **return** $false$;
5 **else**
6 | $\mathcal{O} \leftarrow \{(c_i, c_j) \mid c_j \in O(R_{\sigma_i}) \wedge intersects(L(c_j), o_i)\}$;
7 | $M \leftarrow \{(c_i, (x'(c_i), y'(c_i)))\}$;
8 | **while** $\mathcal{O} \neq \emptyset$ **do**
9 | | $(c_i, c_j) \leftarrow$ par de células com sobreposição em $O$;
10 | | $\mathcal{O} \leftarrow \mathcal{O} - \{(c_i, c_j)\}$;
11 | | **if** $(x(c_j) + w(c_j)/2) \leq (x'(c_i) + w(c_i)/2)$ **then**
12 | | | $(x'(c_j), y'(c_j)) \leftarrow (x'(c_i) - w(c_j), y(c_j))$;
13 | | **else**
14 | | | $(x'(c_j), y'(c_j)) \leftarrow (x'(c_i) + w(c_i), y(c_j))$;
15 | | **end**
16 | | $o_j \leftarrow (w(c_j), h(c_j), x'(c_j), y'(c_j))$;
17 | | $\mathcal{O} \leftarrow \mathcal{O} \cup \{(c_j, c_k) \mid c_k \in O(R_{\sigma_i}) \wedge intersects(L(c_k), o_j)\}$;
18 | | $M \leftarrow M \cup \{(c_j, (x'(c_j), y'(c_j)))\}$;
19 | **end**
20 | **foreach** $(c_j, (x'(c_j), y'(c_j))) \in M$ **do**
21 | | posiciona $c_j$ em $(x'(c_j), y'(c(c_j)))$ e atualiza $R_{\sigma_i}$;
22 | **end**
23 | **return** $true$;
24 **end**

em um conjunto $\mathcal{O}$ (linha 6). Note que as sobreposições são identificadas buscando por layouts de células $L(c_j)$ que intersectem com $o_i$. Após identificar as sobreposições, o algoritmo as remove do conjunto $\mathcal{O}$ (linhas 9-10) e, para cada par de células $(c_j, c_i)$ com sobreposição, ele identifica o deslocamento necessário para remover esta sobreposição (linhas 11-15). Note que estes deslocamentos são feitos apenas para a esquerda ou direita, nunca para cima nem para baixo, uma vez que

estão restritos à sublinha.

Após remover cada sobreposição, o algoritmo busca na R-tree por novas sobreposições que possam ter resultado do deslocamento realizado (linhas 16-17). Quando todas as sobreposições tiverem sido removidas, o algoritmo itera por todos os movimentos (salvos nas linhas 7 e 18) aplicando-os e atualizando a R-tree $R_{\sigma_i}$ (linhas 20-22). Note que todos os movimentos são aplicados somente se a legalização ocorrer com sucesso. Caso contrário nenhuma célula é movida. Desta forma, é possível adicionar restrições adicionais para a legalização (por exemplo, deslocamento máximo de células). Caso alguma destas restrições seja violada durante a legalização, nenhuma célula é movida. Estas restrições são difíceis de serem verificadas antes da legalização. Portanto, aplicar os movimentos apenas ao final do algoritmo permite que elas sejam sempre respeitadas de forma fácil.

Observe que as linhas 6 e 17 do Algoritmo 5 são realizadas através de buscas espaciais na R-tree $R_{\sigma_i}$, enquanto que a linha 21 atualiza a R-tree removendo da mesma os layouts das células movidas antes da legalização e inserindo-os novamente após o movimento. Com exceção destas linhas que executam operações na R-tree, todas as outras operações são realizadas em tempo constante. Portanto, o tempo de execução da técnica proposta é dominado pela complexidade das operações da R-tree. Como apresentado na Seção 4.2, as operações da R-tree possuem complexidade logarítmica, em média. Dado que no pior caso todas as células em uma sublinha precisam ser movidas em uma legalização, a complexidade do algoritmo proposto é $\mathcal{O}(n \log(n))$, onde $n$ representa o número de células na linha que está sendo legalizada (que corresponde ao número de elementos na R-tree).

Por outro lado, para realizar as mesmas operações do Algoritmo 5 utilizando uma lista encadeada, é necessário realizar uma busca linear na lista e as operações seguintes podem ser feitas em tempo constante, resultando em uma complexidade assintótica $\mathcal{O}(n)$. Apesar desta complexidade ser menor do que a obtida pelo uso da R-tree, na prática um pequeno número das células de uma linha é movido a cada legalização[1].

Para verificar na prática o tempo de execução atingido pelo uso de cada estrutura de dados, a Figura 9 apresenta o tempo de execução médio para 10 realizações da legalização incremental em função do número de movimentos durante a legalização (utilizando o Algoritmo 5) para ambas as estruturas de dados. O número de movimentos assumido neste experimento (entre 1 e 50) foi escolhido baseado na média e

[1]Para os *benchmarks* utilizados neste trabalho, o número médio de movimentos a cada legalização foi em torno de 17

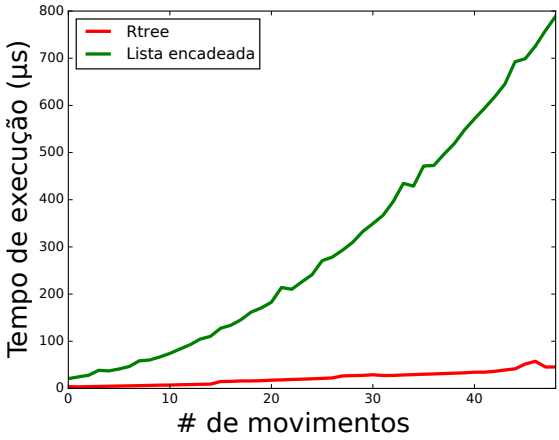


Figura 9 – Tempo de execução em função do número de movimentos durante a legalização incremental. Cada curva representa o tempo de execução usando uma determinada estrutura de dados (R-tree ou lista encadeada).

desvio padrão do número de movimentos observado para os *benchmarks* utilizados neste trabalho. Estes *benchmarks* correspondem aos circuitos disponibilizados pela infraestrutura da competição *ICCAD 2015 CAD Contest (problem C: Incremental Timing-Driven Placement)*, apresentados em Kim et al. (2015), e que serão melhor detalhados na Seção 5.1. Observe que, apesar da maior complexidade teórica da legalização incremental utilizando R-tree, na prática, o uso desta estrutura de dados resulta em um menor tempo de execução. Tal comportamento é consequência do pequeno número de movimentos por legalização, que faz com que o tempo de busca da lista encadeada se torne o gargalo do algoritmo de legalização incremental, quando utilizando esta estrutura de dados.

# 5 RESULTADOS EXPERIMENTAIS

Este capítulo apresenta os resultados experimentais obtidos por este trabalho. Inicialmente, ele descreve a infraestrutura experimental utilizada. Em seguida, analisa as três estratégias de legalização, no contexto de uma técnica de otimização incremental. Por fim, apresenta os resultados experimentais da avaliação da técnica proposta para legalização incremental.

## 5.1 INFRAESTRUTURA EXPERIMENTAL

Os experimentos realizados utilizaram o conjunto de benchmarks disponibilizados pela competição *ICCAD 2015 CAD Contest (problem C: Incremental Timing-Driven Placement)* (KIM et al., 2015), o qual inclui 8 circuitos que possuem entre 768k e 1,93M células, todos derivados de circuitos industriais. Para cada circuito, a infraestrutura da competição fornece um posicionamento inicial, o qual já está legalizado. Optou-se por utilizar tal infraestrutura pois a mesma disponibiliza circuitos com número de células compatível com circuitos contemporâneos. Além disso, tal infraestrutura é de acesso aberto, o que facilita a comparação experimental deste trabalho com futuros trabalhos que possivelmente serão realizados por terceiros.

Todos os algoritmos apresentados neste capítulo foram implementados em C++. Para o desenvolvimento do protótipo com a técnica proposta, utilizou-se a implementação de R-tree disponível na biblioteca Boost (BOOST, 2015). Para fazer uso da R-tree é necessário definir dois de seus parâmetros: o grau da árvore e o algoritmo utilizado para a divisão de nodos durante a operação de inserção. Após alguns experimentos iniciais, não foi observada uma diferença significativa no tempo de execução das operações da R-tree ao variar o seu grau. Devido a isso, optou-se por utilizar uma árvore com grau 16. Por outro lado, o algoritmo de divisão de nodos utilizado tem impacto direto no tempo de execução da operação de inserção, assim como no tempo de execução das operações de busca subsequentes. Portanto, optou-se por utilizar o algoritmo denominado R*, que resulta em buscas espaciais mais rápidas, ao custo de um tempo de inserção um pouco maior.

Todos os experimentos foram realizados em um computador Linux com quatro CPUs Intel® Core® i5-4460 @ 3,20 GHz e 32GB RAM. Os experimentos que avaliam o tempo de execução podem apresentar

resultados diferentes em cada realização, devido a variações causadas pelo computador utilizado. Portanto, para aumentar a confiança estatística obtida pelos experimentos, os mesmos foram repetidos 10 vezes, o que resultou em 99% de confiança estatística[1]. Como a confiança estatística obtida foi suficientemente alta, os experimentos não foram repetidos mais vezes.

## 5.2 COMPARAÇÃO DAS ESTRATÉGIAS DE LEGALIZAÇÃO

Esta seção compara as três estratégias de legalização usadas na otimização incremental, as quais foram apresentadas no Capítulo 2: legalização final, legalização iterativa e legalização incremental. A técnica proposta no presente trabalho foi escolhida como representante da estratégia de legalização incremental. Já as outras duas estratégias foram prototipadas usando o algoritmo Abacus, implementado a partir do pseudocódigo disponível em Spindler, Schlichtmann e Johannes (2008). Abacus foi escolhido como algoritmo de legalização completa (a ser usado nas estratégias final e iterativa) por ser utilizado em diversos trabalhos (com pequenas modificações, quando necessário), tais como os trabalhos de Popovych et al. (2014), Kennings, Darav e Behjat (2014), Li e Koh (2014) e Huang et al. (2015). Como Abacus é um algoritmo bastante usado, outros trabalhos presentes na literatura comparam seus resultados com Abacus, tais como os trabalhos de Puget et al. (2015) e Ho e Liu (2010). Desta forma, é possível comparar, de forma indireta, os resultados deste trabalho com os resultados dos outros trabalhos. Uma comparação das estratégias de legalização também é apresentada no trabalho Netto et al. (2016a). Porém, aquele trabalho utiliza a técnica de Popovych et al. (2014) como representante da legalização incremental. Esta Seção, por outro lado, utiliza a técnica proposta neste trabalho de mestrado como representante da estratégia de legalização incremental.

Para que as três estratégias pudessem ser avaliadas em um fluxo de otimização real, elas foram integradas no algoritmo de posicionamento incremental guiado por atraso de Guth et al. (2015), resultando em três versões de tal técnica de otimização: 1) uma versão que utiliza Abacus para legalizar todo o circuito somente no final da otimização (denotado por FIN - *Final Legalization* ); 2) uma versão que usa Abacus para legalizar o circuito inteiro após cada iteração da técnica de otimização (chamada de ITE - *Iterative Legalization* ); 3) uma versão

[1] A confiança estatística foi medida utilizando o teste t de Student para p=0,01.

que utiliza a técnica proposta de legalização incremental para legalizar o circuito após cada movimento (abreviada por INC - *Incremental Legalization* ). A Figura 10 mostra os fluxogramas das três versões da técnica de otimização incremental. Observe que os fluxogramas correspondem aos da Figura 3, apenas substituindo o nome do passo de legalização pelo nome da técnica utilizada.

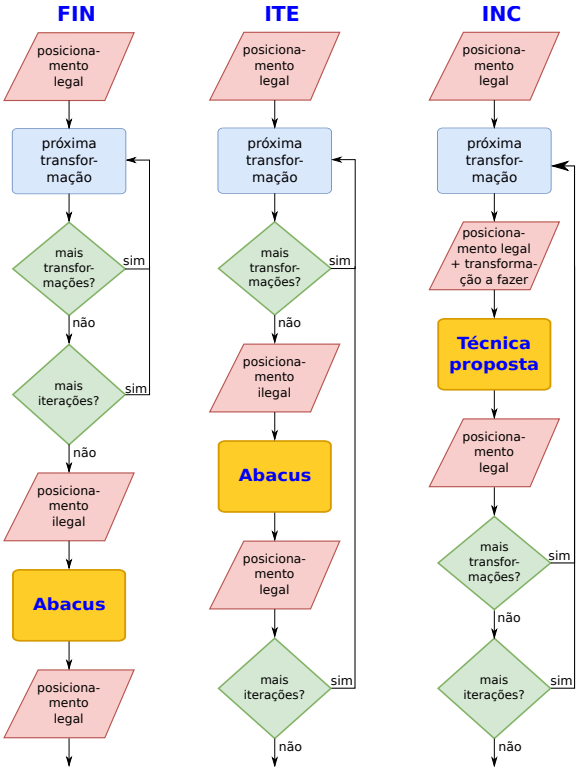


Figura 10 – Fluxogramas das três versões da técnica de otimização incremental avaliada.

O algoritmo de posicionamento incremental guiado por atraso de Guth et al. (2015) recebe como entrada o número de saídas primárias[2] (PO - *primary output*) com violações de atraso a serem otimizadas. Para cada PO a ser otimizada, o algoritmo percorre seu caminho crí-

[2]Uma saída primária é um pino de saída do circuito ou um pino de entrada de um *flip flop*

tico em direção à entrada primária do referido caminho[3] (PI - *primary input*), enquanto move células com violações de atraso, a fim de reduzir a carga capacitiva de suas conexões. Neste caso, o número de células movidas durante a otimização é diretamente proporcional ao número de POs selecionadas para serem otimizadas. Devido a isto, para avaliar o comportamento das estratégias de legalização em função do número de células movidas, foram realizados experimentos variando-se o número de POs a serem otimizadas. Os valores utilizados foram 200, 500, 1000 e 2000 POs. Além disso, o algoritmo de otimização foi executado por 5 iterações, como proposto em Guth et al. (2015).

Para avaliar o tempo de execução foi medido o tempo necessário para executar todo o algoritmo de posicionamento incremental guiado por atraso, de forma a englobar ambos os tempos de otimização e legalização. A qualidade da solução obtida utilizando cada estratégia de legalização foi avaliada utilizando três métricas diferentes: violações de atraso[4], densidade do circuito e comprimento das interconexões. Os resultados obtidos pelas estratégias ITE e INC, em termos de tempo de execução e métricas de qualidade, foram normalizados com base nos resultados obtidos pela estratégia FIN. Note que, apesar de não ter sido medida a potência dos circuitos após a otimização, a variação do comprimento das interconexões pode ser utilizada como um indicador da variação da potência do mesmo. Isto ocorre pois a técnica de otimização incremental utilizada apenas move células e, portanto, o único impacto na potência resultante desta otimização advém da redução do comprimento das interconexões do mesmo.

As Figuras 11 (a) e (b) mostram os tempos de execução para as estratégias INC e ITE, normalizados em relação aos tempos de execução da estratégia FIN. A estratégia INC atingiu os menores tempos de execução para todos os circuitos, sendo de 26% a 73% mais rápida que a estratégia FIN. Por outro lado, ITE foi de 2.16 a 3.69 vezes mais lento que FIN, dado que a estratégia iterativa precisa legalizar todo o circuito cinco vezes. Observe que o tempo de execução de INC depende do número de células movidas, enquanto que o mesmo não ocorre com as outras duas estratégias, que sempre legalizam o circuito inteiro. Desta forma, o tempo de execução normalizado de INC cresce à medida que o número de POs a serem otimizadas aumenta, pois mais

[3]Uma entrada primária é um pino de entrada do circuito ou um pino de saída de um *flip flop*

[4]A violação de atraso em um ponto do circuito é medida como a diferença entre o tempo em que o sinal precisa estar estável para garantir o funcionamento do circuito, dentro do período especificado no projeto (chamado de *required time*), e o tempo em que o sinal efetivamente estabiliza neste ponto (chamado de *arrival time*

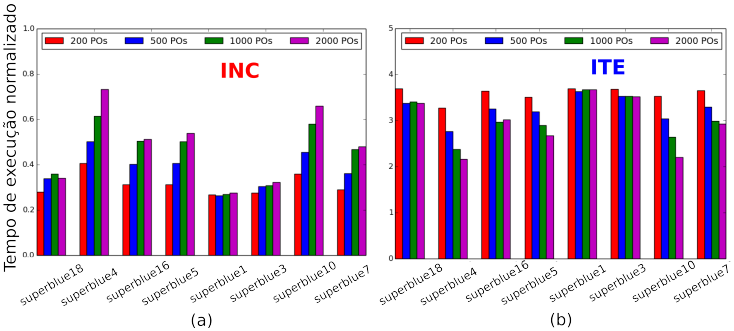


Figura 11 – Tempos de execução para otimizar 200, 500, 1000 e 2000 POs, normalizados em relação aos tempos de execução obtidos utilizando a estratégia FIN.

células são movidas e, consequentemente, legalizadas. Por outro lado, o tempo de execução normalizado de ITE diminui à medida que o número de POs otimizados aumenta, pois o tempo de legalização permanece constante enquanto que o tempo de otimização aumenta. No entanto, para os circuitos superblue18, superblue1 e superblue3, o tempo de execução normalizado permanece aproximadamente constante mesmo com o aumento do número de POs. Isto ocorre porque estes circuitos não possuem um número de POs com violações de atraso grande o bastante. Portanto, aumentar o número de POs a serem otimizadas não aumenta o número de células movidas.

Para avaliar as violações de atraso ao final da otimização, foi medido o *Total Negative Slack* (TNS), calculado como a soma das violações de atraso de todas as POs. As Figuras 12 (a) e (b) apresentam os valores de TNS normalizados para as estratégias INC e ITE. A estratégia INC levou ao maior TNS, resultando em até 15% mais violações de atraso quando comparada à FIN (2,5% em média). O motivo pelo qual INC resultou em mais violações de atraso será explicado com a ajuda do exemplo da Figura 13.

Na Figura 13 (a), a célula $c_1$ é movida para a posição mostrada pelo retângulo tracejado, que intersecta com um macro bloco (retângulo grande em preto) em uma região congestionada de células. Dado que as linhas adjacentes à posição alvo estão congestionadas, a estratégia de legalização incremental não é capaz de legalizar este movimento e, portanto, $c_1$ permanece em sua posição original, como mostra a Figura 13 (b). Por outro lado, como a estratégia FIN legaliza o circuito somente quando todos os movimentos foram realizados, outras células na mesma

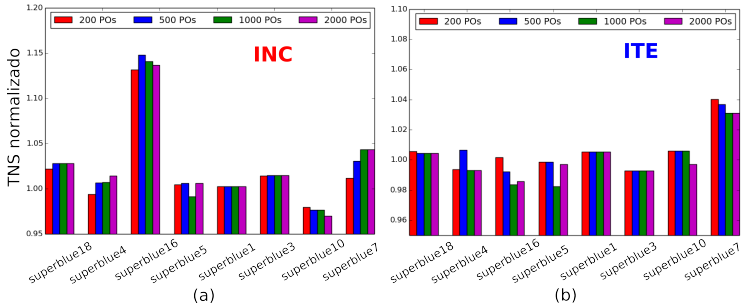


Figura 12 – Variação das violações de atraso resultantes do uso das estratégias INC e ITE, quando comparadas com a estratégia FIN. Todas as violações de atraso foram medidas usando a métrica TNS.

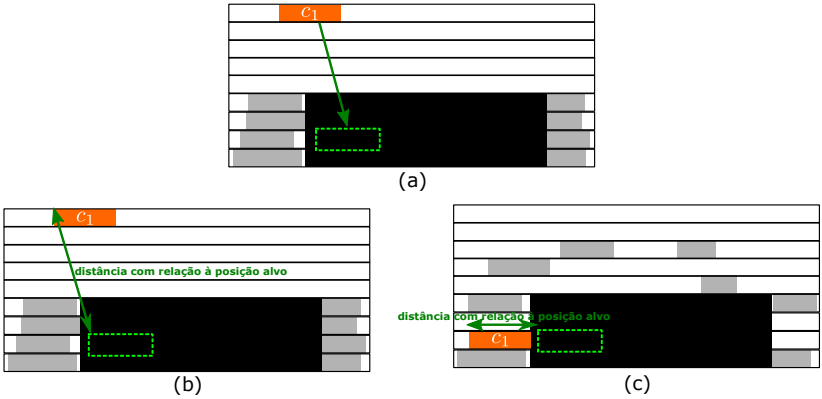


Figura 13 – Exemplo de deslocamento de célula usando duas estratégias de legalização diferentes. A célula $c_1$ (retângulo laranja) possui uma posição alvo (retângulo tracejado em verde) situada sobre um macro bloco (retângulo grande em preto). (b) Legalização resultante usando INC. (c) Legalização resultante usando FIN. As flechas vermelhas mostram o deslocamento da célula.

linha podem ter sido movidas, permitindo que $c_1$ possa ser posicionada próxima a sua posição alvo, como mostra a Figura 13 (c). Como resultado, utilizando a estratégia INC a célula $c_1$ é posicionada longe de sua posição alvo encontrada pelo algoritmo de otimização, resultando em mais violações de atraso. O circuito superblue16 em particular, que resultou no maior TNS normalizado, possui várias células com viola-

ções de atraso situadas em regiões congestionadas com macro blocos, influenciando negativamente nos resultados obtidos por INC.

Dado que a estratégia ITE usa o mesmo algoritmo que FIN para legalizar o circuito, não há diferença significante de violações de atraso entre estas duas estratégias, com TNS normalizado variando entre uma redução de 2% e um aumento de 4%. Além disso, para ambas as estratégias INC e ITE, para a maioria dos circuitos, o TNS normalizado não varia muito quando se aumenta o número de POs a serem otimizadas. Tais resultados mostram que o número de violações de atraso não depende do número de células movidas, mas das características do circuito.

Para avaliar o perfil de densidade das soluções obtidas pelas diferentes estratégias, foi utilizada a métrica *Average Bin Utilization* (ABU). Para calcular esta métrica o circuito é dividido em *bins* de mesmo tamanho, sendo então medida a utilização média dos *bins* mais densos, conforme definido em (KIM et al., 2015). As Figuras 14 (a) e (b) mostram os resultados normalizados de ABU para as estratégias INC e ITE. Observe que não há diferença significativa entre os resultados de densidade obtidos pela maioria dos circuitos. Considerando INC, o circuito superblue5 resultou na maior diferença de ABU (redução de 14%), enquanto que os outros circuitos resultaram em uma diferença de no máximo 7%. Por outro lado, para a estratégia ITE, a maior redução de ABU foi de 5% para o circuito superblue16 com 1000 POs, e de até 3% para todos os outros circuitos. Além disso, o ABU normalizado permanece aproximadamente constante mesmo variando o número de POs para 5 dos 8 circuitos. Tais resultados mostram que a densidade do circuito não é afetada significativamente nem pela estratégia de legalização utilizada, nem pelo número de células movidas.

Utilizando o algoritmo FLUTE de Chu e Wong (2008) para estimar o comprimento das interconexões do circuito, observou-se que a diferença entre todas as três estratégias é de menos de 1% para todos os circuitos. Isto ocorre pois apenas um pequeno número de células é movido durante a otimização incremental. Para mostrar este comportamento, a Figura 15 mostra a porcentagem média de células movidas por cada uma das três versões do algoritmo de otimização (cada um com uma estratégia de legalização) quando o número de POs a serem otimizadas aumenta. Estas porcentagens incluem todas as células movidas, incluindo as movidas pelo algoritmo de legalização. Note que menos de 3% das células do circuito são movidas, mesmo quando 2000 POs são consideradas pela otimização. Estes resultados mostram por que não foi observada uma diferença significante do comprimento das

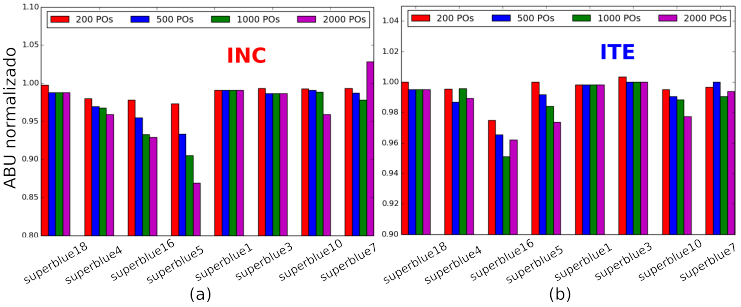


Figura 14 – Variação da densidade dos circuitos resultantes do uso das estratégias INC e ITE, quando comparadas com a estratégia FIN. Todas as densidades foram medidas usando a métrica ABU.

interconexões. Além disso, resultados semelhantes são esperados para outras técnicas de otimização incremental, dado que estas técnicas são projetadas para mover somente um pequeno subconjunto das células do circuito. O pequeno número de células movidas também mostra por que o tempo de execução da estratégia INC é sempre menor que das outras duas estratégias. Dado que menos de 3% das células são movidas, o custo de legalizar o circuito inteiro, como feito por FIN e ITE é muito maior do que o custo de realizar uma pequena legalização a cada movimento de uma célula.

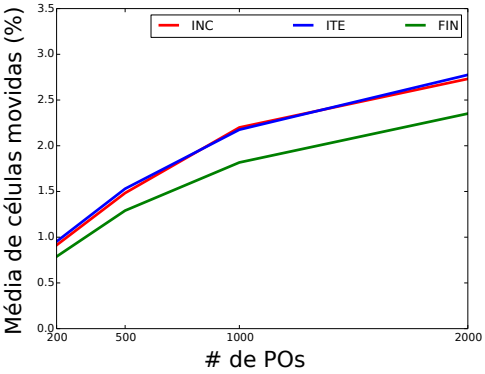


Figura 15 – Porcentagem média do número de células dos circuitos que são movidas quando se varia o número de POs a serem otimizadas.

## 5.3 AVALIAÇÃO DO ALGORITMO DE LEGALIZAÇÃO INCREMENTAL UTILIZANDO R-TREE

Esta seção compara a técnica de legalização incremental proposta com os trabalhos correlatos de Chow et al. (2014) e Popovych et al. (2014), que serão denotados por *Legalized Cell Move* (LCM) e *Incremental Abacus* (IAB), respectivamente. Estes resultados também são apresentados no trabalho Netto et al. (2016b). A Figura 16 apresenta os fluxogramas de otimização incremental utilizando cada um dos três algoritmos avaliados nesta seção.

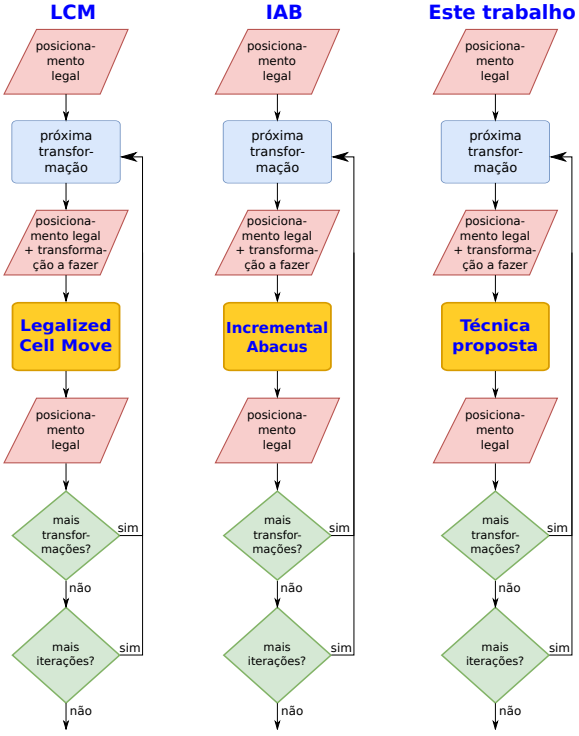


Figura 16 – Fluxogramas das técnicas de legalização incremental avaliadas.

Observe que as três técnicas de legalização são incrementais e, portanto, o fluxograma corresponde ao da estratégia de legalização incremental, apenas especificando o nome de cada técnica utilizada.

Como os códigos dos algoritmos utilizados nas técnicas não estão disponíveis em domínio público, os mesmos foram implementados a partir dos pseudocódigos disponíveis em Chow et al. (2014) e Popovych et al. (2014). Para comparar as diferentes técnicas de legalização incremental, foi utilizado o mesmo vetor de movimentos para cada técnica. Este vetor, com 1 milhão de movimentos, foi gerado usando um gerador pseudo-aleatório e foi aplicado em todos as técnicas, legalizando o circuito após cada movimento. Após realizar todos os movimentos, os resultados obtidos foram comparados.

A Figura 17 compara o tempo de legalização de cada técnica em função do número de legalizações. Este tempo foi acumulado após cada legalização, de forma a avaliar o seu crescimento à medida que o número de legalizações aumenta. Para evitar que um circuito específico possa influenciar a avaliação, o eixo y apresenta o tempo acumulado médio entre todos os circuitos. Ambos os eixos estão em escala logaritmica e cada linha no gráfico representa uma técnica de legalização. A regressão linear na escala logarítmica mostra que a taxa de crescimento do tempo de execução é próxima de 1 para todas as técnicas. No entanto, devido ao termo constante na regressão, o tempo de execução da técnica proposta é por volta de uma ordem de magnitude menor do que o obtido por LCM, e duas ordens de magnitude menor do que o de IAB.

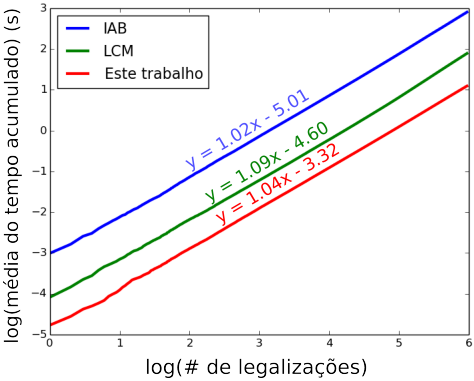


Figura 17 – Comparação das três técncias de legalização incremental, em termos de crescimento do tempo de legalização.

A comparação entre a técnica proposta e LCM permite avaliar de forma direta a aceleração obtida pelo uso da R-tree para realizar buscas espaciais, dado que a técnica proposta adapata o algoritmo utilizado por LCM e, portanto, utiliza a mesma abordagem de legalização

incremental. Por outro lado, ao comparar a técnica proposta com IAB, é possível observar que o uso de uma heurística gulosa permite uma aceleração adicional em relação à programação dinâmica (utilizada por IAB).

Para verificar se esta redução do tempo de legalização foi obtida ao custo de uma menor qualidade da solução final, foram avaliadas a taxa de legalização e a perturbação no posicionamento inicial, ao aplicar cada uma das três técnicas. A Figura 18 compara a taxa de legalização das três técnicas para cada um dos circuitos (eixo x).

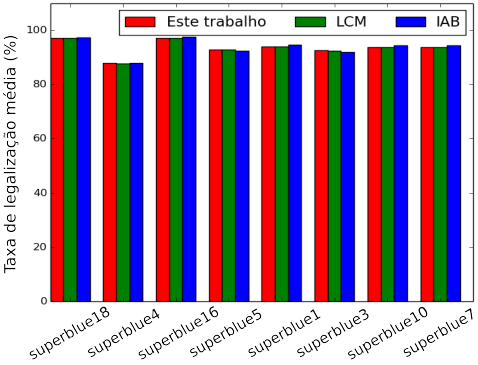


Figura 18 – Comparação das três técncias de legalização incremental, com relação à taxa de legalização.

Como mostrado no Algoritmo 5, caso não seja possível posicionar uma célula na posição alvo, a legalização falha e nenhuma célula é movida. Desta forma, a taxa de legalização (eixo y) foi medida como a proporção de legalizações bem sucedidas com relação ao número total de movimentos realizados. Note que as três técnicas atingem taxas de legalização semelhantes, com um valor médio de 93,4%. Estes resultados mostram que o menor tempo de execução obtido pela técnica proposta não compromete sua taxa de legalização.

A Figura 19 apresenta a perturbação no posicionamento inicial resultante de cada técnica. Esta perturbação (eixo y) representa o deslocamento médio das células (em $\mu$m) considerando todas as legalizações. O deslocamento de cada célula foi medido para todos os circuitos (eixo x) utilizando a Equação (2.7). Note que a técnica proposta resulta na mesma perturbação média que LCM, enquanto que a técnica IAB resulta em uma menor perturbação. Este resultado é esperado, dado que IAB minimiza o deslocamento das células em sua função objetivo.

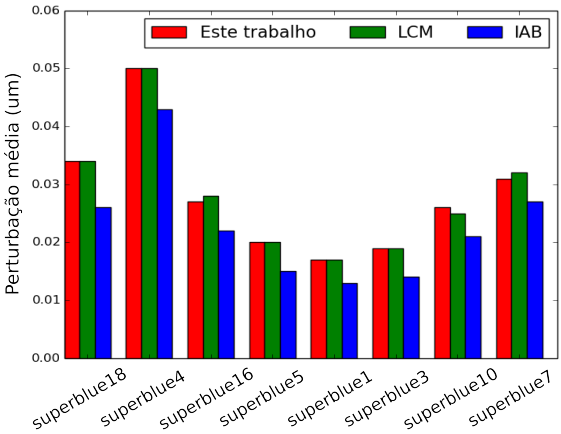


Figura 19 – Comparação das três técncias de legalização incremental, considerando a perturbação média.

A perturbação média resultante da técnica proposta e de LCM foi em média 0,028$\mu$m, enquanto que a perturbação média de IAB foi cerca de 18% menor (0,023$\mu$m). No entanto, a menor perturbação média atingida por IAB não compensa seu tempo de legalização, quase duas ordens de magnitude maior.

# 6 CONCLUSÕES E TRABALHOS FUTUROS

A estratégia de legalização incremental realiza transformações no posicionamento de um circuito enquanto mantém a legalidade do mesmo. Esta estratégia de legalização é particularmente útil durante otimizações incrementais, em que um número pequeno de transformações são realizadas. Neste caso, uma legalização completa resulta em um elevado tempo de execução, conforme demonstrado experimentalmente neste trabalho. No entanto, apesar da estratégia de legalização incremental ser utilizada por diversos trabalhos, nenhum trabalho reporta comparação quantitativa entre diferentes estratégias no contexto prático de uma técnica de otimização incremental. Além disso, e mais importante, algoritmos de legalização incremental presentes na literatura não fazem uso de estruturas de dados especializadas na manipulação de objetos geométricos, tais como layouts de células.

Este trabalho realizou uma avaliação quantitativa das estratégias de legalização (final, iterativa e incremental), e propôs uma técnica de legalização incremental que adapta um algoritmo do estado da arte para fazer uso de uma árvore espacial, chamada R-tree.

A avaliação das estratégias de legalização mostrou que a estratégia incremental foi mais rápida que as outras duas estratégias (sendo até 73% mais rápida que a estratégia final) mas resulta em mais violações de atraso. Isto ocorre porque a estratégia incremental realiza uma série de legalizações locais, com pouca visão do global do circuito, resultando em soluções subótimas. Por outro lado, o impacto na densidade do posicionamento mostrou-se mais relacionado com as características do circuito do que com a estratégia utilizada, enquanto que o impacto no comprimento das interconexões foi desprezível, devido ao pequeno número de transformações realizadas por otimizações incrementais.

A comparação da técnica de legalização incremental proposta com outras técnicas de legalização incremental do estado da arte (LCM e IAB) mostrou que o uso da R-tree acelerou as buscas espaciais necessárias para o algoritmo de legalização. Desta forma, a técnica proposta é de 1 a 2 ordens de grandeza mais rápida do que as outras técnicas, enquanto legaliza um número semelhante de células. No entanto, a técnica proposta resultou em 18% mais perturbações no circuito, quando comparada com IAB. Por outro lado, o tempo de execução elevado de IAB (cerca de 2 ordens de grandeza maior) não compensa sua menor perturbação.

Como trabalhos futuros, é necessário melhorar a qualidade da

solução encontrada pelos algoritmos de legalização incremental para reduzir a perturbação causada no posicionamento inicial e, consequentemente, reduzir as violações de atraso. Além disso, restrições adicionais podem ser avaliadas durante a legalização, tais como restrições de roteamento. Os resultados apresentados neste trabalho ainda podem ser comparados com os obtidos por ferramentas comerciais, de forma a investigar o desempenho dos algoritmos de legalização utilizados por aquelas ferramentas.

Árvores espaciais, tal como a R-tree, também podem ser utilizadas para acelerar outras etapas de síntese física que fazem uso de informações geométricas do circuito. Por exemplo, a R-tree pode ser utilizada para identificar registradores próximos em uma etapa de *clustering*, de forma a agrupá-los em um único cluster. Por fim, é necessário avaliar o potencial de paralelização das diferentes estratégias de legalização. Alguns algoritmos de legalização completa (ex: Abacus) possuem grande potencial de paralelização. Este potencial pode ser estendido para algoritmos de legalização incremental, para que estes continuem atrativos em ambientes multiprocessados.

# REFERÊNCIAS

ALPERT, C.; CHU, C.; VILLARRUBIA, P. The coming of age of physical synthesis. In: *ICCAD*. [S.l.: s.n.], 2007. p. 246–249.

ALPERT, C. et al. Placement: hot or not? In: *ICCAD*. [S.l.: s.n.], 2012. p. 283–290.

BOOST. *Boost C++ Libraries*. 2015. `https://boost.org/`.

BRENNER, U. Vlsi legalization with minimum perturbation by iterative augmentation. In: IEEE. *DATE*. [S.l.], 2012. p. 1385–1390.

CHO, M. et al. History-based vlsi legalization using network flow. In: ACM. *DAC*. [S.l.], 2010. p. 286–291.

CHOI, W.; BAZARGAN, K. Incremental placement for timing optimization. In: IEEE COMPUTER SOCIETY. *Proceedings of the 2003 IEEE/ACM international conference on Computer-aided design*. [S.l.], 2003. p. 463.

CHOW, W. et al. Cell density-driven detailed placement with displacement constraint. In: *ISPD*. [S.l.: s.n.], 2014. p. 3–10.

CHOWDHARY, A. et al. How accurately can we model timing in a placement engine? In: *DAC*. [S.l.: s.n.], 2005. p. 801–806.

CHU, C.; WONG, Y. Flute: Fast lookup table based rectilinear steiner minimal tree algorithm for vlsi design. *TCAD*, IEEE, v. 27, n. 1, p. 70–83, 2008.

GUTH, C. et al. Timing-driven placement based on dynamic net-weighting for efficient slack histogram compression. In: *ISPD*. [S.l.: s.n.], 2015. p. 141–148.

HILL, D. *Method and system for high speed detailed placement of cells within an integrated circuit design*. [S.l.]: Google Patents, abr. 9 2002. US Patent 6,370,673.

HO, T.-Y.; LIU, S.-H. Fast legalization for standard cell placement with simultaneous wirelength and displacement minimization. In: SPRINGER. *IFIP/IEEE International Conference on Very Large Scale Integration-System on a Chip*. [S.l.], 2010. p. 291–311.

HUANG, C.-C. et al. Detailed-routability-driven analytical placement for mixed-size designs with technology and region constraints. In: IEEE. *Computer-Aided Design (ICCAD), 2015 IEEE/ACM International Conference on*. [S.l.], 2015. p. 508–513.

KAHNG, A. B. et al. *VLSI physical design: from graph partitioning to timing closure*. [S.l.]: Springer Science & Business Media, 2011.

KEATING, M. et al. *Low power methodology manual: for system-on-chip design*. [S.l.]: Springer Publishing Company, Incorporated, 2007.

KENNINGS, A.; DARAV, N. K.; BEHJAT, L. Detailed placement accounting for technology constraints. In: IEEE. *2014 22nd International Conference on Very Large Scale Integration (VLSI-SoC)*. [S.l.], 2014. p. 1–6.

KIM, M. et al. Iccad-2015 cad contest in incremental timing-driven placement and benchmark suite. In: *ICCAD*. [S.l.: s.n.], 2015. p. 921–926.

LI, S.; KOH, C.-K. Analytical placement of mixed-size circuits for better detailed-routability. In: IEEE. *2014 19th Asia and South Pacific Design Automation Conference (ASP-DAC)*. [S.l.], 2014. p. 41–46.

LUO, T.; NEWMARK, D.; PAN, D. A new lp based incremental timing driven placement for high performance designs. In: *DAC*. [S.l.: s.n.], 2006. p. 1115–1120.

MANOLOPOULOS, Y. et al. *R-trees: Theory and Applications*. [S.l.]: Springer Science & Business Media, 2010.

MARKOV, I. L.; HU, J.; KIM, M.-C. Progress and challenges in vlsi placement research. In: IEEE. *ICCAD*. [S.l.], 2012. p. 275–282.

NETTO, R. et al. Evaluating the impact of circuit legalization on incremental optimization techniques. In: IEEE. *Integrated Circuits and Systems Design (SBCCI), 2016 29th Symposium on*. [S.l.], 2016. p. 1–6.

NETTO, R. et al. Speeding up incremental legalization with fast queries to multidimensional trees. In: IEEE. *VLSI (ISVLSI), 2016 IEEE Computer Society Annual Symposium on*. [S.l.], 2016. p. 36–41.

PAPA, D. et al. Rumble: an incremental timing-driven physical-synthesis optimization algorithm. *TCAD*, IEEE, v. 27, n. 12, p. 2156–2168, 2008.

POPOVYCH, S. et al. Density-aware detailed placement with instant legalization. In: *DAC*. [S.l.: s.n.], 2014. p. 1–6.

PUGET, J. C. et al. Jezz: An effective legalization algorithm for minimum displacement. In: ACM. *SBCCI*. [S.l.], 2015.

REN, H. et al. Hippocrates: first-do-no-harm detailed placement. In: *ASP-DAC*. [S.l.: s.n.], 2007. p. 141–146.

REN, H.; PAN, D.; KUNG, D. Sensitivity guided net weighting for placement-driven synthesis. *TCAD*, IEEE, v. 24, n. 5, p. 711–721, 2005.

REN, H. et al. Diffusion-based placement migration. In: ACM. *DAC*. [S.l.], 2005. p. 515–520.

SPINDLER, P.; SCHLICHTMANN, U.; JOHANNES, F. M. Abacus: fast legalization of standard cell circuits with minimal movement. In: ACM. *ISPD*. [S.l.], 2008. p. 47–53.

# APÊNDICE A – Lista de Publicações e Prêmios

## A.1 ARTIGOS PUBLICADOS DIRETAMENTE RELACIONADOS AO TEMA DE MESTRADO

A avaliação quantitativa das estratégias de legalização, assim como a técnica proposta de legalização incremental, resultaram em duas publicações diretamente relacionadas ao tema deste trabalho de mestrado. Os detalhes destas publicações estão listados nesta Seção.

### A.1.1 *Proceedings of the 29th Symposium on Integrated Circuits and Systems Design*, 2016

- **Qualis CC 2012:** B1
- **Título:** *Evaluating the Impact of Circuit Legalization on Incremental Optimization Techniques*
- **Autores: NETTO, Renan** ; LIVRAMENTO, Vinicius ; GUTH, Chrystian ; SANTOS, Luiz C. V. ; GüNTZEL, José Luís
- ***Abstract:*** *During physical synthesis, global placement and incremental optimization steps such as gate sizing, buffer insertion and timing-driven placement, produce placements where cells are overlapped or misaligned with respect to sites and rows predefined in the used standard cell library. Therefore, a legalization procedure must be used to keep the placement legality. In the case of incremental optimization techniques, the legalization step can be applied as a final step, after each optimization iteration or even after each primitive placement transformation. Although different legalization strategies are used by different works on incremental optimization techniques found in the literature, they do not present practical results on how those different strategies impact on the runtime and quality of the optimized placements. This work investigates such issue by comparing the runtime, timing degradation, density profile and wirelength resulted from using three legalization strategies on an incremental timing driven placement technique. Experimental results show that incremental legalization is from 22% to 71% faster than a single final legalization step and achieves similar density and wirelength profiles, but may increase timing violations up to 48% (9% on average). On the other hand, legalizing the circuit after each optimization iteration is the least interesting strategy, since it leads to similar quality results*

*than a single final legalization, at the cost of a runtime about 3.2 times greater on average. Therefore, incremental legalization may be an interesting strategy for incremental optimization techniques due to its shorter runtime. Finally, we point out alternatives to improve the quality of incremental legalization techniques.*

### A.1.2 *Proceedings of the 23rd IEEE International Conference on Electronics, Circuits and Systems*, 2016

- **Qualis CC 2012:** B1
- **Título:** *Speeding up Incremental Legalization With Fast Queries to Multidimensional Trees*
- **Autores: NETTO, Renan** ; LIVRAMENTO, Vinicius ; GUTH, Chrystian ; SANTOS, Luiz C. V. ; GüNTZEL, José Luís
- ***Abstract:*** *Circuit legalization removes overlaps and keeps cell alignment with power rails while minimizing total cell displacement. Legalization is applied not only after global placement, but also after incremental optimization steps like detailed placement, gate sizing, and buffer insertion. Applying full legalization after such incremental optimizations is too time-consuming. That is why physical synthesis has been shifting from entire circuit legalization to incremental mode legalization, which keeps legality after every primitive transformation. Unfortunately, recent incremental legalization strategies employ data structures that are not suitable for handling geometric data. This work proposes a new technique that relies on an R-tree, a data structure tailored to efficient geometric data storage where objects are represented by their minimum bounding box rectangles, which allows for fast spatial queries. As compared with state-of-the-art incremental legalization algorithms, the proposed technique is at least 6 times faster and performs as many successful legalizations.*

## A.2 PARTICIPAÇÃO EM OUTRAS PUBLICAÇÕES DURANTE O MESTRADO

Durante a realização deste trabalho de mestrado, houve participação em outras publicações que, apesar de não estarem diretamente

relacionados com o problema da legalização, utilizaram alguma das técnicas de legalização incremental apresentadas neste trabalho. Portanto, os detalhes destas publicações estão listados nesta Seção.

### A.2.1 *Proceedings of the International Conference on Computer-Aided Design*, 2015

- **Qualis CC 2015:** A1

- **Título:** *Exploiting Non-Critical Steiner Tree Branches for Post-Placement Timing Optimization*

- **Autores:** LIVRAMENTO, Vinicius ; GUTH, Chrystian ; **NETTO, Renan** ; GüNTZEL, José Luís ; SANTOS, Luiz C. V.

- ***Abstract:*** *The increasing impact of interconnections on the overall circuit performance renders physical design a crucial step to timing closure. Several techniques are used to optimize timing within the flow, such as gate sizing, buffer insertion, and timing-driven placement (TDP). Unfortunately, gate sizing and buffer insertion are not capable of modifying the length of interconnections. Although TDP is able to shorten critical interconnection by finding new legal locations for a subset of cells, it generally overlooks the impact of non-critical branches on the delay of critical cells. This work proposes a post-placement timing optimization technique to reduce the capacitive load of critical cells by shortening non-critical Steiner tree branches. To shorten such branches, our technique uses computational geometry for finding effective cell movements that consider maximum displacement constraints and macro blocks. Our experiments evaluate the capability of our technique to further reduce the timing violations from a TDP solution. We applied our technique on the solutions obtained by the top 3 teams in the ICCAD 2014 TDP Contest, where short and long displacement constraints are defined. For the short constraints, the average reductions assuming worst and total late negative slack metrics are 23% and 34%. Considering the long constraints, the average reductions are 62% and 67%. We also present extensions of our technique to tackle related physical design problems such as early violations reduction and electrical correction.*

### A.2.2 *ACM Transactions on Design Automation of Electronic Systems*, 2016

- **Qualis CC 2015:** B1
- **Título:** *Clock-Tree-Aware Incremental Timing-Driven Placement*
- **Autores:** LIVRAMENTO, Vinicius ; **NETTO, Renan** ; GUTH, Chrystian ; GüNTZEL, José Luís ; SANTOS, Luiz C. V.
- ***Abstract:*** *The increasing impact of interconnections on overall circuit performance makes timing-driven placement (TDP) a crucial step toward timing closure. Current TDP techniques improve critical paths but overlook the impact of register placement on clock tree quality. On the other hand, register placement techniques found in the literature mainly focus on power consumption, disregarding timing and routabilty. Indeed, postponing register placement may undermine the optimization achieved by TDP, since the wiring between sequential and combinational elements would be touched. This work proposes a new approach for an effective coupling between register placement and TDP that relies on two key aspects to handle sequential and combinational elements separately: only the registers in the critical paths are touched by TDP (in practice they represent a small percentage of the total number of registers), and the shortening of clock tree wirelength can be obtained with limited variation in signal wirelength and placement density. The approach consists of two steps: (1) incremental register placement guided by a virtual clock tree to reduce clock wiring capacitance while preserving signal wirelength and density, and (2) incremental TDP to minimize the total negative slack. For the first step, we propose a novel technique that combines clock-net contraction and register clustering forces to reduce the clock wirelength. For the second step, we propose a novel Lagrangian Relaxation formulation that minimizes total negative slack for both setup and hold timing violations. To solve the formulation, we propose a TDP technique using a novel discrete search that employs a Euclidean distance to define a proper neighborhood. For the experimental evaluation of the proposed approach, we relied on the ICCAD 2014 TDP contest infrastructure and compared our results with the best results obtained from that contest in terms of timing closure, clock tree compactness, signal wirelength, and density. Assuming a long displacement*

*constraint, our technique achieves worst and total negative slack reductions of around 24% and 26%, respectively. In addition, our approach leads to 44% shorter clock tree wirelength with negligible impact on signal wirelength and placement density. In the face of such results, the proposed coupling seems a useful approach to handle the challenges faced by contemporary physical synthesis.*

## A.3 PRÊMIOS

### A.3.1 2015 CAD Contest (Problem C: Incremental Timing-Driven Placement) @ ICCAD 2015

A técnica de legalização apresentada neste trabalho foi utilizada na ferramenta de posicionamento incremental guiado por atraso submetida à competição ICCAD 2015 CAD Contest (problem C: Incremental Timing-Driven Placement), a qual proporcionou à equipe do Embedded Computing Laboratory da UFSC o primeiro lugar na competição.

- **Equipe:** First Place (`cada014`)
- **Membros:** Vinícius Livramento, Chrystian Guth, Renan Netto, Prof. José Güntzel e Prof. Luiz Santos
- **Colocação:** 1º Lugar