UNIVERSIDADE FEDERAL DA FRONTEIRA SUL
CAMPUS CHAPECÓ
CURSO DE CIÊNCIA DA COMPUTAÇÃO

GRADE DE HANAN DINÂMICA

HENRIQUE JOSÉ DALLA CORTE

CHAPECÓ
2018

HENRIQUE JOSÉ DALLA CORTE

GRADE DE HANAN DINÂMICA

Trabalho de conclusão de curso de graduação apresentado como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação da Universidade Federal da Fronteira Sul.
Orientador: Prof. Dr. Emílio Wuerges

CHAPECÓ
2018

Corte, Henrique José Dalla

Grade de Hanan Dinâmica / por Henrique José Dalla Corte. – 2018.
44 f.: il.; 30 cm.

Orientador: Emílio Wuerges
Monografia (Graduação) - Universidade Federal da Fronteira Sul, Ciência da Computação, Curso de Ciência da Computação, SC, 2018.

1. VLSI. 2. Hanan. 3. Arvore de Steiner. 4. Steiner. 5. ICCAD 2017. 6. Ponto de Steiner. 7. Arvore de Intervalo. I. Wuerges, Emílio. II. Título.

E-mail: henrique.dcorte@gmail.com

HENRIQUE JOSÉ DALLA CORTE

GRADE DE HANAN DINÂMICA

Trabalho de conclusão de curso de graduação apresentado como requisito para obtenção do grau de Bacharel em Ciência da Computação da Universidade Federal da Fronteira Sul.

Orientador: Prof. Dr. Emílio Wuerges

Aprovado em: 05\12\2018

BANCA EXAMINADORA:

Dr. Emílio Wuerges - UFFS

Doglas André Finco - UFFS

Me. Adriano Sanick Padilha - UFFS

AGRADECIMENTOS

Agradeço a Deus por ter me dado forças para superar minhas dificuldades e nunca ter me deixado faltar fé em momento algum.

A minha mãe e vó minhas heroínas por mesmo em caminhos tortuosos e ásperos sempre me apoiarem, pelos seus cuidados comigo ao longo de minha vida, pelo incentivo nas horas difíceis, de desânimo e cansaço.

A meu irmão por ser um motivo de orgulho para mim e ser o motivo de meu esforço sempre.

Meus agradecimentos aos meus amigos, muitos conquistados dentro desta caminhada acadêmica, companheiros de trabalhos e irmãos na amizade que ajudaram na minha formação e estarão sempre presentes.

Obrigado ao meu orientador, pelo magnifico auxílio na elaboração deste trabalho, a banca examinadora pela contribuição e tempo para obtenção de melhores resultados, aos demais professores atuantes em minha formação por me proporcionar a oportunidade de obter novos conhecimentos, não apenas racionais, mas de caráter e em minha educação para formação profissional plena, pela sua dedicação em ensinar.

RESUMO

Com a lei de Moore a construção de componentes de larga escala utilizando-se de circuitos integrados, ou seja, uma *Very-large-scale integration* (VLSI) se torna cada vez mais complexa, para isto são utilizadas ferramentas que buscam construir modelos para uma VLSI. Com isto, a proposta deste trabalho é a implementação de uma grade de Hanan dinâmica com complexidade de espaço inferior a O(n^2) em ferramentas que se utilizem da *Rectilinear Steiner Minimum Tree* (RSMT) que buscam a integração de circuitos. Tendo como contribuição, além da implementação da grade de Hanan dinâmica, uma aplicação utilizando caminhos minímos e uma revisão bibliográfica do funcionamento da mesma. A ferramenta é validada por casos de testes dados pelo *International Conference On Computer Aided Design* (ICCAD) 2017.

Palavras-chave: VLSI. Hanan. Arvore de Steiner. Steiner. ICCAD 2017. Ponto de Steiner. Arvore de Intervalo.

ABSTRACT

With Moore's law, the construction of large-scale components using integrated circuits, ie a very-large-scale integration (VLSI) becomes increasingly complex, for this are used tools that seek to build models for a VLSI. Therefore, the proposed work is the implementation of a dynamic Hanan grid with less than $O(n^2)$ complexity for tools that use the Regenerinear Steiner Minimum Tree (RSMT) to search for circuit integration. In addition to the implementation of the dynamic Hanan grid, an application using minimal paths and a bibliographical review of its operation. The tool is validated by test cases provided by the International Conference On Computer Aided Design (ICCAD) 2017

Keywords: VLSI. Hanan. Steiner Tree. Steiner. ICCAD 2017. Steiner Point, Interval Tree.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ALGORITMOS

LISTA DE APÊNDICES

LISTA DE ABREVIATURAS E SIGLAS

ICCAD	*International Conference On Computer Aided Design*
ST	*Steiner Tree*
RSMTP	*Rectilinear Steiner Minimum Tree Problem*
RSMT	*Rectilinear Steiner Minimum Tree*
VLSI	*Very-large-scale integration*
RST	*Rectilinear Steiner Tree*
RFST	*Rectilinear Full Steiner Tree*

SUMÁRIO

1 INTRODUÇÃO

A grade de Hanan é definida por (HANAN, 1966) como o resultado do tracejamento de linhas verticais e horizontais sobre os conjuntos de terminais, tornando as intersecções das linhas pontos de Steiner, estes utilizados para reduzir a árvore miníma resultante.

Segundo (ESMAEILZADEH et al., 2011) a construção e modelagem de uma VLSI está tornando-se cada vez mais complexa para conseguir acompanhar a Lei de Moore, que prevê que a cada 18 meses o número de transistores dobre com mesmo custo da geração anterior. Atualmente para modelar uma VLSI complexa os engenheiros utilizam-se de ferramentas automatizadas que auxiliam na construção do circuito integrado. Em muitas destas ferramentas utilizam-se de algoritmos que baseiam-se em uma grade de Hanan para aplicar seu algoritmo de solução para gerar um circuito integrado de larga escala dos componentes.

Por estes motivos foi criado o problema "*Net Open Location Finder with Obstacles*" (SHEN; KAI-SHUN HU SYNOPSYS TAIWAN CO., 2017), dentro do Contest (COMPUTER AIDED DESIGN, 2017) (ICCAD). Algoritmos que se propõe a resolver este problema, como por exemplo o *Iterated 1-Steiner* (SILVA, 2009) utiliza-se de uma grade de Hanan, entretanto, a mesma é muito custosa para um construção completa em memória que levaa uma complexidade O(n^2) em utilização de memória, tornando assim necessário um elevado nível em armazenamento de memória cache.

Para entender como é construída a grade de Hanan, é necessário do conceito de árvore de Steiner ou *Steiner Tree* (ST), com a variação mais importante para este trabalho a Árvore de Steiner Retilínea ou *Rectilinear Steiner Tree* (RST) (HWANG, 1976). Sendo assim, para posteriormente aplicar a grade de Hanan em um algoritmo de caminhos mínimos ou como o de Dijkstra, este que no presente trabalho conta com o auxílio árvore de intervalos para identificação de componentes que pode ser vista em (CORMEN et al., 2009).

Em conclusão, busca-se apresentar uma construção da grade de Hanan dinâmica para o problema "*Net Open Location Finder with Obstacles*" do Contest (COMPUTER AIDED DESIGN, 2017), posteriormente demonstrando que é possível utilizá-la em um algoritmo ou ferramenta de caminhos mínimos que pode ser usado para encontrar uma árvore miníma, demonstrando-se uma melhoria em memória volátil.

1.1 Estrutura do Trabalho

O restante deste trabalho está estruturado da seguinte forma: o próximo capítulo 2 apresenta a justificativa. No seu sucessor capítulo 3 exposto a fundamentação teórica. Já no capítulo 4 são apresentados os trabalhos relacionados. No capítulo 5 apresenta-se o projeto de implementação. Na sequência, o capítulo 6 contém os resultados obtidos no desenvolvimento do trabalho. Por fim, no capítulo 7 são apresentadas as conclusões.

2 JUSTIFICATIVA

Este capítulo tem o objetivo de definir o problema, a justificativa da realização deste trabalho e demonstrar os objetivos que foram alcançados ao longo do trabalho.

2.1 Problema

Com a necessidade e o objetivo do acompanhamento da Lei de Moore, a construção de VLSI's se torna complexa, pois a Moore prediz que se tenha o dobro de transistores com mesmo custos a cada 18 meses. Nisto, surge o intuito de problemas indústrias serem trazidos para os meios acadêmicos, ou seja,o Contest da *International Conference On Computer Aided Design* é onde são apresentados problemas industriais (COMPUTER AIDED DESIGN, 2017) .

Com o motivo anteriormente citado já existiria uma justificava de implementação de novos algoritmos para construção de grade de Hanan para produção circuitos integrados de larga escala complexos. Para modelagem de uma VLSI pode-se usar uma grade de Hanan, mas quando construído de forma completa tem complexidade O(n^2) em espaço de memória com n sendo a quantidade de objetos de entrada.

Portanto, o problema é construir uma grade de Hanan com complexidade inferior a O(n^2) em utilização de memória de um computador. Visto que, o crescimento da grade acontece na complexidade n^2 e com apenas 3 componentes em uma VLSI são gerados 36 pontos na grade para serem armazenados em memória cache enquanto o algoritmo estiver sendo executado como exemplo na Figura 2.1 onde se vê a proporção de crescimento, onde x é o número de componentes na entrada e y é o total de pontos da grade de Hanan.

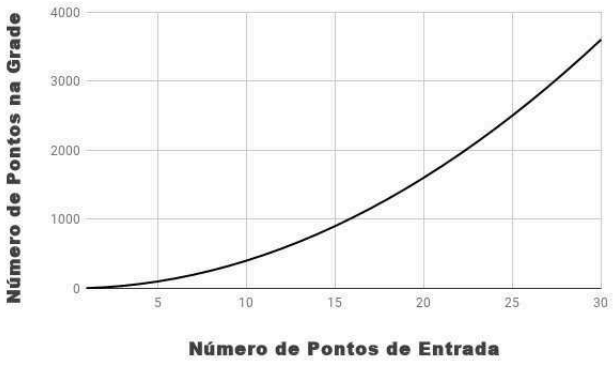

Figura 2.1 – Exemplo de Gráfico da Expectativa de Pontos da grade de Hanan

2.2 Objetivos

O objetivo deste trabalho foi realizar a implementação da grade de Hanan dinâmica, utilizando-se da técnica de armazenamento das coordenadas de cada ponto dos objetos em vetores de elementos únicos ordenados de forma crescente. Tem como objetivo de resultado final, uma redução de complexidade em espaço de armazenamento em memória volátil, como também, a contribuição bibliográfica sobre grades de Hanan, pontos de Steiner, Árvore de Steiner Retilínea e Árvore de Intervalos, com resultados obtidos usando a técnica implementada proposta para este trabalho.

3 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo serão apresentadas técnicas de árvores geradoras mínimas de Steiner e caminho mínimo, que são aplicados em grafos não-dirigidos com custos positivos nas arestas neste caso. O custo de um subgrafo não-dirigido T de G é a soma dos custos das arestas de T (CORMEN et al., 2009).

O problema proposto trata, de certa forma de uma utilização de grafos como circuitos, considerando arestas como fios e vértices como pontos de componentes. Como circuitos tem obstáculos e visto os requisitos do problema, os pontos provenientes do obstáculo devem manter uma distância do ponto original para proporcionar a oportunidade de desvio seguro, assim como a preposição de que navegar em um componente não existe custo. Na Figura 3.1 as arestas e pontos incluídos na área do componente são parte de um componente, portanto estas arestas tem custo zero de navegação.

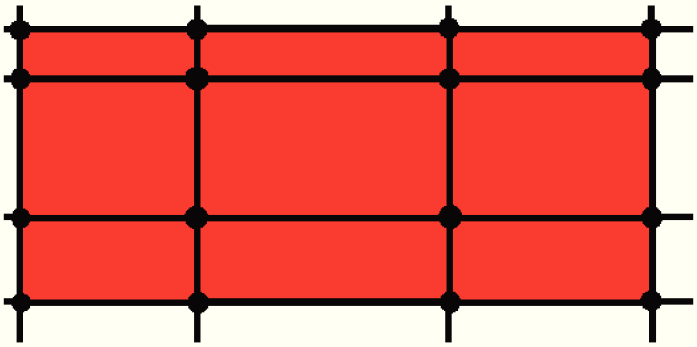

Figura 3.1 – Exemplo de Componente com Pontos e Arestas Internos

3.1 Árvore de Steiner

O problema de árvore mínima de Steiner, referente a Jakob Steiner, é um problema de otimização combinatória, objetivando encontrar a menor interconexão de um conjunto (HWANG, 1976). Este problema é conhecido como problema da auto-estrada, com este nome pode-se perceber que tem semelhança ao problema proposto a ser solucionado neste trabalho.

A árvore de Steiner (ST) consiste em um conjunto de vértices interligados por arestas tendo o menor tamanho de comprimento das soma de comprimentos alçando todas as arestas (LU; TANG; LEE, 2003). Nesta árvore os vértices e arestas intermediários são adicionados ao grafo, reduzindo assim o comprimento do tamanho de expansão. Portanto, diminuindo o com-

primento total de ligação, estes são os pontos ou vértices de Steiner, sendo que estas ligações formam uma árvore, ou seja a árvore de Steiner.

Considerando que a maioria dos problemas da árvore de Steiner são NP-Completos, tendo um entre os vinte e um problemas originais NP-completos de Karp (SILVA, 2009). Podendo ter casos resolvidos em tempo polinomial, sendo quase sempre utilizadas heurísticas.

3.2 Árvore de Steiner Retilínea Mínima

A Árvore Mínima de Steiner Retilínea (RSMT), ou o Problema da Árvore de Steiner Retilínea Mínima, é uma variante geométrica da ST, no qual a distância euclidiana se torna distância retilínea (HWANG, 1976).

Este problema é dado por n pontos sendo todos interligados por uma rede de arestas mais curtas que são partes de linhas arestas verticais e horizontais, sendo esta rede uma árvore com vértices de entrada e mais os vértices extras, ou seja, Pontos de Steiner como nas topologias da Figura 3.2.

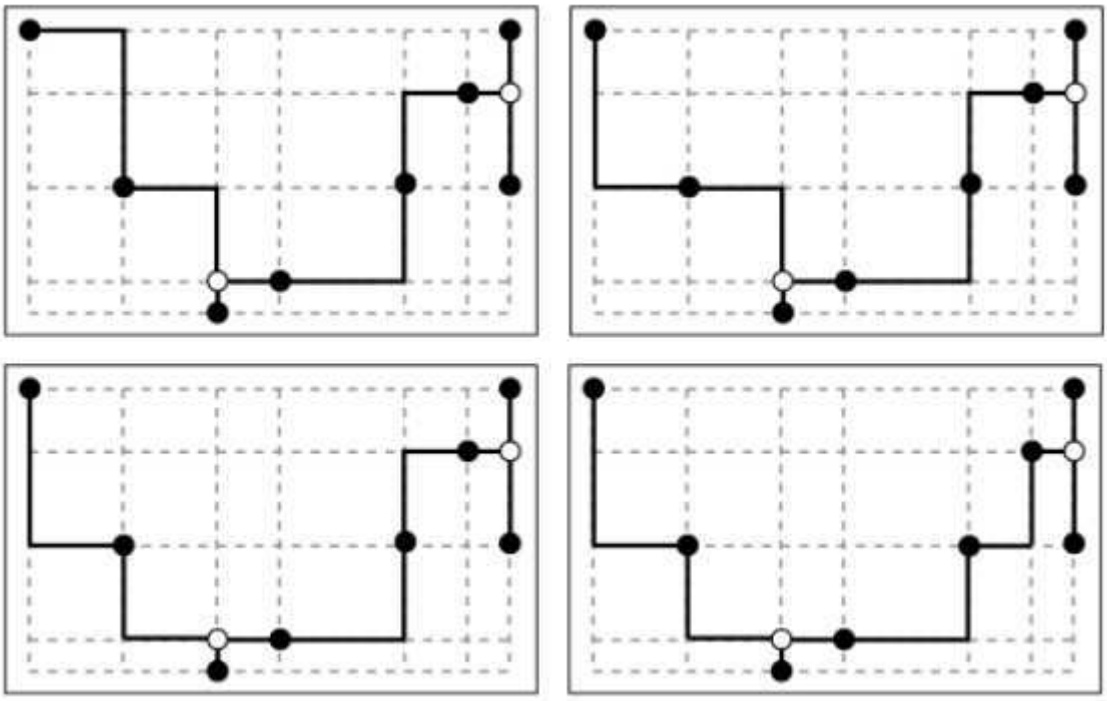

Figura 3.2 – Topologias de uma mesma Árvore Retilínea de Steiner (SILVA, 2009)

O problema é usado para o design da automação de projetos eletrônicos. Em circuitos VLSI, a rede de fios são feitas apenas verticalmente e horizontalmente devido ao processo de fabricação de circuitos desenhados com este técnica é menos propenso a falhas e mais barato.

Concluindo, a soma dos segmentos verticais e horizontal é o comprimento do fio e a distância de dois pontos interligados nesta rede é a distância retilínea entre pontos geométricos no plano do projeto de design.

A RSMT pode ser restrita a uma grade de Hanan, como uma rede com linhas verticais e horizontais. Este é um problema NP-difícil, por isso é mais comum se usar algoritmos aproximados, heurísticos e por casos especiais com soluções.

3.3 Grade de Hanan

A grade de Hanan é o resultado de uma grande pesquisa sobre o problema de RSMT e técnicas de construção de RSMT feita por (HANAN, 1966). Ela é construída através do tracejamento de linhas verticais e horizontais sobre os conjuntos de terminais como na Figura 3.3, neste caso os pontos de vértice dos componentes.

Sabe-se que existe pelo menos uma solução ótima para gerar uma RSMT que utiliza os pontos das intersecções das linhas (HANAN, 1966). A grade de Hanan oportuniza a redução do problema de RSMT, com total de (n^2) pontos, com n terminais e $(n^2 - n)$ pontos de Steiner.

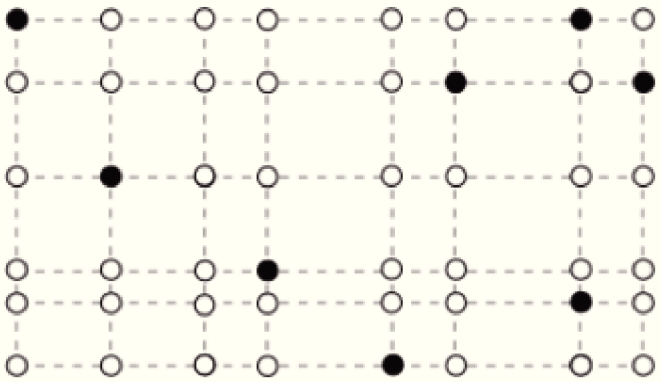

Figura 3.3 – Teorema da grade de Hanan (SILVA, 2009)

3.4 Caminhos Mínimos

Em um problema de caminhos mínimos, existe um grafo dirigido G(V, E) com função peso $w : E \to \mathbb{R}$ que mapeia as arestas em pesos. O peso (custo) do caminho p = (v_0, v_1, ..., v_k) é a soma dos pesos das arestas que os constituem:

$$w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i) \tag{3.1}$$

Definindo o peso do caminho mínimo de u a v por:

$$\delta(u,v) = \begin{cases} min\left\{w(p) : u \overset{p}{\rightsquigarrow} v\right\} \\ \infty \end{cases} \tag{3.2}$$

Portanto, o caminho mínimo do ponto u ao ponto v é definido como um caminho qualquer p com custo $w(p) = \delta(u,v)$ como por exemplo a Figura 3.4 (CORMEN et al., 2009).

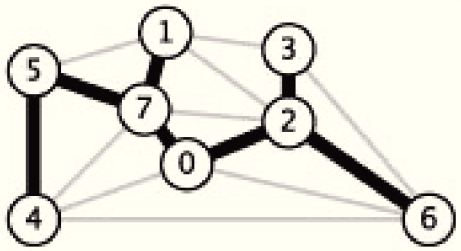

Figura 3.4 – Exemplo Caminho Mínimo

3.4.1 Algoritmo de Dijkstra

O algoritmo de Dijkstra resolve o problema de caminhos mínimos de única fonte em grafo dirigido ponderado $G = (V,E)$ com pesos nas arestas não negativo. Este algoritmo mantém um conjunto S de vértices com pesos finais do caminhos mínimos que partem da fonte s já foram determinados. O algoritmo seleciona os pontos $u \in V - S$ e relaxa as arestas que saem de u (CORMEN et al., 2009).

O processo de relaxar uma aresta (u,v), consiste em testar se é possível melhorar o caminho até v que já foi encontrado passando por u neste passo, caso positivo atualiza-se $v.d$ e $v.pi$. A etapa de relaxamento pode diminuir o caminho mínimo $v.d$ e atualizar o predecessor de $v, v.pi$ com o custo de tempo O(1) (CORMEN et al., 2009).

3.5 Árvore de Intervalos

As árvores de intervalos são a ampliação das árvores vermelho-preto para suportar operações em um conjunto dinâmicos de intervalos (CORMEN et al., 2009). Um intervalo fechado é o par ordenado de números reais [x,y], com $x \leq y$, incluindo as extremidades do intervalo. Já o semi-aberto e aberto excluem uma ou ambas das extremidades.

Intervalos são convenientes para representar eventos em períodos contínuos de tempo. Entretanto, para este trabalho é usado para representar o intervalo que uma aresta ocupa dentro da área determinada para o circuito integrado.

Para representar um intervalo [x,y] (CORMEN et al., 2009) usa um objeto i, com atribuição i.baixo = x (o menor valor do intervalo) e i.alto = y (o maior valor do intervalo). Com os intervalos i e i' se sobrepõem se $i \cap i' \neq \emptyset$, satisfazem a tricotomia de intervalos, ou seja, se as propriedades são válidas:

a. i e i' se sobrepõem.

b. i está à esquerda de i' (isto é, i.alto < i'.baixo).

c. i está à direita de i' (isto é, i'.alto < i.baixo).

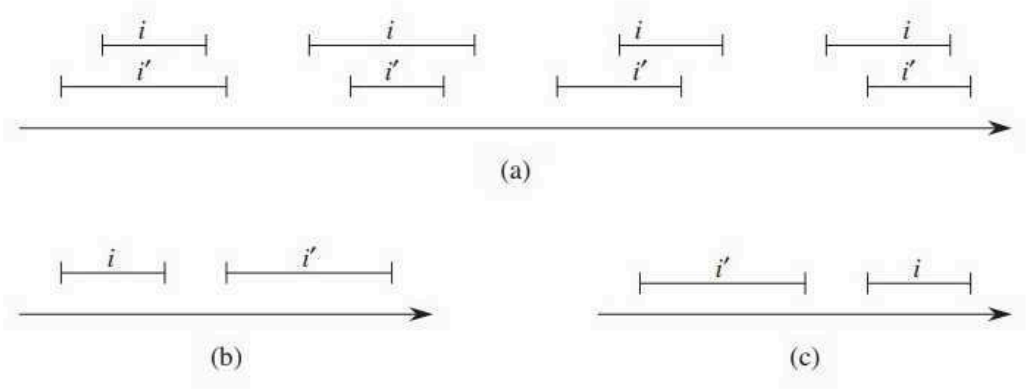

Figura 3.5 – A tricotomia de intervalos (a), (b) e (c) (CORMEN et al., 2009)

Como dito anteriormente e demonstrado na Figura 3.5 a árvore de intervalo mantém um conjunto dinâmico de elementos, sendo que cada elemento n tem um intervalo. A árvore de intervalos suporta as operações de inserção e remoção de um elemento n na árvore de intervalo, assim como seu intervalo n.*int* e a busca na árvore, o qual retorno se existe um elemento com o intervalo que se sobrepõe ao intervalo buscado ou retorna a inexistência deste, tendo um exemplo prático na Figura 3.6.

De acordo com (CORMEN et al., 2009) em seu livro, no subcapítulo "Árvore De Busca Binária Construídas Aleatoriamente" apresentou o teorema 12.4, o qual mostra e prova que a altura esperada de uma árvore de busca binária construída aleatoriamente em n chaves distintas é O($log\, n$) por este motivo a entrada para a árvore de intervalos utilizada por este trabalho é feita aleatoriamente de chaves distintas.

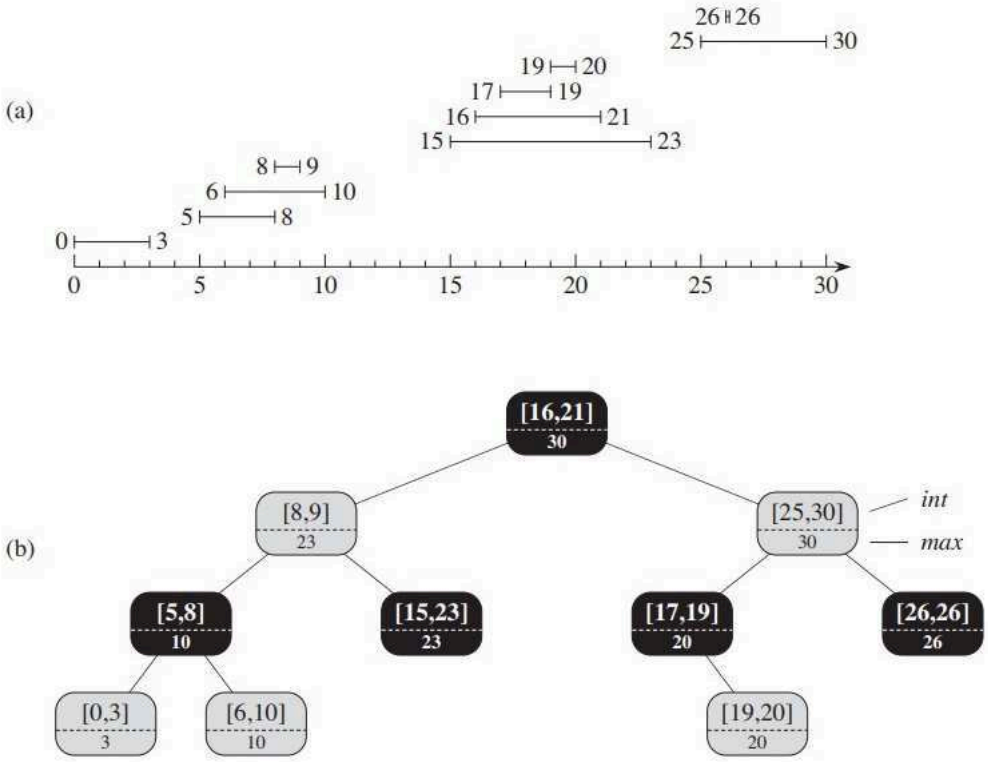

Figura 3.6 – Uma árvore de intervalos, com (a) Conjunto de 10 intervalos ordenados do ponto extremo esquerdo, (b) A árvore de intervalo que representa o conjunto (a) ordenada pelo ponto extremo esquerdo, contendo em cada elemento também o valor máximo extremo da sua respectiva sub-árvore. (CORMEN et al., 2009)

4 TRABALHOS RELACIONADOS

Existem vários trabalhos na literatura que utilizam-se da grade de Hanan em ferramentas de caminhos mínimos e em árvores mínimas de Steiner que buscam construir modelos de circuitos integrados. (SILVA, 2009) apresenta uma revisão da literatura com os trabalhos mais relevantes sobre heurísticas com a árvore de *Steiner Rectilinear*. Porém, este trabalho foca na construção da grade de Hanan dinâmica, assim foram selecionados três trabalhos que englobam esta proposta.

O teorema da grade de Hanan é dado pelo resultado de pesquisas sobre RSMTP, com a grande maioria das técnicas de construção de RSMT. Sendo o teorema em resumo a demonstração da utilização da distância retilínea e os pontos de Steiner tornando-se capaz de construir uma grade com linhas verticais e horizontais sobre pontos do conjunto de elementos, criando pela intersecção destas linhas os pontos de Steiner (HANAN, 1966).

Já Hwang utiliza-se da árvore de Steiner com distância retilínea do trabalho de Hanan e sua definição de *Rectilinear Full Steiner Tree* (RFST), na qual todo terminal é folha e vice-versa, e todos outros são vértices para definir suas topologias (HWANG, 1976). Definindo então que a RSMT é a união de RFSTs e sempre existe uma RSMT na qual cada RFST que a compõe possui duas topologias de Hwang .

Com estas topologias Hwang define a melhor forma de construção de uma RSMT, portanto definindo a base do melhor algoritmo de geração de RSMT da atualidade, o GeoSteiner (ZACHARIASEN, 2001).

O trabalho feito por Silva traz a definição de muitos conceitos envolvidos com grade de Hanan, assim como aplicações que utilizam do RSMT e conclui com um comparativo entre as aplicações. Com um estudo abrangente, profundo e detalhado sobre o problema da árvore retilínea mínima de Steiner, concebe um conjunto com novas abordagens heurísticas e metaheurísticas para este problema (SILVA, 2009). Assim ele faz uma centralização em um único trabalho com as melhores características das mais bem sucedidas técnicas de RSMT encontradas na literatura, ele também trás a sugestão proposta por este trabalho da construção dinâmica pelas coordenadas dos pontos.

Diferentemente das obras citadas anteriormente, o presente trabalho realiza a construção de uma grade de Hanan dinâmica para aplicação em uma ferramenta de caminhos mínimos, buscando-se apontar melhorias em armazenamento de memória e tempo de execução em cons-

traste da construção completa em memória da grade de Hanan.

5 PROJETO DE IMPLEMENTAÇÃO

Neste capítulo encontra-se os passos que foram seguidos até o término deste trabalho de conclusão de curso, desde técnicas utilizadas, configuração do ambiente, base da dados e a modelo de implementação dos algoritmos propostos neste trabalho.

5.1 Configuração do Ambiente

Durante o desenvolvimento do projeto, utilizou-se um computador Dell Optiplex 7010 com as seguintes configurações, processador *Intel Core i7-3770 CPU 3.40GHz x 8*, 8GB de memória RAM, disco rígido de 500GB de capacidade, placa gráfica *Intel HD Graphics 2500* e sistema operacional *Ubuntu 16.04 LTS 64-bits*.

Para a implementação da grade de Hanan dinâmica e do algoritmo de Prim com a árvore de intervalo com o objetivo de encontrar um caminho mínimo na grade de Hanan Dinâmica proposta, usou-se a linguagem de programação Python 3.6.6 e PyPy 5.10.0 with GCC 8.0.1 20180324 (Red Hat 8.0.1-0.20).

5.2 Pré-Execução dos Dados de Validação

A base de dados para validação utilizada neste trabalho é a base de casos de teste do (COMPUTER AIDED DESIGN, 2017), disponibilizada para problema B denominado "*Net Open Location Finder with Obstacles*" feito por (SHEN; KAI-SHUN HU SYNOPSYS TAIWAN CO., 2017). Contando com casos de teste com até 8 camadas, sendo de dimensões entre o (0,0) e (500000,150000), disponibilizando 97146 componentes e 79012 obstáculos entre todas as camadas, utilizou-se dez entradas contendo 100 à 1000 objetos, sendo metade componentes e metade obstáculos. O primeiro arquivo de entrada se iniciou com 100 objetos somando-se 100 objetos a cada próximo arquivo até a décima entrada com 1000 objetos.

Utilizando-se dos casos de teste fornecidos pelo ICCAD foi necessário alterar o arquivo de entrada com os casos para teste, retirando as linhas com os valores de *ViaCost*, *MetalLayers*, *RoutedShapes*, *RoutedVias* e *Obstacles*, os quais não são usados para criar a grade de Hanan dinâmica, exemplificado na Figura 5.1.

```
ViaCost = 30
Spacing = 8
Boundary = (0,0) (40000,15000)
MetalLayers = 5
RoutedShapes = 4518
RoutedVias = 34
Obstacles = 4773
```

Figura 5.1 – Exemplo de Caso de Teste Sem Tratamento

Entretanto, os valores de *Spacing* que é o espaço mínimo de distância entre o obstáculo e pontos criados para a grade de Hanan e a variável *Boundary* que é o limite das coordenadas de todos os objetos contidos na grade de Hanan foram utilizados, visto na Figura 5.2.

```
Spacing = 8
Boundary = (0,0) (40000,15000)
```

Figura 5.2 – Exemplo de Caso de Teste Com Tratamento

5.3 Construção e Aplicação da Grade de Hanan Dinâmica

Nesta seção é apresentado a construção da grade de Hanan dinâmica seguindo a proposta de apenas armazenar as coordenadas únicas dos n objetos em ordem crescente. Posteriormente, aplicando-se o algoritmo de Dijkstra navegando sobre a grade de Hanan dinâmica armazenando os caminhos mínimos entre componentes e tendo o auxílio da árvore de intervalos implementada para identificar estes componentes.

5.3.1 Construção da Grade de Hanan Dinâmica

A construção da Grade de Hanan Dinâmica neste trabalho traz a melhoria em aspecto em utilização de durante a execução. Para isto, ao contrário de uma construção completa com complexidade O(n^2) em utilização de memória, utilizou-se a proposta contida no trabalho de Silva que consiste em armazenar as coordenadas x, y dos pontos iniciais de forma ordenada em vetores de elementos únicos x_Hanan e y_Hanan respectivamente (SILVA, 2009). Pode-se observar o que foi dito anteriormente no pseudocódigo no algoritmo 5.1 e a construção dos pontos de Hanan acontece ao longo da execução de um algoritmo de caminhos mínimos.

Como por exemplo a entrada de uma linha contém o Nome do objeto e as coordenadas, como *RoutedShape M.((20,30) (50,60))* ou *Obstacle M.((100,30) (100,60))*, onde as coorde-

nadas tem forma (x, y) e são armazenados $x_0, .., x_n$ em outra lista e $y_0, .., y_n$ em outra sem a repetição de elementos e posteriormente são ordenadas de forma crescente. A repetição de algum x ou y é desnecessária para a combinação das linhas para criar as intersecções e traz dificuldade na localização dos pontos nas listas durante o algoritmo de Dijkstra.

Algoritmo 5.1 – Pseudo Construção de Hanan Dinâmica

```
arquivo_entrada = open(dados)
componentes = []
y = set()
x = set()
while linha in arquivo_entrada:
    #Pontos nas linhas da forma
    #RoutedShape M. (x0,y0) (x1,y1)
    if 'RoutedeShape':
        componentes.append(x0.linha,y0.linha,
            x1.linha,y1.linha)
    if not(x0.linha in x):
        x.add(x0.linha)
    if not(x1.linha in y):
        y.add(y0.linha)
    if not(y0.linha in x):
        x.add(x1.linha)
    if not(y1.linha in y):
        y.add(y1.linha)
x_Hanan = list(x)
y_Hanan = list(y)
x_Hanan.sort()
y_Hanan.sort()
```

Para a execução do algoritmo de caminhos mínimos foi escolhido o algoritmo de Dijkstra com pequenas modificações no grafo que o mesmo utiliza para a fila de prioridade e a árvore de intervalos para determinar qual é a distância dos pontos para se adequar as características propostas no problema B do (COMPUTER AIDED DESIGN, 2017). Sabendo-se da ocorrência de arestas e pontos de Steiner com a característica de estarem dentro da área disposta para um determinado componente utilizou-se da árvore de Intervalos redigida por (CORMEN et al., 2009) para identificar esta característica. Sabendo-se que as arestas entre pontos de um mesmo componente tem custo zero de distância, assim também a navegação entre pontos no interior do componente tem este mesmo custo.

5.3.2 Árvore de Intervalos No Algoritmo de Dijkstra

A árvore de intervalos é usada para poder verificar pontos e distância da arestas entre pontos em um componente. Para implementar a árvore de intervalos seguiu-se a proposta de (CORMEN et al., 2009) em seu livro, apenas fazendo a modificação para uma árvore de intervalos bidimensional, ou seja, composta de dois intervalos de números reais [x_0, y_0] e [x_1, y_1] com seus níveis alternados entre par e ímpar, para classificação da árvore são usadas as coordenadas x_0 para níveis pares e y_0 para os níveis ímpares visto no Algoritmo 5.2 com a construção da árvore de intervalo, Algoritmo 5.3 criação dos nodos e Algoritmo 5.4 o balanceamento da árvore. Também foi preciso utilizar o máximo e mínimo dos intervalos de cada elemento e de suas sub-árvores.

Algoritmo 5.2 – Construção da Árvore de Intervalo pelos Componentes

```
def arvore_intervalo(componentes):
    raiz = []
    n_componente = 0
    for componente in componentes:
        if(n_componente==0):
            raiz = novo_nodo(nivel,componente,n_componente)
        else:
            constroi_arvore(raiz,componente,n_componente)
        n_componente = n_componente + 1
return raiz
```

Um exemplo seria a existência de dois componentes em sequência *RoutedShape M.* (20,30) (50,60) e *RoutedShape M1* (80,20) (120,30). Assim seguindo a função o componente $arvore\_intervalo$ recebe estes componentes, com *RoutedShape M.* (20,30) (50,60) sendo o primeiro e a raiz com o $maximo = (50, 60)$ e $minimo = (20, 30)$. O próximo nível é impar e será classificado pela coordenada y_0 do nodo atual que é a raiz, o próximo componente é o *RoutedShape M1* (80,20) (120,30), tal que $y_0 == 20$ é menor y_0 da raiz, ficando assim posicionado no filho da esquerda da raiz e como $(y_0 < minimo.y)$ a coordenada y_0 se torna novo $minimo.y$ e $(x_1 > maximo.x)$ a coordenada x_1 se torna o novo $maximo.x$. Adicionando-se um componente no próximo nível seu posicionamento é feito pelo valor de x_0.

Algoritmo 5.3 – Nodo para Árvore de Intervalos 2D

```
def novo_nodo(nivel,componente,n_componente):
    nodo.maximo = componente.superior
    nodo.minimo = componente.inferior
    nodo.xs = componente.xs
    nodo.ys = componente.ys
    nodo.anterior = nodo.proximo = []
    nodo.nivel = nivel
    nodo.numero = n_componente
    return nodo
```

Algoritmo 5.4 – Pseudo Balanceamento da Árvore de Intervalos 2D

```
def constroi_arvore(raiz,componente,n_componente):
    if componente.maximo>raiz.maximo:
        raiz.maximo = componente.maximo
    if componente.minimo<raiz.minimo:
        raiz.minimo = componente.minimo
    if raiz.nivel == 0:
        if componente.y < raiz.y:
            if raiz.anterior != []:
                novo_nodo(1,componente,n_componente)
            else:
                constroi_arvore(raiz.anterior,
                componente,n_componente)
        else:
            if raiz.proximo != []:
                novo_nodo(1,componente,n_componente)
            else:
                constroi_arvore(raiz.proximo,
                componente,n_componente)
    else:
        if componente.x < raiz.x:
            if raiz.anterior != []:
                novo_nodo(0,componente,n_componente)
            else:
                constroi_arvore(raiz.anterior,
                componente,n_componente)
        else:
            if raiz.proximo != []:
                novo_nodo(0,componente,n_componente)
            else:
                constroi_arvore(raiz.proximo,
                componente,n_componente)
```

Já na operação de busca de um ponto qualquer (x,y) na árvore de intervalos foi utilizada como base a função *Interval-Search* feita por (CORMEN et al., 2009), com a alteração das verificações de x_1 e y_1 máximos e de x_0 e y_0 mínimos de cada elemento conhece de si e de

suas sub-árvores e se as coordenadas x e y do ponto estão inclusas em um componente, isto é, $x_0 \leq x \leq x_1$ e $y_0 \leq y \leq y_1$. Também foi criado uma forma de numeração dos componentes para caso de uma pesquisa do ponto para ele mesmo retornar o número do componente que ele pertence, estas características podem ser vistas no pseudocódigo da árvore de intervalos 2D no algoritmo 5.5.

Algoritmo 5.5 – Pseudo Consulta na Árvore de Intervalos 2D

```
def Consulta_Intervalo(p1, p2):
    x = raiz
    while((x != []) and not (((x.x1 <=p1.x and x.x2 >=p1.x)
                    and (x.y1<=p1.y and x.y2>=p1.y))
                    and ((x.x1<=p2.x and x.x2>=p2.x)
                    and (x.y1<=p2.y and x.y2>=p2.y)))):

        esquerda = x.anterior
        if(esquerda!=[] and (esquerda.maximo.x>=p1.x
                        and esquerda.maximo.y>=p1.y)
                        and (esquerda.minimo.x<=p1.x
                        and esquerda.mini.y<=p1.y)):
            x = x.anterior
        else:
            x = x.proximo
    if(x != []):
        return x.numero
    else:
        return -1
```

Para uma consulta utilizando a função $Consulta\_Intervalo$ (Algoritmo 5.5) basta dois pontos ou o mesmo ponto para os parâmetros da função, esta que irá percorrer os ramos da árvore se existir um intervalo possível em algum dos ramos pelos seus máximos e mínimos. Por exemplo, a consulta do ponto (85,25) baseada na árvore de intervalo anterior, que em sua raiz não contém o ponto em seu intervalo, portanto é verificado se o máximo e mínimo do ponto da esquerda abrange o ponto consultado, ou seja, $minimo.x \leq ponto.x \leq maximo.x$ e $minimo.y \leq ponto.y \leq maximo.y$, ocorrendo isto o nodo da esquerda torna-se o raiz e é consultado se seu intervalo contém o ponto, que neste caso é confirmado assim retornando o número "2" que é o identificar do componente.

5.3.3 Algoritmo de Dijkstra Utilizando Grade de Hanan Dinâmica

Para encontrar os caminhos entre os componentes do problema foi utilizado o algoritmo de Dijkstra com a fila de prioridade mínima e o relaxamento (Algoritmo 5.6) (CORMEN et al.,

2009). Utilizando de um subgrafo S, mas como ao contrário do grafo completo G, usou-se as coordenadas das linhas verticais e horizontais ordenadas. Com isto foi possível montar os pontos pelas intersecções e tomar os valores das arestas pela distância das coordenadas dos pontos, construindo um subgrafo C onde ficam os caminhos mínimos de componentes.

Algoritmo 5.6 – Função Relax Definida por (CORMEN et al., 2009)

```
def Relax(u,v,w):
    if v.d > u.d + w(u,v)
        v.d = u.d + w(u,v)
        v.pi = u
```

A função $Vizinhos$ (Algoritmo 5.7) localiza-se a posição das coordenadas do ponto nos vetores de elementos únicos e ordenados, utiliza-se da posição para saber as coordenadas anteriores e posteriores nos vetores da grade de Hanan dinâmica e cria os pontos da grade com o custo da aresta pela função $Distancia$ e inicializa os vizinhos $v_0, ..., v_n$ com $v.d = \infty$ e $v.pi = Null$.

Algoritmo 5.7 – Função Vizinhos

```
def Vizinhos(Q, Ponto, x_Hanan, y_Hanan):
    index_x = x_Hanan,index(ponto.x)
    index_y = y_Hanan.index(ponto.y)

    superior = [ponto.x,linhas[index_y+1]]
    inferior = [ponto.x,linhas[index_y-1]]
    esquerdo = [x_Hanan[index_x-1],ponto.y]
    direito = [x_Hanan[index_x+1],ponto.y]

    Pontos_Vizinhos = superior,inferior, esquerdo, direito
    for P_Vizinho in Pontos_Vizinhos:
        P_Vizinho.w = Distancias_Pontos+
        Distancia(Ponto, P_Vizinho)
        P_Vizinho.d = Infinito
        P_Vizinho.pi = NUll
    Q = Q + (Pontos_Vizinhos)
    return Q
```

Um exemplo são as listas $x\_Hanan = [4, 8, 12]$ e $y\_Hanan = [5, 20, 120]$ e o $ponto = (8, 20)$, sabendo-se que a posição de seu x e y é a segunda posição das listas é possível saber saber a próxima posição e a anterior para construir seus pontos vizinhos. Após construídos é possível se tomar a distância e inicializar os pontos para adiciona-los na fila de prioridade.

Já a função $Distancia$ (Algoritmo 5.8) verifica utilizando-se da árvore de intervalos do ponto atual para outro qualquer dos seus vizinhos se ambos pertencem a um mesmo componente, caso o resultado for sim o custo até este outro ponto é zero, se não o custo é a dife-

rença das coordenadas. Está também inicializa os novos pontos v conhecidos com $v.d = \infty$ e $v.pi = Null$.

Algoritmo 5.8 – Função de Distância

```
def Distancia(Ponto_um, Ponto_dois):
    if (Consulta_Intervalo(Ponto_um, Ponto_dois) != 0):
        return 0
    else:
        return modulo(Ponto_um.x - Ponto_dois.x +
                      Ponto_um.y - Ponto_dois.y)
```

Para o Algoritmo 5.8 um exemplo são os $ponto\_um$ e $ponto\_dois$ com suas coordenadas $(4, 20)$ e $(8, 20)$ respectivamente, que caso os dois pontos pertençam a um mesmo componente da árvore de intervalo tem valor 0 no custo de sua aresta, pelo motivo de navegação de custo zero no mesmo componente ou o valor 4 na aresta que é a diferença em módulo das coordenadas dos pontos.

O algoritmo de $Dijkstra$ inicia-se pelo ponto inicial s de um componente escolhido qualquer com os valores de custo até o ponto $s.d = 0$ e $s.pi = Null$ sendo o ponto predecessor que é adicionado na fila de prioridade Q, no caminho mínimo S que será adicionado os pontos ao longo da execução e no grafo de caminhos mínimos entre os componentes de C.

Posteriormente, enquanto esta fila Q não estiver vazia, retira o ponto mínimo da mesma u, adiciona-se em S e verifica-se u se é o ponto de um novo componente não conhecido em C caso seja verdadeiro é adicionado no caminho mínimo C. Logo após, adiciona-se seus vizinhos $v_0, ..., v_n$ nesta fila pela função $Vizinhos$, que utiliza-se da função $Distancia$ para o cálculo dos pesos das arestas de (u, v_n).

Por fim, é realizado o relaxamento das arestas proposto por (CORMEN et al., 2009) para o caminho mínimo S que também é utilizado para o caminho mínimo entre componentes C. A execução acaba com o termino da fila de prioridade com o caminho mínimo S de todos os componentes que é usado pelo algoritmo Dijkstra (CORMEN et al., 2009). C é onde fica armazenado o caminho mínimo entre componentes iniciando-se por um determinado componente. A implementação pode ser vista no pseudocódigo de Dijkstra utilizando a grade de Hanan dinâmica no Algoritmo 5.9.

Algoritmo 5.9 – Pseudo Dijkstra Utilizando a Grade Hanan Dinâmica

```
from arvore_intervalo import Consulta_Intervalo

def Dijkstra(x_Hanan,y_Hanan, s):
    Ponto_inicial.x = s.x
    Ponto_inicial.y = s.y
    Ponto_inicial.d = 0
    Ponto_inicial.pi = Null
    S = Q = C = []
    Ponto_inicial.n = Consulta_Intervalo(u,u)
    C = C + Ponto_inicial
    Q = Q + u
    while Q != []:
        u = Min_Ponto(Q)
        S = S + u
        Componente = Consulta_Intervalo(u,u)
        if (Componente != -1 and not Componente in C.n):
            u.n = Componente
            C = C + u
        Q = Q + vizinhos(u,x_Hanan,y_Hanan)
        for cada ponto v de G.adjacente[u]:
            RelaxS(u,v,w)
            Consulta = Consulta_Intervalo(v,v)
            if(Consulta!=-1):
                Componente = (Componente.n == Consulta)
                RelaxC(u,Componente,w)
```

6 EXECUÇÃO E RESULTADOS

Neste capítulo apresenta-se como foi realizado o processo de execução dos testes sobre a grade de Hanan dinâmica construída feita através da técnica proposta nos capítulos anteriores, os resultados da execução da grade de Hanan dinâmica e sua aplicação em um algoritmo de caminhos mínimos, além de considerações finais sobre o trabalho.

6.1 Execução

Com o arquivo de entrada de casos de testes devidamente preparado realizou-se a execução do algoritmo desenvolvido em cima das definições do capítulo anterior. Foram executados dez casos de testes de tamanhos crescentes, assim formando várias grades de Hanan diferentes.

6.2 Resultados

Percebeu-se com n objetos sem coordenadas compartilhadas no circuito chega-se em torno de $4n^2$ vértices na grade de Hanan, com apenas $4n$ sendo vértices de objetos. Entretanto, pelo fato do algoritmo unir as coordenadas de todos em vetores de valores distintos e existir compartilhamento de coordenadas entre os objetos, o número de pontos da grade é menor que $4n^2$ demonstrado na Tabela 6.1.

Tabela 6.1 – Tabela de Pontos que Compõe a Grade de Hanan

Tamanho do Subcaso	**Número de Componentes**	**Número de Obstáculos**	**Número de pontos da Grade de Hanan Dinâmica**	**Número de pontos da Grade de Hanan Completa**
100	50	50	39.801	40.000
200	100	100	153.258	160.000
300	150	150	338.092	360.000
400	200	200	591.233	640.000
500	250	250	911.803	1.000.000
600	300	300	1.285.513	1.440.000
700	350	350	1.729.886	1.960.000
800	400	400	2.212.893	2.560.000
900	450	450	2.760.096	3.240.000
1000	500	500	3.347.224	4.000.000

Foram utilizados os trabalhos de (SILVA, 2009) e (HANAN, 1966) para se obter comparativos de construção da grade de Hanan completa em memória, ou seja, uma construção de

todos pontos armazenados em memória teria o custo de $4n^2$ inicialmente e para percorrer este grafo é necessário este mesmo custo.

Portanto, o principal objetivo foi reduzir este custo representado na Tabela 6.2 utilizando a técnica dinâmica proposta que esta no trabalho de (SILVA, 2009). Percebe-se que existe um custo muito baixo de armazenamento da grade de Hanan dinâmica até para o maior caso, diferentemente da criação e armazenamento da grade completa em memória que levaria cerca de $O(n^2)$ em tempo e espaço de memória.

Tabela 6.2 – Tabela de Memória usada pela Grade de Hanan Dinâmica

Tamanho do Subcaso	**Número de Componentes**	**Número de Obstáculos**	**Memória Utilizada pela Grade de Hanan Dinâmica**
100	50	50	9208
200	100	100	9280
300	150	150	9244
400	200	200	9156
500	250	250	9260
600	300	300	9260
700	350	350	9204
800	400	400	9292
900	450	450	9320
1000	500	500	9332

Já utilizando-se da grade de Hanan dinâmica aplicada em uma ferramenta de caminhos mínimos com o custo de $O(log\, n)$ para cada ponto de componente. Identifica-se a redução de memória utilizada pelo armazenamento de apenas $n-1$ arestas e no máximo $4n^2$ pontos no pior caso do caminho mínimo resultante, mas em todos os casos de validação houve a redução de número de pontos no caminho mínimo pelo fato do agrupamento das coordenadas e a redução de pontos, o algoritmo de caminho mínimo percorreu um grafo menor.

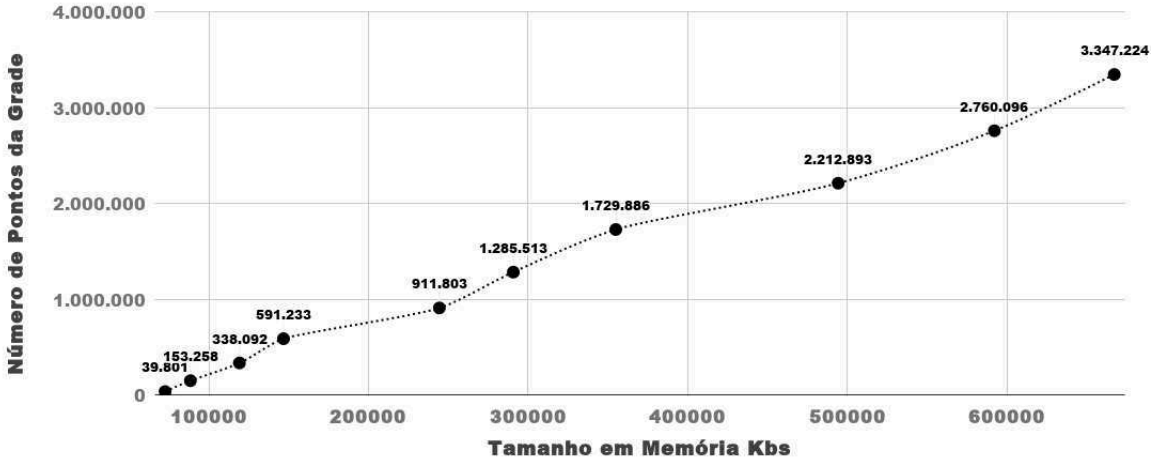

Figura 6.1 – Gráfico de Memória Máxima Utilizada na Execução do Algoritmo de Dijkstra de Um Ponto de Componente utilizando a Grade de Hanan Dinâmica com Pypy

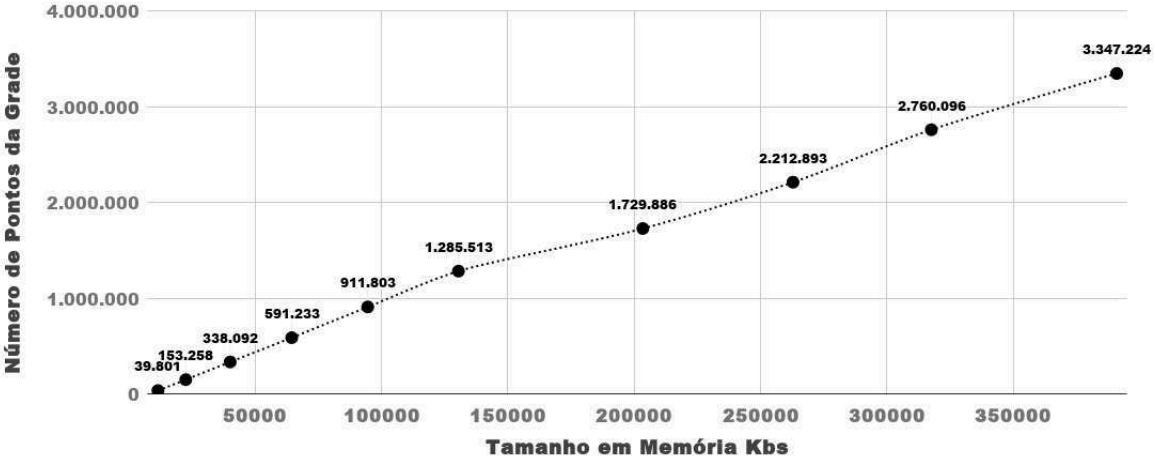

Figura 6.2 – Gráfico de Memória Máxima Utilizada na Execução do Algoritmo de Dijkstra de Um Ponto de Componente utilizando a Grade de Hanan Dinâmica com Python3

Se utilizando do gráfico da Figura 6.1 com Pypy e Figura 6.2 com Python3, observa-se que utilizando a grade de Hanan dinâmica nos algoritmos de Dijkstra junto da árvore de intervalos foi possível respeitar a complexidade $O(n\,log\,n)$ em utilização de memória proposta. Sendo assim, ainda mantendo-se em uma complexidade $O(n^2)$ em utilização de memória da grade de Hanan completa.

Já nas Figura 6.3 e Figura 6.4 é possível observar o tempo de execução, onde é respeitado a redução da complexidade de $O(n^2)$ em tempo, que é um adicional ao trabalho proposto de redução em memória da construção da grade de Hanan para uma grade de Hanan dinâmica e sua aplicação em um algoritmo de caminhos mínimos para demonstrar sua possível aplicação.

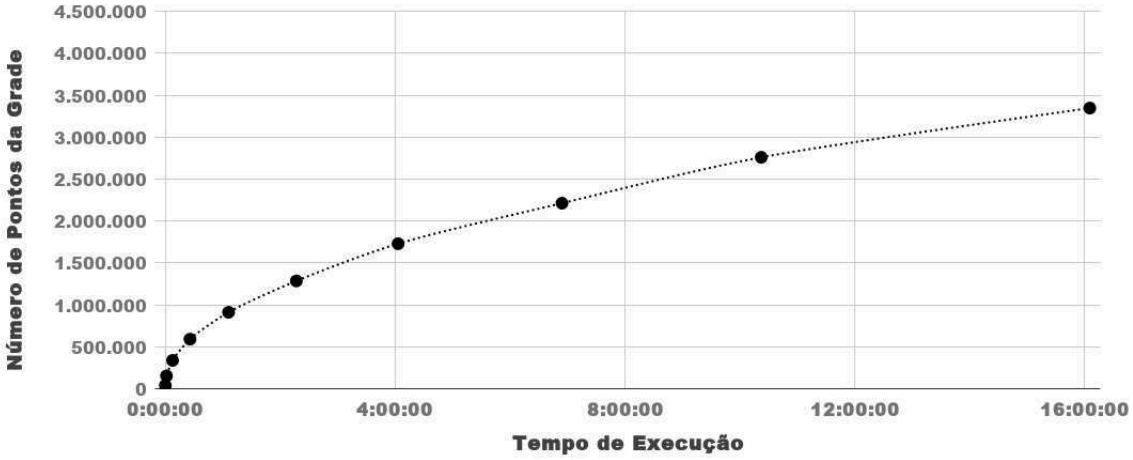

Figura 6.3 – Gráfico de Tempo de Execução Máximo na Execução do Algoritmo de Dijkstra de Um Ponto de Componente utilizando a Grade de Hanan Dinâmica com Pypy

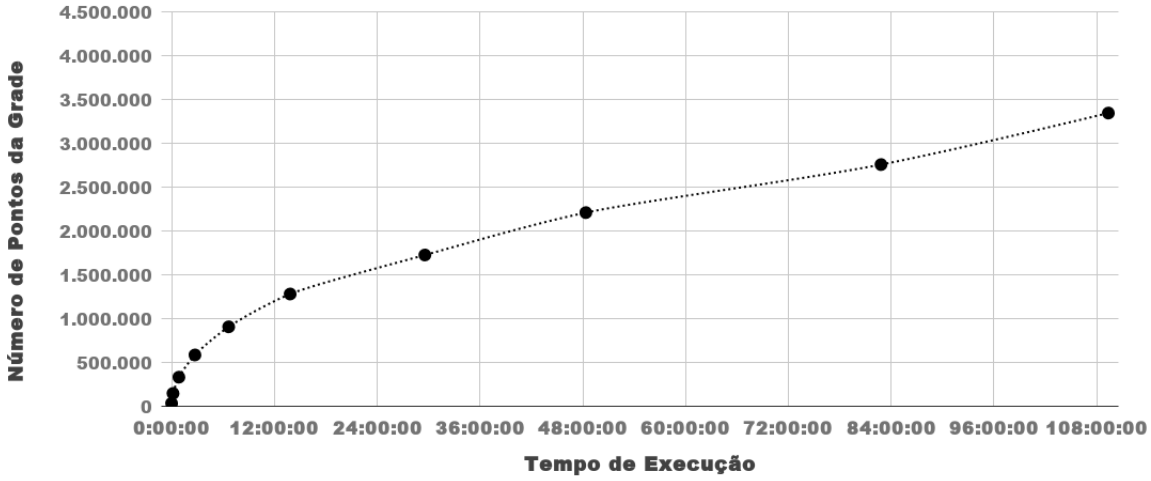

Figura 6.4 – Gráfico de Tempo de Execução Máximo na Execução do Algoritmo de Dijkstra de Um Ponto de Componente utilizando a Grade de Hanan Dinâmica com Python3

Enfim, a aplicação utilizando-se do Pypy e do Python3 são parecidas em complexidade, mas atuam de forma diferente. Obteve-se um ganho em tempo de execução pelo Pypy comparado ao Python3, mas um decréscimo em utilização memória. É possível ver as tabelas mais detalhadas dos resultados vistos nos gráficos no Apêndice A.

6.3 Considerações Finais

Por fim, através dos resultados vistos nas Tabelas 6.1 e Tabela 6.2 da grade de Hanan dinâmica construída foi possível analisar seu comportamento em memória e pontos gerados. Observou-se um bom desempenho geral e obteve-se um bom resultado de redução na construção

da grade de Hanan anterior a sua aplicação em uma ferramenta de complexidade $O(n^2)$ em memória para a construção da grade de Hanan dinâmica que aproximadamente em $O(n)$, onde n é o número de objetos de entrada. Tais afirmações são baseadas nas métricas aplicadas e apresentadas nos gráficos com seus resultados.

Com base nos resultados apresentados, constata-se que, dentre os valores de teste analisados com n entre 100 e 1000, as execuções demonstraram que é possível a aplicação de uma grade de Hanan dinâmica em um algoritmo de caminhos mínimos adequado para a proposta feita no problema do (COMPUTER AIDED DESIGN, 2017) utilizando-se de uma árvore de intervalo. Para tal afirmação, é levando em conta a complexidade das execuções com os dos algoritmos de Dijkstra utilizando-se da árvore de intervalo com o teorema 12.4 que estão redigidos no livro de (CORMEN et al., 2009).

7 CONCLUSÃO

Neste trabalho de conclusão de curso, foi realizado a construção da grade de Hanan dinâmica sobre a proposta do problema B do (COMPUTER AIDED DESIGN, 2017). Para viabilizar a implementação do trabalho foi necessário o pré-processamento dos dados de validação, construção da grade de Hanan dinâmica, aplicação da grade no algoritmo de Dijkstra e a implementação de uma árvore de intervalos.

Para avaliar a execução da implementação da grade de Hanan dinâmica, aplicou-se a grade construída em um algoritmo de caminhos mínimos utilizando-se do tempo de execução e do tamanho da memória máxima como parâmetros de avaliação de comportamento, resultados que estão apresentados no capítulo anterior. Assim foi observado o comportamento do algoritmo implementado proveniente dos casos de testes especificados no projeto de construção da grade de Hanan dinâmica.

Além da implementação da grade de Hanan dinâmica, implementou-se um algoritmo de Dijkstra e uma árvore de intervalos para melhor avaliação dentro da proposta do problema proposto por (COMPUTER AIDED DESIGN, 2017), submetendo na aplicação os dados de teste mensurando-se o tempo, espaço de memória alocado e número de pontos existentes na grade de Hanan.

Assim foi possível realizar comparações entre os dados obtidos através de cada caso, identificando que a grade de Hanan dinâmica diminui o número de alocação de memória e de tempo em execução comparado ao esperado em uma alocação primária em memória da grade de Hanan completa. Posteriormente, aplicando-se sobre ela o algoritmo de Dijkstra, obteve-se resultados de diminuição no consumo de memória e de tempo de execução mesmo em piores casos.

7.1 Trabalhos Futuros

Afim de abranger o estudo e a construção da grade de Hanan dinâmica aplicada em um algoritmo de caminhos mínimos, é possível a realização trabalhos futuros com mais estudos sobre como reduzir em tamanho de memória e processamento da árvore mínima resultante dos caminhos mínimos feitos sobre a grade de Hanan dinâmica, assim como uma forma de verificação de pontos e arestas em obstáculos eficiente.

Além disso, pode-se elaborar uma avaliação sobre os resultados obtidos por métricas

comparadas com a geração completa da grade de Hanan, buscando-se mensurar o crescimento da grade e como também conseguir minimizar o custo de armazenamento em memória do algoritmo que gere a árvore mínima.

REFERÊNCIAS

COMPUTER AIDED DESIGN, I. I. C. on. **The 2017 CAD Contest at ICCAD**. Accessed: 2018-03-01, http://cad-contest-2017.el.cycu.edu.tw/CAD-contest-at-ICCAD2017/.

CORMEN, T. H. et al. **Introduction to Algorithms Third Edition**. [S.l.]: MIT Press, 2009.

ESMAEILZADEH, H. et al. Dark Silicon and the End of Multicore Scaling. **International Symposium on Computer Architecture**, [S.l.], v.38, p.365–376, 2011.

HANAN, M. On Steiner's problem with rectilinear distances. **SIAM, Applied Math**, [S.l.], v.30, n.1, p.255–265, 1966.

HWANG, F. K. Steiner Minimal Trees with Rectilinear Distance. **SIAM Journal of Applied Mathematics**, [S.l.], v.30, n.1, p.104–114, 1976.

LU, C. L.; TANG, C. Y.; LEE, R. C.-T. The full Steiner tree problem. **Theoretical Computer Science**, [S.l.], v.306, p.55–67, 2003.

SHEN, C.; KAI-SHUN HU SYNOPSYS TAIWAN CO., L. **ICCAD 2017 Contest Net Open Location Finder with Obstacles**. Accessed: 2018-03-01, http://cad-contest-2017.el.cycu.edu.tw/Problem_B/default.html.

SILVA, T. Gouveia da. **Métodos Heurísticos Aplicados ao Problema da Árvore de Steiner Rectilinear**. 2009. Dissertação (Mestrado em Ciência da Computação) — Centro de Ciências Exatas e da Natureza da Universidade Federal da Paraíba.

ZACHARIASEN, M. A Catalog of Hanan Grid Problems. **NETWORKS**, [S.l.], v.38, n.2, p.76–83, 2001.

APÊNDICES

APÊNDICE A – Apêndice A - Tabelas com Resultados de Execução

Tabela A.1 – Tabela Com Resultados De Execução Utilizando Pypy

Número de Pontos da Grade de Hanan Dinâmica	**Memória Usada em kbytes**	**Tempo de Execução (h/m/s)**
39.801	72648	00:00:06
153.258	88508	00:01:18
338.092	119228	00:07:38
591.233	146856	00:25:48
911.803	244464	01:06:08
1.285.513	290804	02:16:52
1.729.886	354988	04:03:05
2.212.893	494304	06:54:12
2.760.096	592168	10:22:14
3.347.244	667268	16:05:16

Tabela A.2 – Tabela Com Resultados De Execução Utilizando Python3

Número de Pontos da Grade de Hanan Dinâmica	**Memória Usada em kbytes**	**Tempo de Execução (h/m/s)**
39.801	16520	00:00:35
153.258	29624	00:10:03
338.092	49592	00:58:17
591.233	78304	3:05:12
911.803	112980	7:49:07
1.285.513	155364	16:01:22
1.729.886	203424	29:34:22
2.212.893	262832	48:21:43
2.760.096	317572	82:49:13
3.347.244	366048	109:21:29