# *Optical Graph Recognition*

Reconhecimento da Estrutura Topológica de um Grafo Através de sua Representação Ótica

Alexsander Andrade de Melo*

*PESC - COPPE*
*Universidade Federal do Rio de Janeiro*

Rio de Janeiro - RJ, Junho de 2015

**Resumo**

Este trabalho tem por objetivo a implementação de um método para reconhecimento da estrutura topológica de um grafo a partir de sua representação ótica (seja por desenhos feitos a mão, e posteriormente digitalizados, ou por desenhos feitos diretamente de maneira digital através de um *software*). Para tanto, nos baseamos no trabalho de Auer. C., et al. [1], que se utiliza de técnicas morfológicas de processamento de imagens para tal fim.

# 1 Introdução

Caminhando na direção inversa da área de *graph drawing*, que objetiva fornecer a "melhor" representação gráfica para um grafo, temos por interesse neste trabalho reconhecer a estrutura topológica de um grafo através de sua representação ótica[1]. Isto é, dada uma imagem de um grafo, conforme a Figura 1 ilustra, desejamos reconhecer algumas propriedades topológicas do

*aamelo@cos.ufrj.br

[1]Quando nos referimos a representação ótica de um grafo, estamos nos referindo a sua representação gráfica usual: regiões, geralmente circulares, representam vértices e curvas (ou retas) conectando dois vértices representam arestas (ou arcos, no caso de grafos direcionados).

grafo dado. Mais especificamente, neste trabalho nos atemos em reconhecer os vértices do grafo e suas respectivas conexões.

Como motivação para este trabalho temos que, em diversas situações são realizados desenhos de grafos através do uso de papel e lápis, e posteriormente tais desenhos são refeitos digitalmente, utilizando-se algum software (por exemplo: *Inkscape*, *Tikz/PGF*), para compor livros, artigos, apresentações, notas de aula, etc. Então seria de interesse a existência de alguma ferramenta que fizesse essa transposição de uma imagem feita a mão de um grafo, para um formato digital (possivelmente, vetorizado), de forma que algumas imperfeições comuns à desenhos realizados manualmente fossem corrigidas, ou descartadas. Uma outra aplicação para este trabalho seria a facilitação no fornecimento da entrada de algoritmos de grafos, após ser realizado o reconhecimento do grafo representado numa imagem. Por exemplo, suponha que tenhamos em mãos o desenho de um grafo (feito digitalmente ou não), e que desejamos verificar se este grafo referente ao desenho satisfaz alguma propriedade. Então poderíamos, primeiramente, reconhecer a estrutura topológica deste grafo a partir da imagem de entrada, e por fim executar o algoritmo que realiza tal verificação passando como entrada o resultado do reconhecimento topológico.

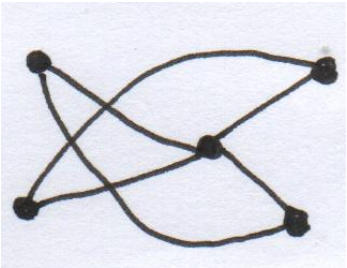


**Figura 1:** Representação ótica de um grafo.

*Observação* 1. O presente trabalho se baseia no trabalho de Auer, C., et al. [1]. No entanto, realizamos algumas pequenas modificações da abordagem proposta pelos autores, principalmente no que concerne à fase de Reconhecimento de Topologia, que será devidamente descrita na Seção 3.3.

## 1.1 Organização do trabalho

Este trabalho é organizado como segue: na Seção 2 descrevemos as suposições que são feitas com relação aos tipos de grafos que consideramos e suas respectivas representações gráficas dadas como entrada, além de expormos, em linhas gerais, as principais técnicas de processamento de imagens utilizadas no desenvolvimento do trabalho. Na Seção 3 descrevemos

o processo de reconhecimento de grafos em si, onde em cada uma das suas subseções detalhamos uma etapa específica desse processo. Na Seção 4 apresentamos alguns resultados preliminares. Por fim, na Seção 5 são descritas as conclusões e trabalhos futuros.

# 2 Preliminares

Neste trabalho supomos que os grafos (ou multigrafos) a serem reconhecidos são não direcionados, sem rótulos, sem pesos nas arestas e sem propriedades relacionadas à cor de arestas e de vértices[2]. Ademais, supomos também que a representação gráfica dos vértices dos grafos são dadas por formas (geralmente circulares, mas também podendo assumir outras formas, como retangulares, triangulares, etc.) com preenchimento, e também assumimos que os "raios" das representações gráficas de cada vértice possuem tamanhos similares entre si, e são consideravelmente maiores do que as espessuras da representação gráfica de cada aresta. A Figura 1 ilustra o tipo de representação que assumimos.

As principais operações que são realizadas na imagem de entrada são operações morfológicas binárias, mais especificamente utilizamos as operações *erosão*, *dilatação*, *abertura*, *fechamento* e *esqueletização*. Seja $A$ uma imagem binária, $B$ um elemento estruturante e $(B)_z$ a translação do elemento estruturante $B$ num ponto $z = (z_1, z_2)$, isto é,

$$(B)_z = \{b + z \,|\, b \in B\},$$

temos que as operações morfológicas binárias listadas anteriormente são definidas conforme segue:

- Erosão:
$$A \ominus B = \{z \,|\, (B)_z \subseteq A\}$$
- Diltação:
$$A \oplus B = \{z \,|\, (B)_z \cap A \neq \varnothing\}$$
- Abertura:
$$A \circ B = (A \ominus B) \oplus B$$
- Fechamento:
$$A \bullet B = (A \oplus B) \ominus B$$

[2]Algumas dessas propriedades podem ser consideradas sem muitas dificuldades num trabalho futuro.

- Esqueletização: tem por objetivo preservar as informações topológicas dos objetos da imagem enquanto os reduz para a espessura de um único pixel (quando possível) [3] (para saber mais, veja [4]).

Um outro conceito amplamente utilizado neste trabalho é o de 8-*vizinhança* de um pixel. Seja $p = (x, y)$ um pixel de uma imagem, a 8-*vizinhança* de $p$ é definida como o conjunto de pixels

$$N_8(x, y) = \{(x \pm 1, y \pm 1), (x \pm 1, y), (x, y \pm 1)\}$$

da mesma imagem. A matriz abaixo ilustra o posicionamento dos pixels que pertencem a 8-vizinhança de um pixel $(x, y)$:

$$\begin{bmatrix} (x-1, y-1) & (x-1, y) & (x-1, y+1) \\ (x, y-1) & (\mathbf{x}, \mathbf{y}) & (x, y+1) \\ (x+1, y-1) & (x+1, y) & (x+1, y+1) \end{bmatrix}.$$

Para saber mais sobre esses conceitos de operações morfológicas em imagens binárias, e de vizinhança, veja [4].

# 3 Metodologia

Como em [1], o processo de reconhecimento das propriedades topológicas do grafo é divido em quatro fases, a saber:

- Pré-processamento
- Segmentação
- Reconhecimento topológico
- Pós-processamento

Ao longo desta seção estaremos detalhando cada uma dessas fases.

## 3.1 Fase de Pré-processamento

Nesta fase de pré-processamento temos como:

- **Entrada:** matriz de pixels referente à imagem original.
- **Propósito:** separar os pixels de fundo (ou de ruído) dos pixels que de fato representam o grafo.

Para tanto, são realizados os seguintes procedimentos:

1. Aplicação do filtro gaussiano para remoção de ruídos;
2. Binarização da imagem (utilizando o algoritmo de Otsu, veja [4], para obter um limiar mais adequado);
3. Aplicação de abertura e fechamento para remover ruídos, e também para "reparar"algumas (pequenas) más formações no desenho do grafo. Por exemplo, o problema de algumas arestas não "tocarem" seus respectivos vértices pode ser resolvido com a aplicação de fechamento. Por outro lado, pequenos ruídos na imagem podem ser removidos ao aplicar abertura, como pixels isolados na imagem ou algumas saliências não desejadas.

A Figura 2 exibe o resultado da fase de pré-processamento aplicado à imagem da Figura 1.

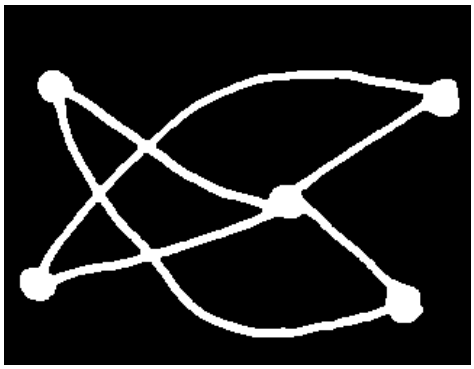


**Figura 2:** Resultado da fase de pré-processamento.

## 3.2 Fase de Segmentação

Nesta fase de segmentação temos como:

- **Entrada:** matriz de pixels resultante da fase de pré-processamento.
- **Propósito:** separar os pixels dos vértices dos pixels das arestas.
- **Saída:** matriz de pixels de vértices.

Para tanto, são realizados os seguintes procedimentos:

1. São aplicadas $k$ erosões na imagem de entrada

2. São aplicadas $k$ dilatações na imagem resultante do passo anterior, para que os vértices voltem aos seus tamanhos originais.

*Observação* 2. A quantidade $k$ de erosões e dilatações aplicadas depende da imagem de entrada. Mais precisamente, depende da espessura das arestas da imagem e do "raio" dos vértices. Nos resultados apresentados neste trabalho, utilizamos $k = 5$. Porém, em [1], sugere-se utilizar o algoritmo *distance image* de Chamfer [2] para se obter o valor de $k$ automaticamente. Deixamos a implementação deste algoritmo como trabalho futuro.

A Figura 3 exibe o resultado da fase de segmentação aplicado à imagem da Figura 2. Observe que os pixels referentes às arestas não se fazem mais presentes, restando apenas os pixels referentes aos vértices.

**Figura 3:** Resultado da fase de segmentação.

## 3.3 Fase de Reconhecimento de Topologia.

Nesta fase de reconhecimento de topologia temos como:

- **Entrada:** matriz de pixels resultante da fase de pré-processamento, $M_1$, e matriz de pixels de vértices resultante da fase de segmentação, $M_2$.
- **Propósito:** realizar uma esqueletização da imagem da matriz $M_1$, classificar os pixels do esqueleto como *edge pixels*, *port pixels*, *crossing pixels* e *miscellaneous pixels*, e por fim identificar as seções de arestas tratando os possíveis cruzamentos de arestas.
- **Saída:** matriz com os pixels classificados e seções de arestas que representam as arestas do grafo.

Para tanto, esta fase é dividida em quatro subfases, as quais são listadas abaixo:

- Subfase de esqueletização.
- Subfase de classificação dos pixels do esqueleto.
- Subfase de identificação de seções de arestas.
- Subfase de junção de seções resultantes do cruzamento de arestas.

### 3.3.1 Esqueletização

Neste trabalho foi utilizado o algoritmo de Zhang e Suen [7] para obtenção do esqueleto da imagem[3], aplicando-se posteriormente um algoritmo de $m$-conectividade (veja [4]) para que a espessura do esqueleto fosse a mínima possível. A Figura 4 exibe o resultado da esqueletização da imagem da Figura 2.

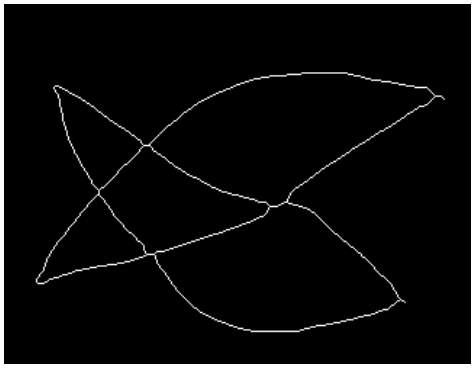


**Figura 4:** Resultado da subfase de esqueletização.

### 3.3.2 Classificação dos pixels do esqueleto

A classificação dos pixels do esqueleto é baseado no conceito de 8-vizinhança, visto na Seção 2. Seja $S$ a matriz de pixels referente ao esqueleto da imagem e $N_8^*(x, y)$ o conjunto de vizinhos do pixel $(x, y)$ tais que

$$(i, j) \in N_8^*(x, y) \Leftrightarrow (i, j) \in N_8(x, y) \text{ e } S(i, j) = 1.$$

Desta forma, os pixels do esqueleto são classificados de acordo com os seguintes critérios:

[3]Em [1], os autores utilizam afinamento morfológico para obter o esqueleto da imagem.

- Se $|N_8^*(x,y)| \leq 1$, então o pixel $(x,y)$ é classificado como um *miscellaneous pixels*.
- Se $|N_8^*(x,y)| = 2$ e $(x,y)$ possui um pixel relacionado à vértice em $N_8^*(x,y)$, então o pixel $(x,y)$ é classificado como um *port pixel*.
- Se $|N_8^*(x,y)| = 2$ e $(x,y)$ não possui pixel de relacionado à vértice em $N_8^*(x,y)$, então o pixel $(x,y)$ é classificado como um *edge pixel*.
- Se $|N_8^*(x,y)| > 2$, então o pixel $(x,y)$ é classificado como um *crossing pixel*.

A Figura 5 exibe a classificação dos pixels do esqueleto da Figura 4. Os pixels em vermelho representam os *port pixels*, os pixels em azul representam os *edge pixels* e os pixels em verde representam os *crossing pixels*.

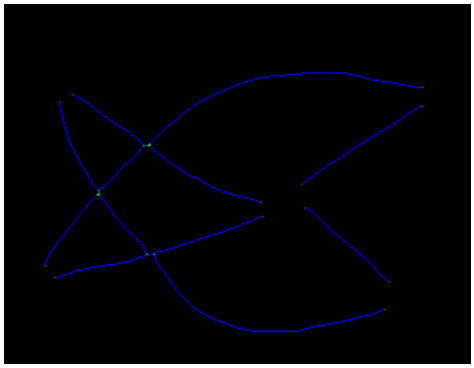


**Figura 5:** Resultado da subfase de classificação de pixels do esqueleto.

### 3.3.3 Identificação de seções das arestas

Seções de arestas que possuem um *port pixel* como origem e destino são facilmente identificadas, basta seguir os pixels da vizinhança que não são *miscellaneous pixels*. Note que tais seções representam diretamente arestas do grafo, já que suas duas extremidades estão ligadas a pixels referentes a vértices. Estas seções são chamadas de *trivial sections*.

Por outro lado, embora seções de arestas que possuem um *port pixel* como origem e um *crossing pixel* como destino também sejam facilmente identificadas (de forma semelhante a feita para *trivial sections*), tais não representam diretamente uma aresta, já que uma de suas extremidades não está ligada a nenhum pixel referente a vértice. Estas seções são chamadas de *port sections*.

Por fim, as seções que possuem *crossing pixels* como origem e destino serão identificadas na subfase seguinte, já que a identificação de tais é um pouco mais complexa do que as das *trivial sections* e *port sections*. Estas seções são chamadas de *crossing sections*.

### 3.3.4 Junção de seções

Nesta subfase temos por objetivo identificar *crossing sections* e juntá-las a duas *port sections* distintas, de forma que o resultado dessa junção represente uma aresta do grafo. No entanto, note que esta tarefa por vezes é complexa, pois nem sempre é tão claro qual direção deve ser seguida, isto é, quais combinações de *port sections* devem ser selecionadas para serem juntadas a uma dada *crossing section*. Desta forma, o que realmente realizamos nesta subfase não é a identificação de *crossing sections*, para então juntá-las a *port sections*, mas sim para cada *port section* selecionamos uma possível *crossing section* (dada pela vizinhança do pixel que é um *crossing pixel* da *port section* selecionada), e então procuramos seguir os pixels que estão na direção mais semelhante às direções anteriormente seguidas quando estávamos identificando a *port section* selecionada. Descrevemos esse processo abaixo:

1. Iniciamos de um *crossing pixel* $p = (x, y)$ de uma *port section*, e definimos o próximo pixel a ser seguido dentre seus vizinhos $N_8^*(x, y)$. Digamos, por exemplo, que o pixel selecionado para ser seguido seja o $q = (x', y') \in N_8^*(x, y)$. Para selecionarmos $q$ a partir de $p$, nos baseamos nos vetores direcionais $\vec{\alpha}_i$ que foram utilizados para identificar a *port section* que contém $p$. Em nossa implementação, priorizamos a direção dos vetores direcionais mais frequentes, sendo estes armazenados numa lista circular de tamanho $M$[4] que, se cheia, a cada inserção o mais novo elemento sobrescrevia o mais antigo.

2. Se nenhum desses vetores direcionais $\vec{\alpha}_i$ levam a uma direção válida (isto é, uma direção que não seja para um *miscellaneous pixel*), então utilizamos o vetor direcional que menos dista da média ponderada por frequência dos vetores direcionais $\vec{\alpha}_i$. Em nossa implementação, esta noção de distância é dada pela distância Euclidiana com peso nos eixos, isto é,
$$d = \sqrt{\alpha(x - x_0)^2 + \beta(y - y_0)^2},$$
onde os valores de $\alpha$, $\beta \in \mathbb{R}$ são definidos conforme o quão comum é variar a direção do eixo $x$ e do eixo $y$, respectivamente, baseado nos vetores direcionais $\vec{\alpha}_i$. Por exemplo, se é pouco comum variar a direção

[4]Em nossos testes utilizamos $M = 25$.

do eixo $x$, então o valor atribuído a $\alpha$ será consideravelmente maior do que o atribuído a $\beta$.

3. Repetimos esse processo até se chegar num *crossing pixel* de uma *port section* diferente da de partida.

No término deste processo, juntamos a *port section* de início com a *crossing section* formada pelos pixels selecionados no processo (em ordem), juntamente com a *port section* localizada na última etapa, e por fim marcamos o *crossing pixel* desta última *port section* para que a mesma não seja considera numa próxima iteração de junção de arestas como ponto de partida, visto que a aresta correspondente a essa seção já fora identificada. Observe que se as direções escolhidas forem sempre as corretas, teremos identificados todas as seções que levam à arestas do grafo. No entanto, é possível que algumas direções sejam escolhidas erroneamente.

### 3.4 Fase de Pós-processamento

Nesta fase, basicamente, extraímos as componentes conexas referentes aos vértices, a partir matriz de pixel de vértices obtida na fase de segmentação. Para tanto utilizamos um algoritmo baseado em busca em largura, que atribui rótulos para cada componente conexa (conjunto de pixels que formam uma região de 8-vizinhança) encontrada. Com isso, podemos definir explicitamente os vértices e as arestas do grafo, armazenando-os numa estrutura de dados mais adequada. Ademais, atribuímos coordenadas aos vértices, e às arestas, para uma melhor visualização dos resultados obtidos[5].

Enfim, a Figura 6 exibe a plotagem do grafo resultante da nossa implementação de reconhecimento topológico para a imagem da Figura 1.

## 4 Resultados

Nesta seção expomos os resultados de cada fase aqui descrita, para alguns grafos utilizados em nossos testes.

[5]Embora tenhamos atribuído coordenadas às arestas com o intuito de representar suas curvas, os resultados a serem apresentados não as denotam, pois não foi localizado um *software* para plotagem dos grafos que considerasse tais coordenadas (ou ao menos não era conhecido como se fazer isso).

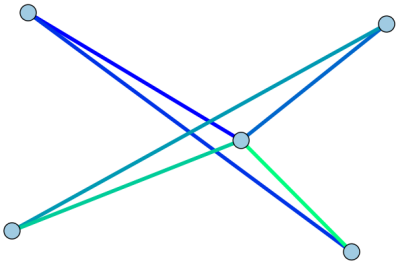


**Figura 6:** Resultado final de reconhecimento do grafo.

## 4.1 Resultados Bem Sucedidos

O reconhecimento dos grafos referentes às Figuras 7-11 não possui muitas complicações, já que não há cruzamento de arestas. Obtemos portanto resultados bem sucedidos para essas imagens.

No entanto, o reconhecimento dos grafos referentes às Figura 12-18 já é um pouco mais complicado, visto que todos esses possuem cruzamento de arestas, e alguns, inclusive, mais de um cruzamento. Para tais grafos se faz, então, necessário a execução da subfase de junção de arestas. Porém ainda assim, todos os resultados para essas imagens foram bem sucedidos, reconhecendo os respectivos grafos corretamente.

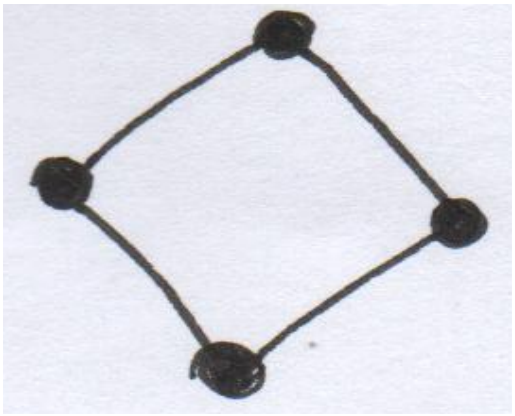
## 4.2 Resultados Mal Sucedidos

Alguns dos resultados que obtivemos foram mal sucedidos, e tais insucessos são devidos principalmente a dois possíveis tipos de erro que a nossa implementação está sujeita:

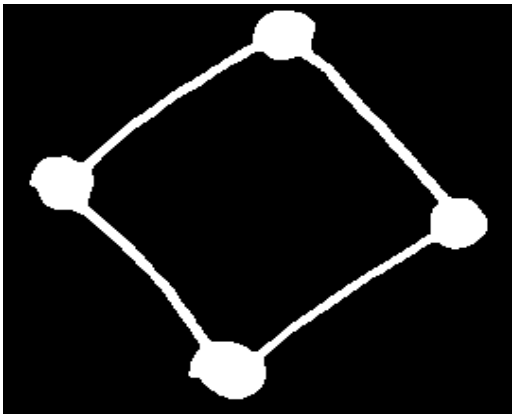
1. a quantidade $k$ de erosões e dilatações realizadas na fase de segmentação escolhida não é a ideal para isolar os pixels referentes aos vértices dos pixels referentes à arestas, e

2. o vetor direcional escolhido na subfase de junção de arestas leva a uma seção de aresta, e por fim a uma aresta, inexistente no grafo original.

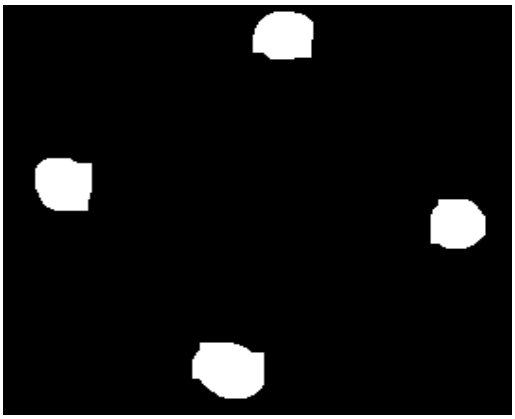
Os erros do primeiro tipo são mais fáceis de serem corrigidos, bastando implementar o algoritmo *distance image*, como feito em [2], para que o valor de $k$ seja escolhido de acordo com a imagem de entrada. Já os erros do segundo tipo são um pouco mais complicados de serem corrigidos, e depende



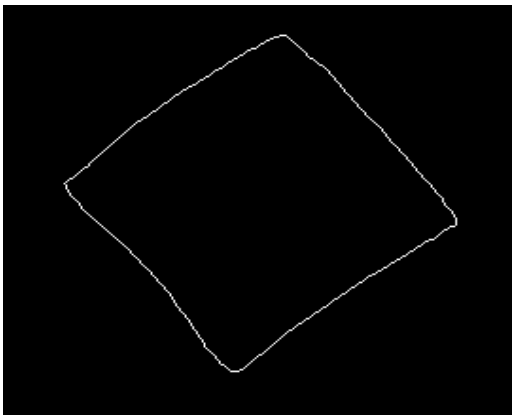
**(a)** Imagem de entrada.



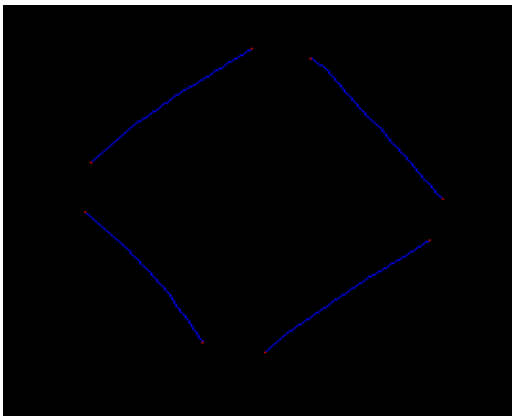
**(b)** Resultado pré-processamento.



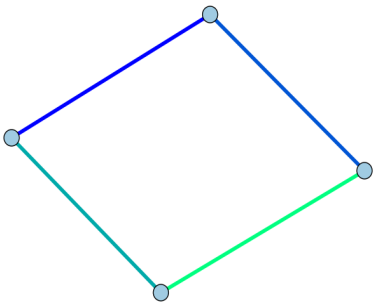
**(c)** Resultado segmentação.



**(d)** Resultado esqueletização.

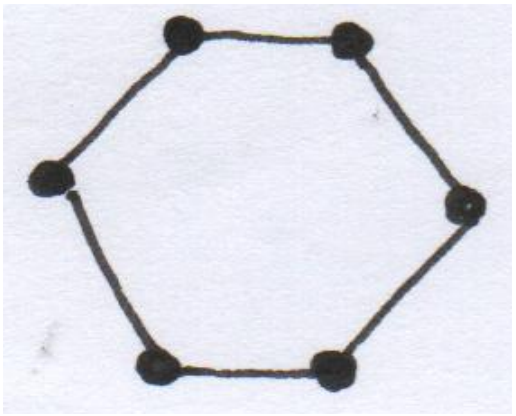


**(e)** Resultado classificação de pixels.

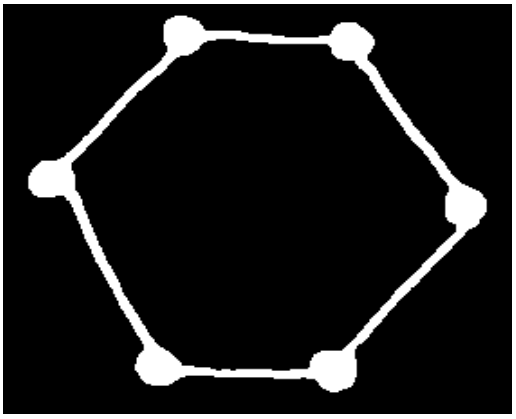


**(f)** Resultado final.

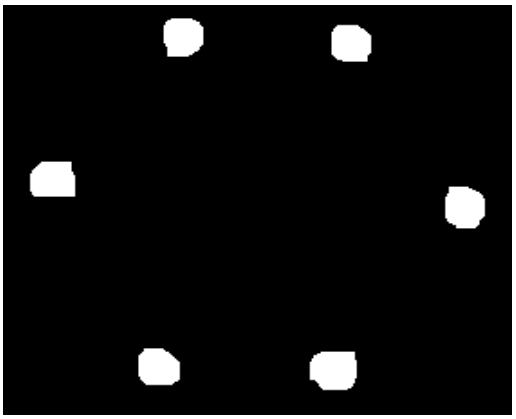
**Figura 7:** Grafo 2.



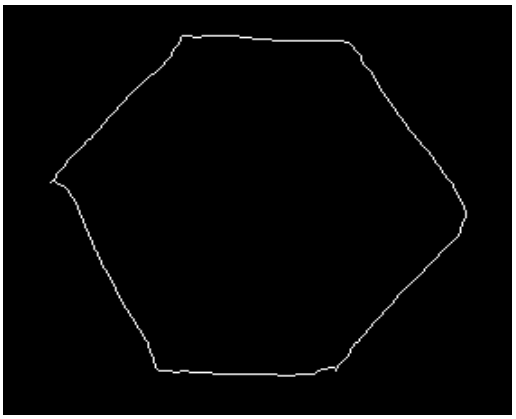
**(a)** Imagem de entrada.



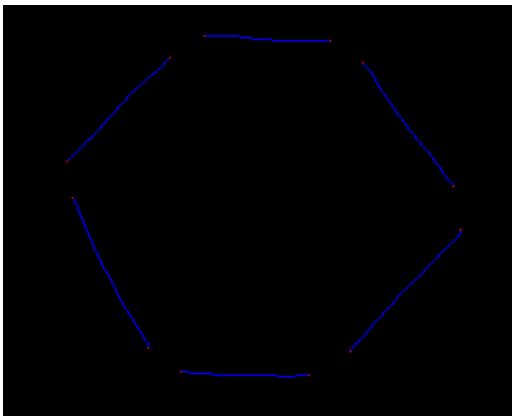
**(b)** Resultado pré-processamento.



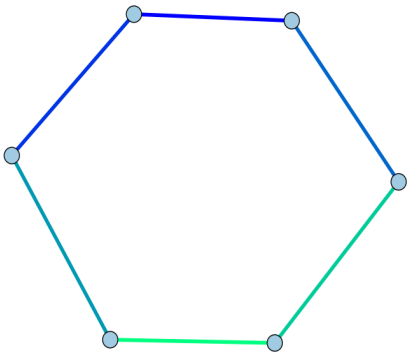
**(c)** Resultado segmentação.



**(d)** Resultado esqueletização.

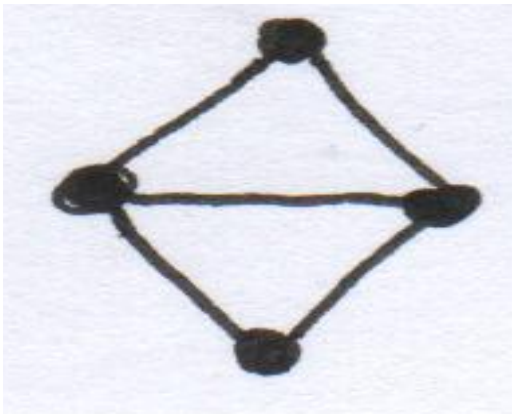


**(e)** Resultado classificação de pixels.

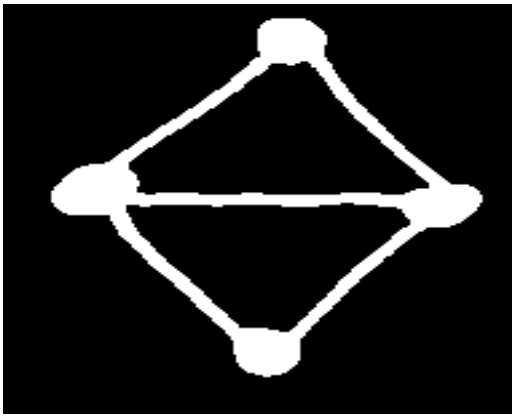


**(f)** Resultado final.

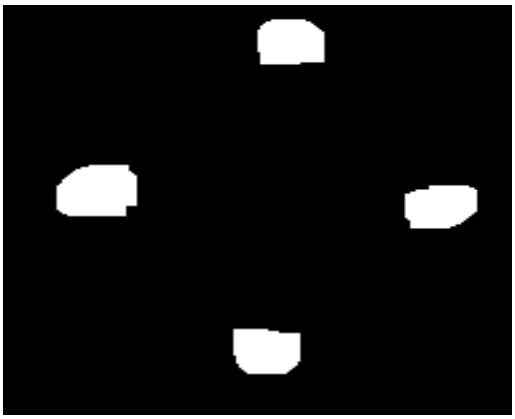
**Figura 8:** Grafo 3.



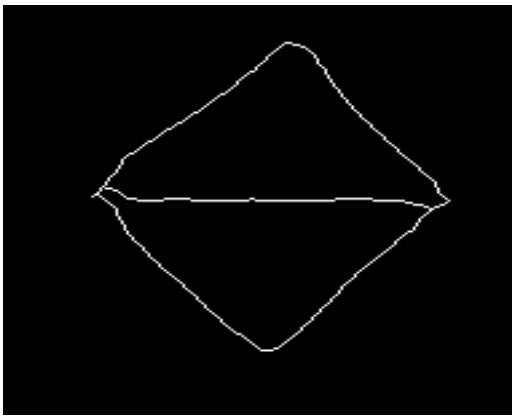
**(a)** Imagem de entrada.



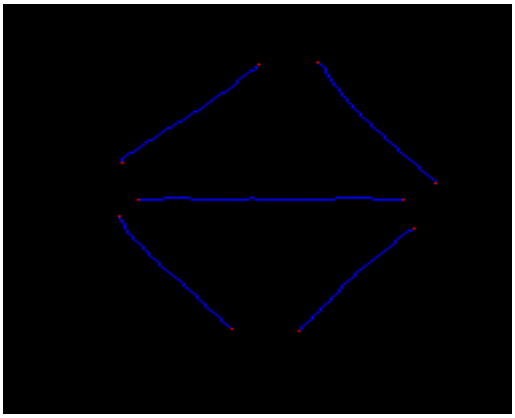
**(b)** Resultado pré-processamento.



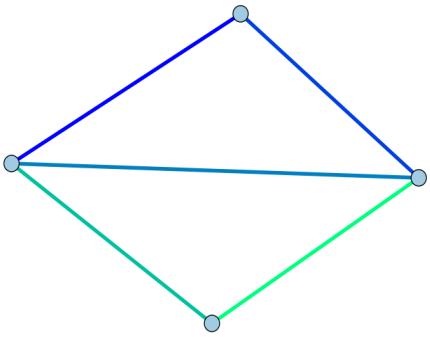
**(c)** Resultado segmentação.



**(d)** Resultado esqueletização.

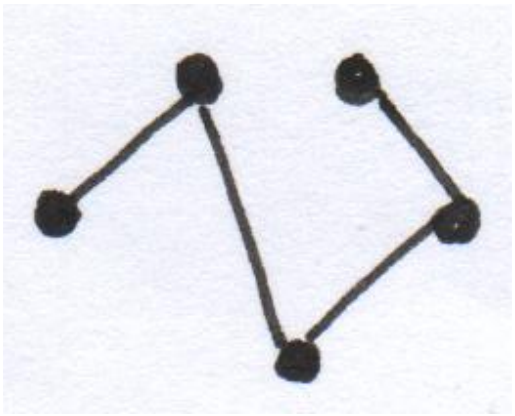


**(e)** Resultado classificação de pixels.

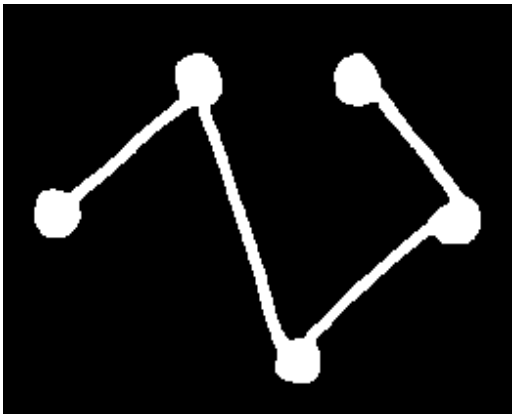


**(f)** Resultado final.

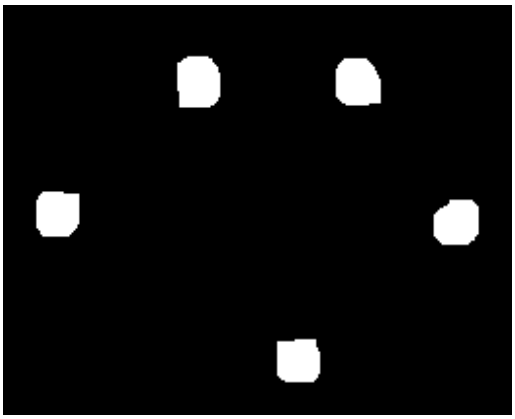
**Figura 9:** Grafo 4.



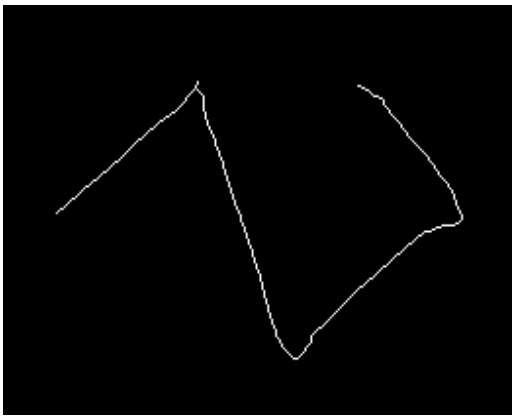
**(a)** Imagem de entrada.



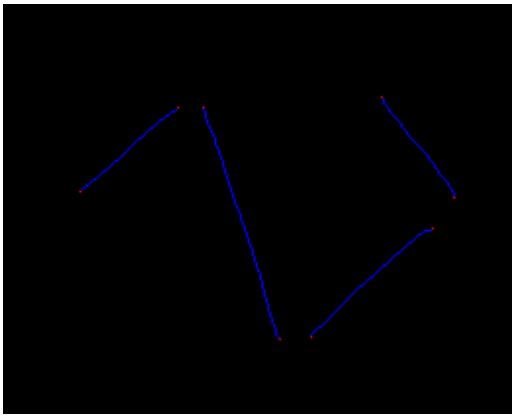
**(b)** Resultado pré-processamento.



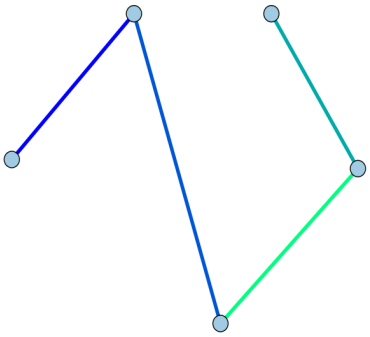
**(c)** Resultado segmentação.



**(d)** Resultado esqueletização.

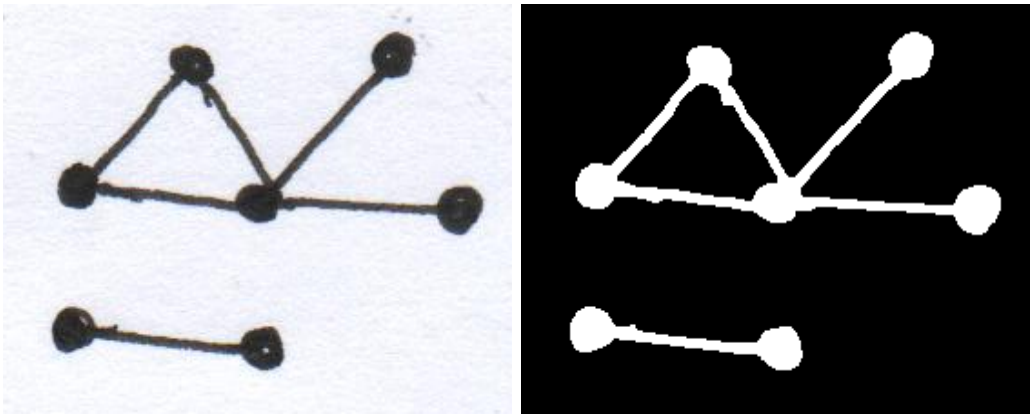


**(e)** Resultado classificação de pixels.



**(f)** Resultado final.

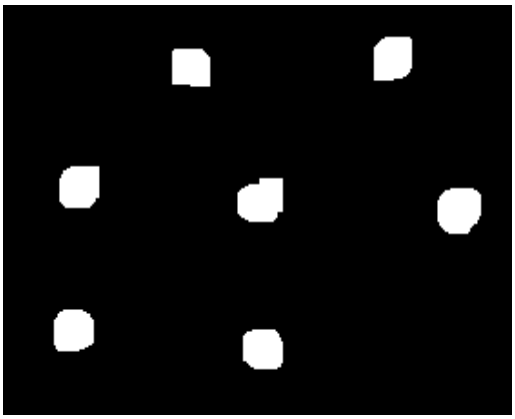
**Figura 10:** Grafo 5.



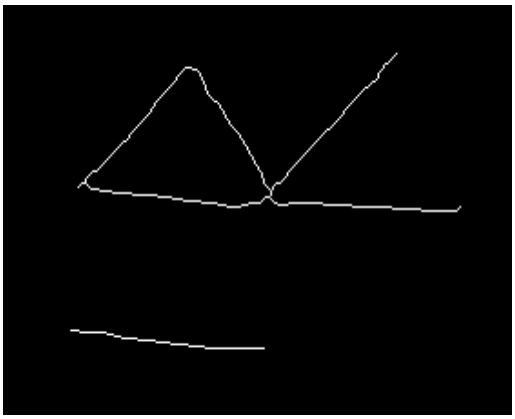
**(a)** Imagem de entrada.



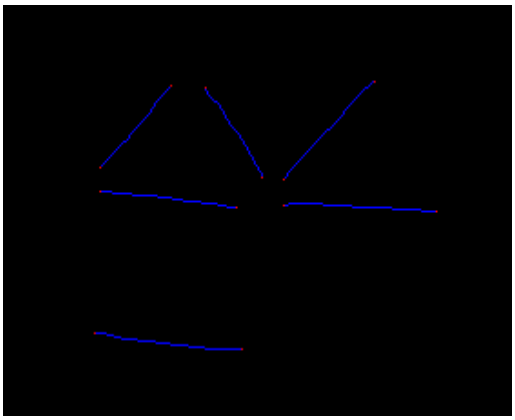
**(b)** Resultado pré-processamento.



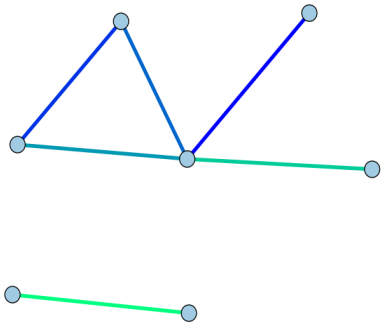
**(c)** Resultado segmentação.



**(d)** Resultado esqueletização.

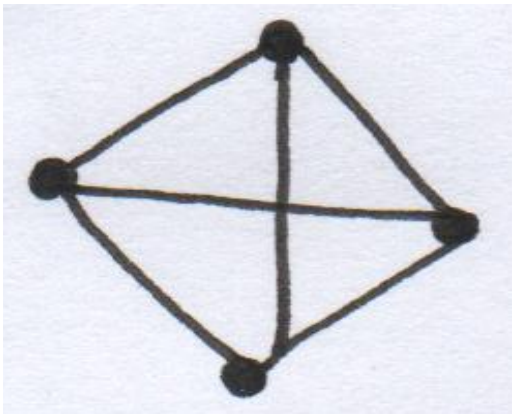


**(e)** Resultado classificação de pixels.

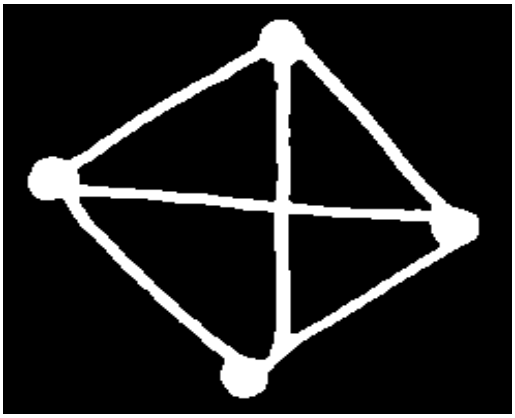


**(f)** Resultado final.

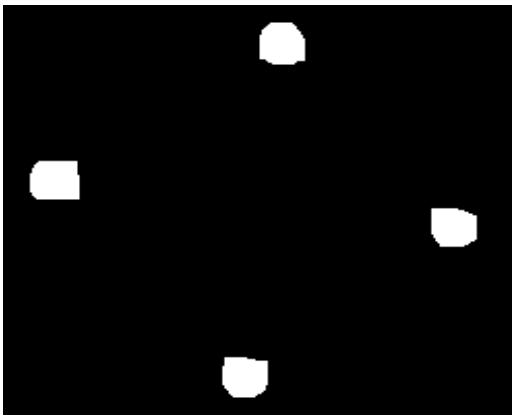
**Figura 11:** Grafo 6.



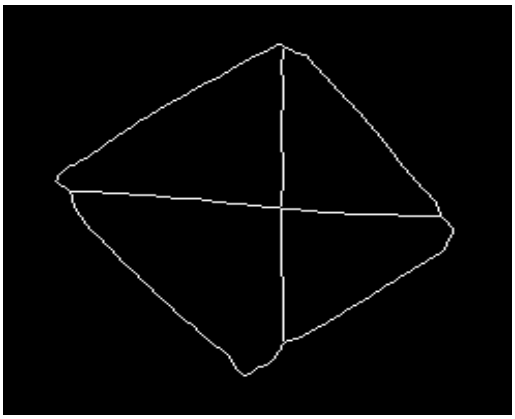
**(a)** Imagem de entrada.



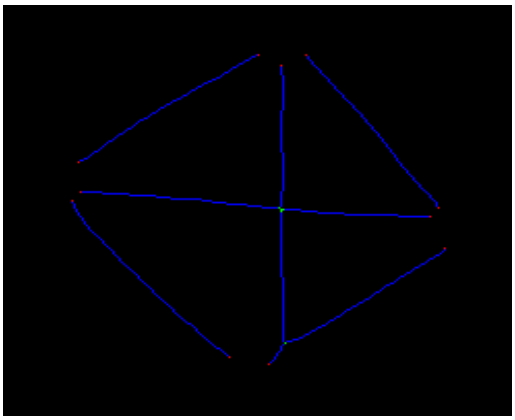
**(b)** Resultado pré-processamento.



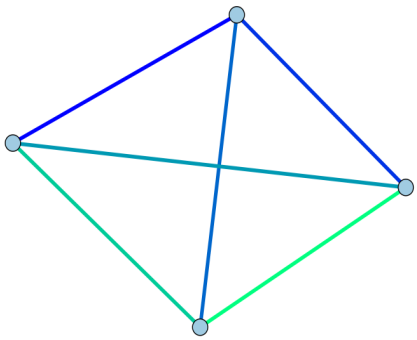
**(c)** Resultado segmentação.



**(d)** Resultado esqueletização.

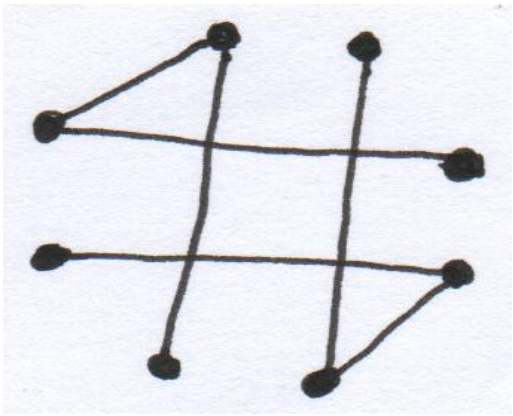


**(e)** Resultado classificação de pixels.

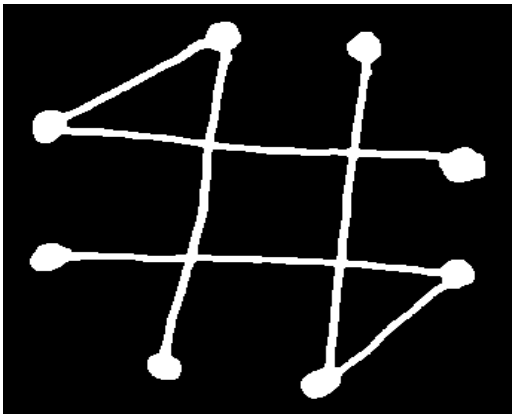


**(f)** Resultado final.

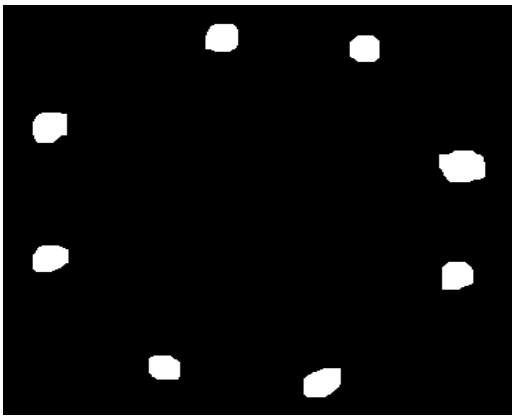
**Figura 12:** Grafo 7.



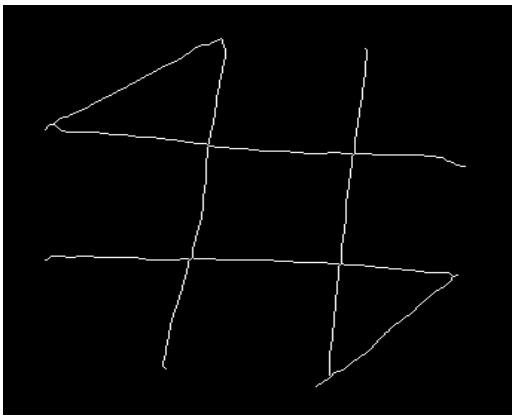
**(a)** Imagem de entrada.



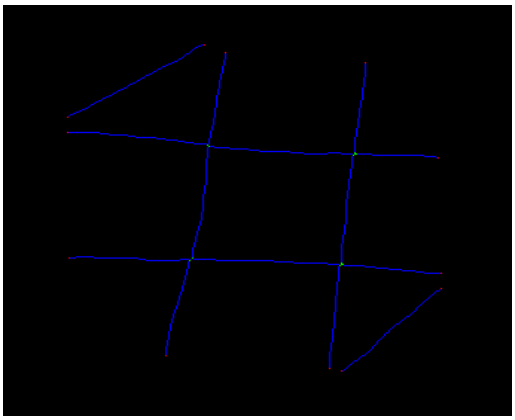
**(b)** Resultado pré-processamento.



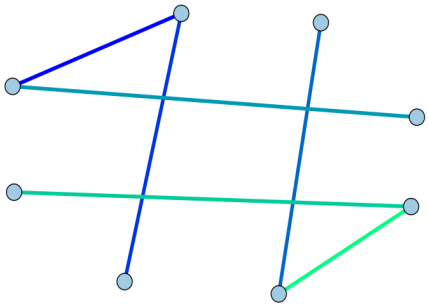
**(c)** Resultado segmentação.



**(d)** Resultado esqueletização.

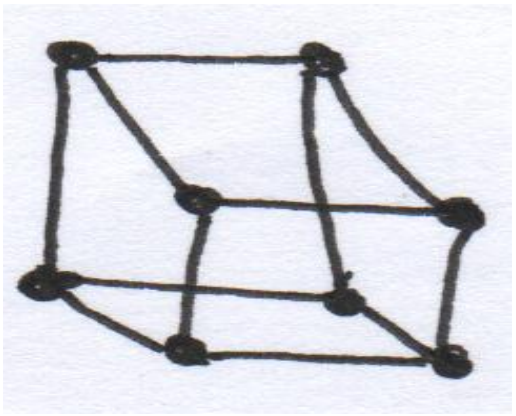


**(e)** Resultado classificação de pixels.

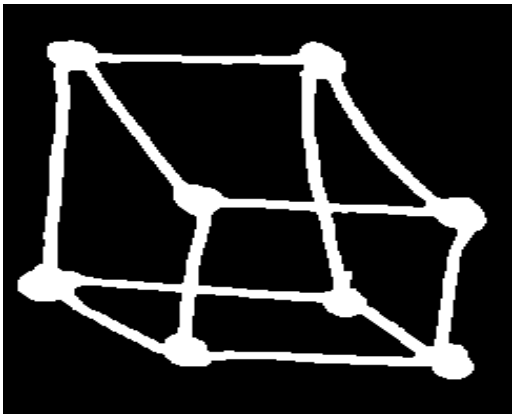


**(f)** Resultado final.

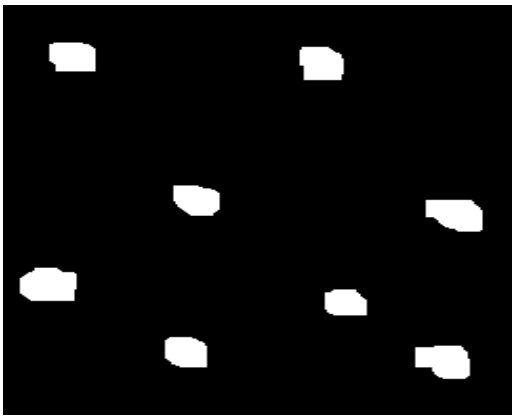
**Figura 13:** Grafo 8.



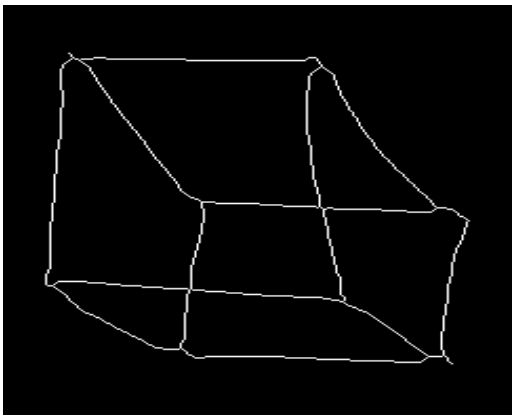
**(a)** Imagem de entrada.



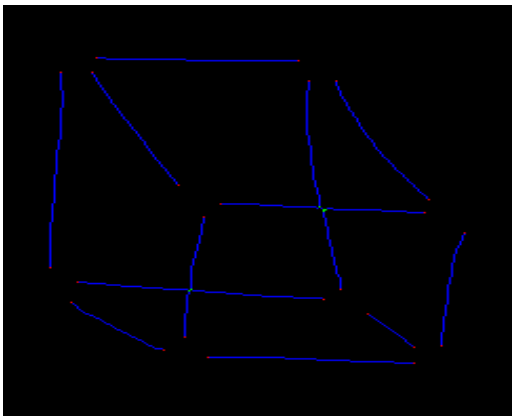
**(b)** Resultado pré-processamento.



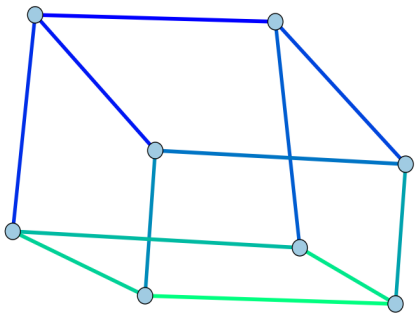
**(c)** Resultado segmentação.



**(d)** Resultado esqueletização.

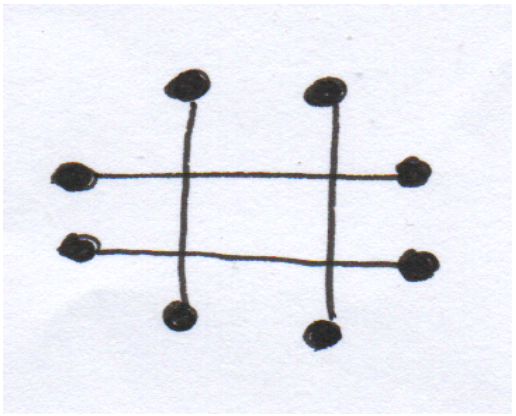


**(e)** Resultado classificação de pixels.

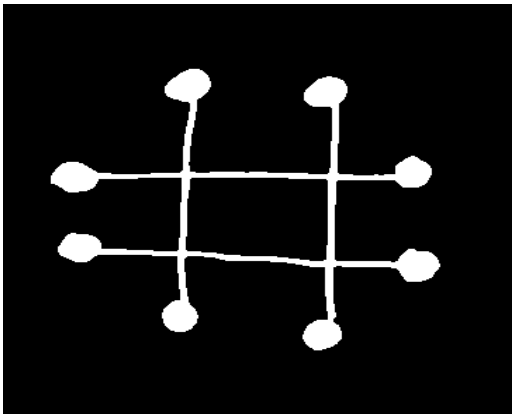


**(f)** Resultado final.

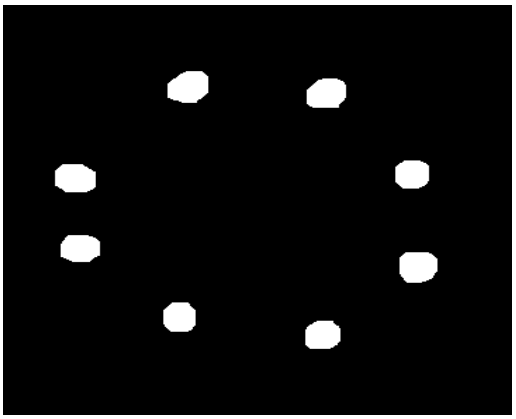
**Figura 14:** Grafo 9.



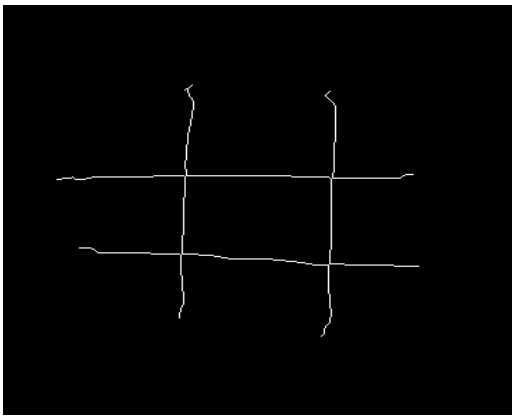
**(a)** Imagem de entrada.



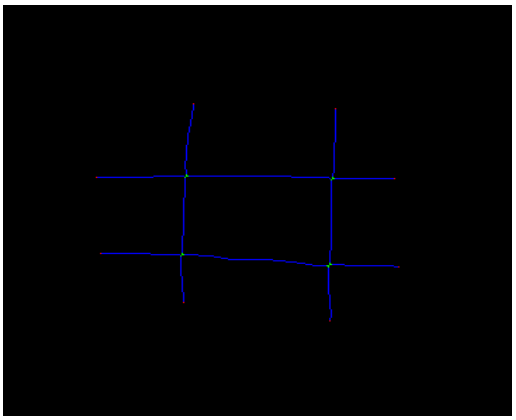
**(b)** Resultado pré-processamento.



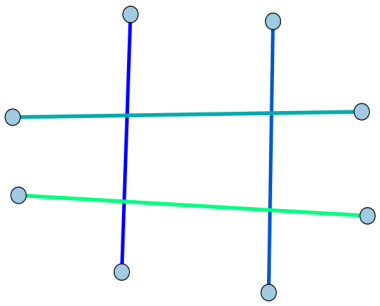
**(c)** Resultado segmentação.



**(d)** Resultado esqueletização.

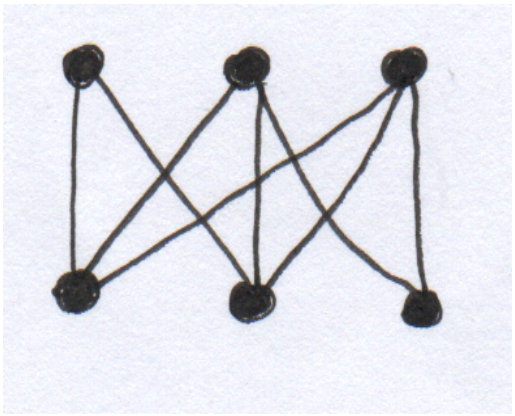


**(e)** Resultado classificação de pixels.

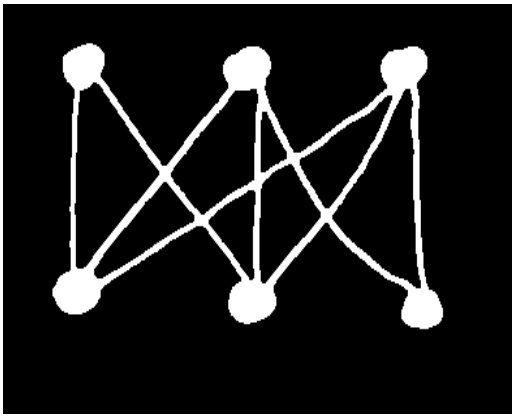


**(f)** Resultado final.

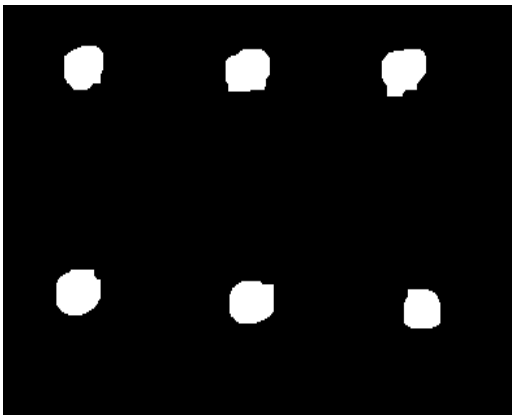
**Figura 15:** Grafo 10.



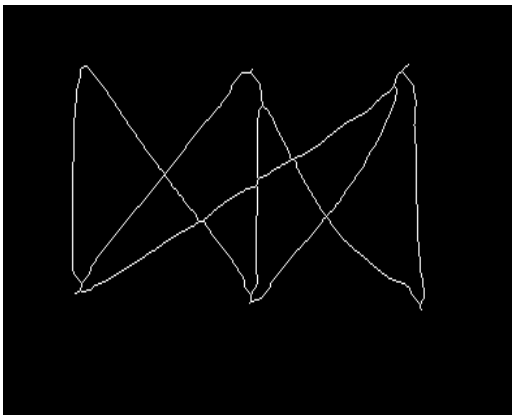
**(a)** Imagem de entrada.



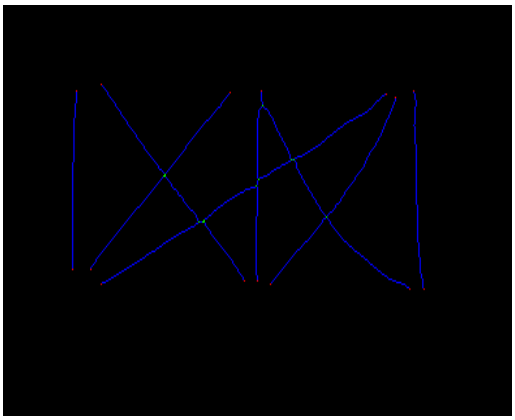
**(b)** Resultado pré-processamento.



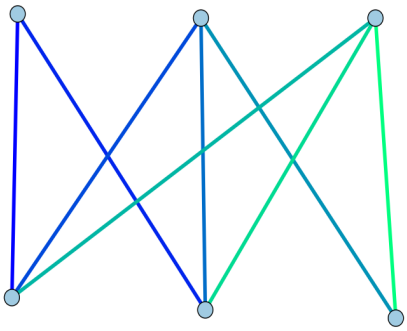
**(c)** Resultado segmentação.



**(d)** Resultado esqueletização.



**(e)** Resultado classificação de pixels.



**(f)** Resultado final.

**Figura 16:** Grafo 11.

que a escolha do vetor direcional escolhido sempre seja o correto. Em muitos casos é difícil fazer uma escolha exata para o vetor direcional, porém podemos aprimorar nossa heurística de junção de seções. Uma ideia inicial seria não analisar apenas o próximo pixel para se escolher uma direção, e sim analisar os $s \in \mathbb{N}^*$ próximos pixels para então definir os vetores direcionais a serem seguidos.

Para ilustrar o quão difícil é, por vezes, escolher o vetor direcional correto baseado somente no pixel seguinte, vejamos a Figura 17. Note que ambas as possíveis direções a serem seguidas pelo *crossing pixel* da seção de aresta mais a esquerda da imagem, não correspondem a direção mais frequente em passos anteriores. Analisando a imagem mais amplamente, reconhecemos que a direção mais natural a ser seguida seria a $(1, 1)$, porém até onde vemos nesta imagem, esta direção nunca fora tomada anteriormente. Assim, seria natural (mesmo que erroneamente) ser escolhida a direção $(-1, 1)$, visto que esta já fora tomada anteriormente. Portanto, acreditamos ser uma boa heurística não considerar apenas o próximo pixel a ser seguido, mas sim os $s$ seguintes pixels.

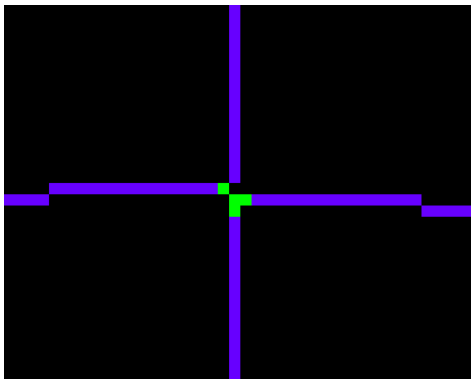


**Figura 17:** Direção da *crossing section*.

As Figuras 18-19 exibem alguns resultados mal sucedidos. Tais insucessos são devidos a erros do primeiro tipo, resultando em adição de vértices não existentes no grafo original, que na verdade deveriam ser pontos de cruzamento de arestas.

A Figura 20 exibe um resultado mal sucedido devido a erros do segundo tipo. Note que a Figura 20a é a mesma que a Figura 13a, porém rotacionada. A explicação para que uma tenha obtido sucesso no reconhecimento, e outra não é dada pelo fato que ao rotacionar a imagem, possivelmente, iniciamos a subfase de junção de um *crossing pixel* diferente se não rotacionada a imagem. Com isso, podemos iniciar de um *crossing pixel* cujo vetor direcional seja mais difícil de ser selecionado corretamente, e então ser feita uma escolha errônea.

Após uma escolha incorreta, todo as demais escolhas são influenciadas, visto que marcamos o *crossing pixel* da *port section* localizada para ser juntada.

*Observação* 3. O *crossing pixel* referente a primeira direção tomada incorretamente na Figura 20a é exatamente o da escolha da direção $(-1, 1)$ mostrado na Figura 17, que na verdade é a Figura 20a ampliada.

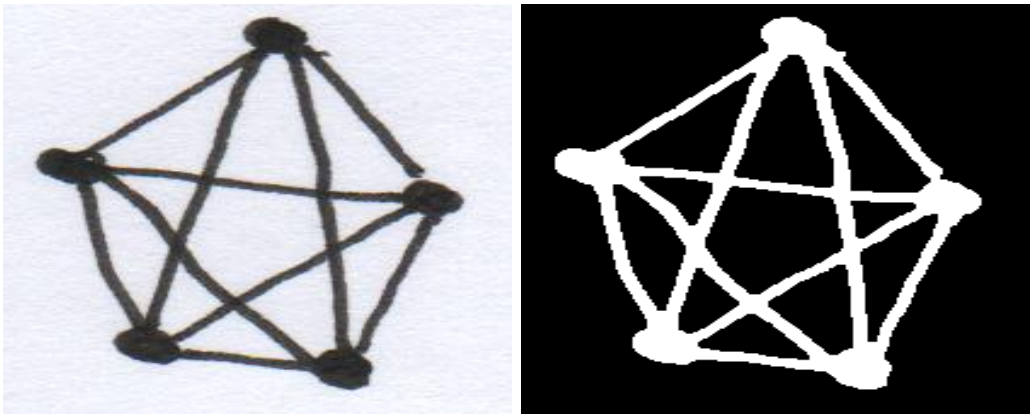
# 5 Conclusão e Trabalhos Futuros

Como podemos ver através dos exemplos dados, os resultados de reconhecimento foram razoáveis. Alguns erros de reconhecimento podem ser facilmente resolvidos com a utilização do algoritmo *distance image* [2], como feito em [1], para determinar automaticamente o valor das $k$ erosões e aberturas a serem feitas na fase de segmentação. Outros erros já são mais difíceis de serem contornados, como quando o vetor direcional tomado não leva a resultados bons. Para se resolver este último problema poderíamos realizar uma análise um pouco menos local para determinar o vetor direcional a ser escolhido, ou modificar (ou utilizar outro) algoritmo de esqueletização que facilitasse a escolha do vetor direcional.

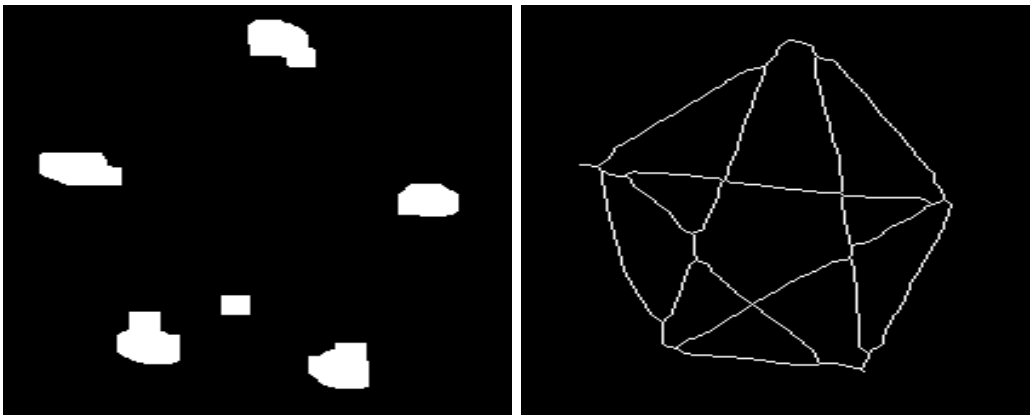
Como trabalhos futuros deixamos a implementação do algoritmo *distance image*, e do aprimoramento da heurística para escolha do vetor direcional mais adequado. Além disso, também deixamos como trabalhos futuros a execução de testes mais exaustivos para avaliar mais precisamente os resultados de nossa implementação. E, por fim, otimizarmos o nosso código, com o intuito de diminuir o tempo de execução da tarefa de reconhecimento topológico.

*Observação* 4. Este trabalho foi programado utilizando-se a linguagem de programação *Python*, juntamente com a biblioteca *PIL* para se abrir e salvar imagens. Também foram utilizadas as bibliotecas *numpy* para manipular *arrays*, *ndimage* para realizar testes das operações morfológicas antes de implementarmos as mesmas e *networkx* para armazenar, plotar e salvar o grafo resultante.

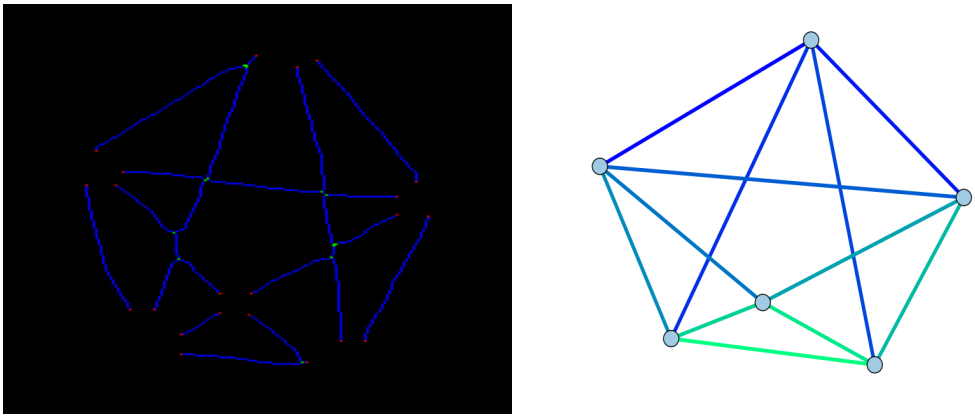
*Observação* 5. O código fonte deste trabalho pode ser encontrado em: `https://github.com/alexsanderam/OGR`.



**(a)** Imagem de entrada. **(b)** Resultado pré-processamento.

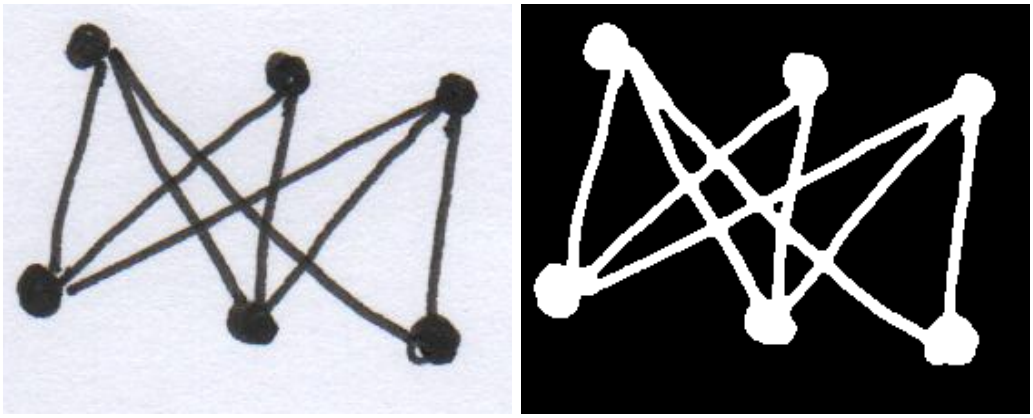


**(c)** Resultado segmentação. **(d)** Resultado esqueletização.

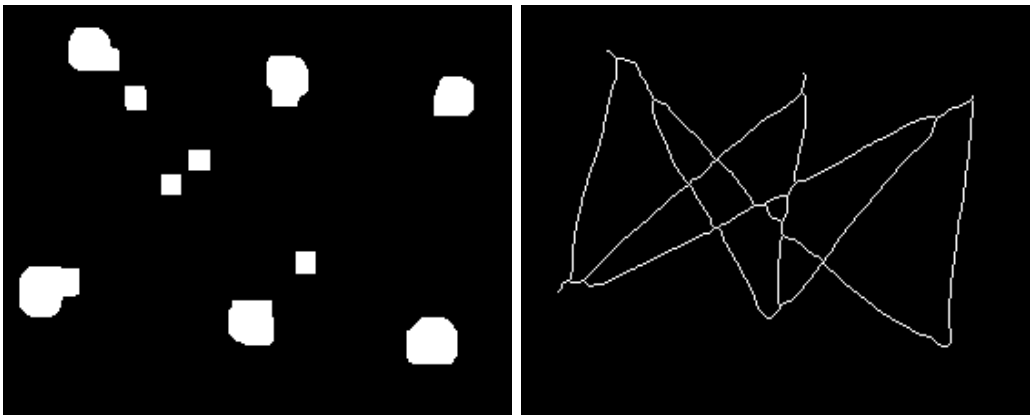


**(e)** Resultado classificação de pixels. **(f)** Resultado final.

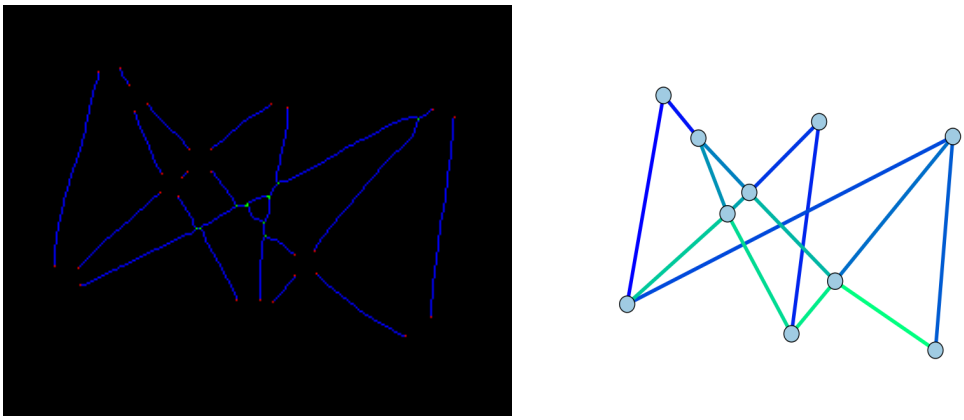
**Figura 18:** Grafo 12.



**(a)** Imagem de entrada. **(b)** Resultado pré-processamento.

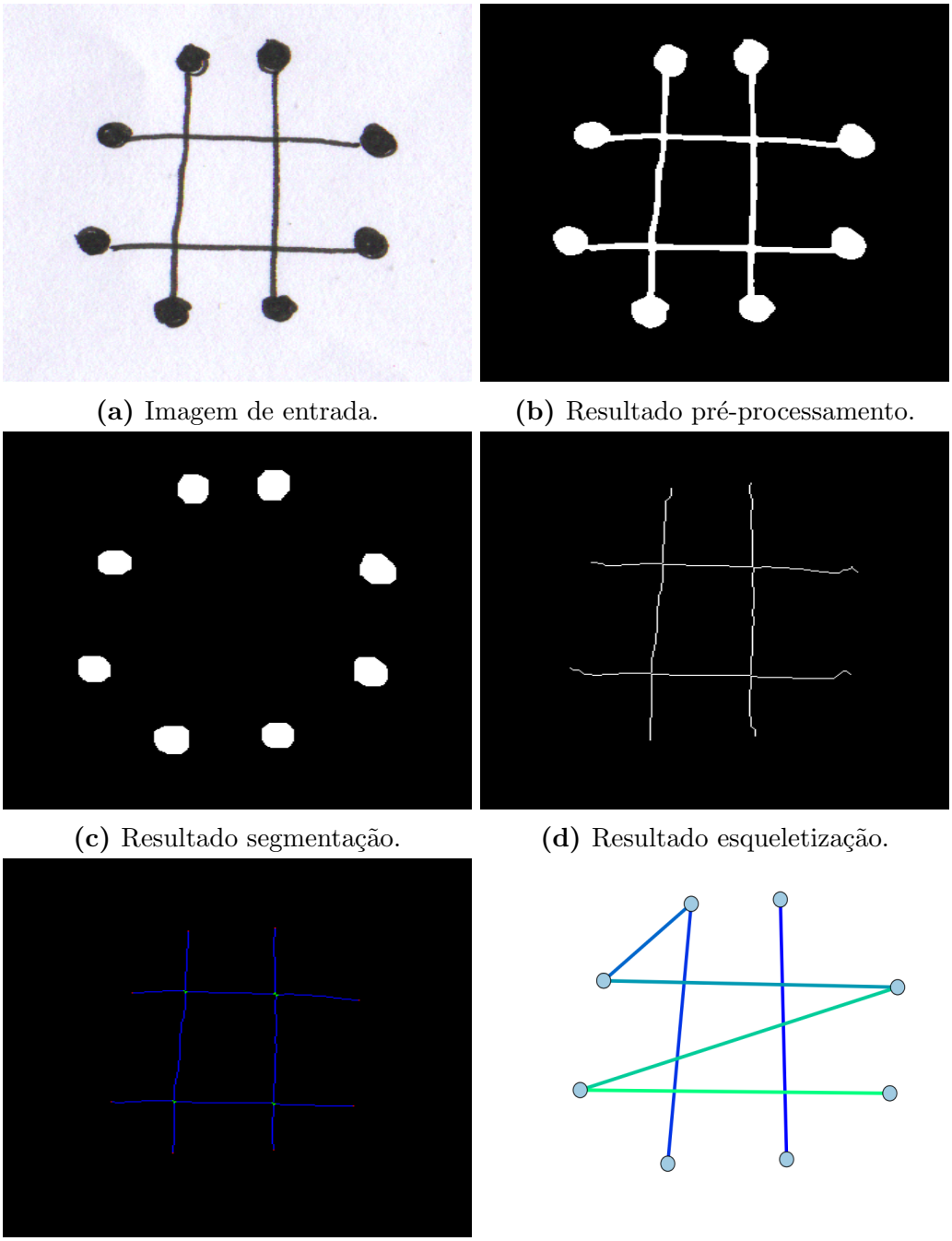


**(c)** Resultado segmentação. **(d)** Resultado esqueletização.



**(e)** Resultado classificação de pixels. **(f)** Resultado final.

**Figura 19:** Grafo 13.



**(a)** Imagem de entrada.

**(b)** Resultado pré-processamento.

**(c)** Resultado segmentação.

**(d)** Resultado esqueletização.

**(e)** Resultado classificação de pixels.

**(f)** Resultado final.

**Figura 20:** Grafo 10 rotacionado.

# Referências Bibliográficas


[1] Auer, C., Bachmaier, C., Brandenburg, F. J., Gleissner, A., e Reislhuber, J. Optical graph recognition. In *Graph Drawing* (2013), Springer, pp. 529–540.

[2] Borgefors, G. Distance transformations in digital images. *Computer Vision, Graphics, and Image Processing 34*, 3 (1986), 344 – 371.

[3] Glasbey, C. A., e Horgan, G. W. *Image Analysis for the Biological Sciences*. John Wiley & Sons, Inc., New York, NY, USA, 1995.

[4] Gonzalez, R., e Woods, R. *Processamento de imagens digitais*. Edgard Blucher, 2000.

[5] Hagberg, A. A., Schult, D. A., e Swart, P. J. Exploring network structure, dynamics, and function using NetworkX. In *Proceedings of the 7th Python in Science Conference (SciPy2008)* (Pasadena, CA USA, agosto de 2008), pp. 11–15.

[6] Jones, E., Oliphant, T., Peterson, P., e others. SciPy: Open source scientific tools for Python, 2001–. [Online; accessed 2015-06-12].

[7] Zhang, T., e Suen, C. Y. A fast parallel algorithm for thinning digital patterns. *Communications of the ACM 27*, 3 (1984), 236–239.