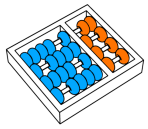
Universidade Estadual de Campinas
Instituto de Computação



Raí Caetano de Jesus

Formulações e Algoritmos para o Problema da Poligonização de Área Máxima

CAMPINAS
2018

Raí Caetano de Jesus

Formulações e Algoritmos para o Problema da Poligonização de Área Máxima

Dissertação apresentada ao Instituto de Computação da Universidade Estadual de Campinas como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.

Orientador: Prof. Dr. Fábio Luiz Usberti

Este exemplar corresponde à versão final da Dissertação defendida por Raí Caetano de Jesus e orientada pelo Prof. Dr. Fábio Luiz Usberti.

CAMPINAS
2018

Agência(s) de fomento e nº(s) de processo(s): CAPES; FUNCAMP

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca do Instituto de Matemática, Estatística e Computação Científica
Ana Regina Machado - CRB 8/5467

J499f

Jesus, Raí Caetano de, 1993-
Formulações e algoritmos para o problema da poligonização de área máxima / Raí Caetano de Jesus. – Campinas, SP : [s.n.], 2018.

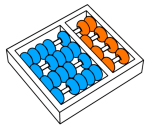
Orientador: Fábio Luiz Usberti.
Dissertação (mestrado) – Universidade Estadual de Campinas, Instituto de Computação.

1. Otimização combinatória. 2. Programação heurística. 3. Programação linear inteira. 4. Meta-heurística. 5. Matheurística. I. Usberti, Fábio Luiz, 1982-. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

Informações para Biblioteca Digital

Título em outro idioma: Formulations and algorithms for the maximum area poligonization problem
Palavras-chave em inglês:
Combinatorial optimization
Heuristic programming
Integer linear programming
Metaheuristic
Matheuristic
Área de concentração: Ciência da Computação
Titulação: Mestre em Ciência da Computação
Banca examinadora:
Fábio Luiz Usberti [Orientador]
Cid Carvalho de Souza
Pedro Henrique Del Bianco Hokama
Data de defesa: 30-01-2018
Programa de Pós-Graduação: Ciência da Computação

Universidade Estadual de Campinas
Instituto de Computação



Raí Caetano de Jesus

Formulações e Algoritmos para o Problema da Poligonização de Área Máxima

Banca Examinadora:

- Prof. Dr. Fábio Luiz Usberti
  Instituto de Computação - UNICAMP
- Prof. Dr. Cid Carvalho de Souza
  Instituto de Computação - UNICAMP
- Prof. Dr. Pedro Henrique Del Bianco Hokama
  Departamento de Engenharia de Produção - UFSCar

A ata da defesa com as respectivas assinaturas dos membros da banca encontra-se no processo de vida acadêmica do aluno.

Campinas, 30 de janeiro de 2018

A persistência é o menor caminho do êxito.
(Charles Chaplin)

Agradecimentos

Primeiramente, agradeço a minha noiva Michelle por toda compreensão diante às inúmeras vezes que não pude estar presente, e por todo seu apoio dedicado a mim para seguir em frente em meio a tantas dificuldades.

Agradeço a todos os amigos e familiares que direta ou indiretamente contribuíram para que eu pudesse realizar esse trabalho.

Agradeço também a tia de minha noiva, Marilene Rocha, por todo incentivo, ajuda e inúmeras caronas para minha cidade.

Meu profundo agradecimento ao meu orientador, Prof. Dr. Fábio Luiz Usberti, pelo profissionalismo e competência demonstrados, pelo apoio, compreensão e constante presença durante o desenvolvimento deste trabalho, mesmo diante a impossibilidade de minha presença nos meses finais, e por todos os conhecimentos repassados durante a pesquisa.

Agradeço aos servidores e professores do Instituto de Computação da Unicamp, que tornam possível dia a dia um ambiente acadêmico de excelência o qual pude usufruir durante o mestrado.

Gostaria de agradecer também às diversas pessoas que contribuíram para minha formação acadêmica e me permitiram chegar até aqui. Em especial, agradeço à todos os professores do IFMG - *Campus* Formiga, pelo esforço e excelentes conhecimentos repassados, conhecimentos estes que de certa forma me fizeram prosseguir na vida acadêmica.

Por fim, agradeço à Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) e ao Fundo de Apoio ao Ensino, à Pesquisa e Extensão (FAEPEX), que financiaram e tornaram possível essa pesquisa.

Resumo

O problema do caixeiro viajante euclidiano (*Traveling Salesman Problem - TSP*) do ponto de vista geométrico tem por objetivo encontrar um polígono simples sobre um dado conjunto de vértices cujo perímetro é mínimo. É possível derivar o problema de modo que o objetivo seja encontrar um polígono simples cuja área interna seja máxima: tal problema é conhecido como Poligonização de Área Máxima (*Maximum Area Polygonization* - MAXAP). O MAXAP é um problema de otimização combinatória NP-difícil com aplicações práticas em segmentos como reconhecimento de padrões, reconstrução de imagens, clusterização e robótica. Este trabalho propõe novas metodologias de solução e formulações matemáticas para o MAXAP, visando a implementação de algoritmos para metodologias exata, aproximada e heurística, bem como um estudo computacional para avaliar o desempenho das metodologias para o conjunto de instâncias desenvolvido. São propostos neste trabalho dois modelos matemáticos de programação linear inteira, duas heurísticas construtivas, uma metaheurística GRASP e uma *matheuristic* aplicada sobre um dos modelos matemáticos. Experimentos computacionais foram executados para comparar as metodologias propostas entre si e com um algoritmo $\frac{1}{2}$-aproximado da literatura. Análises comparativas de desempenho foram realizadas sobre os resultados obtidos mostrando que o GRASP obteve o melhor desempenho no critério de qualidade das soluções. As heurísticas construtivas propostas por sua vez obtiveram um desempenho superior sobre o algoritmo aproximado. Finalmente, os modelos matemáticos propostos mostram a dificuldade de resolver de maneira exata o MAXAP, encontrando soluções ótimas em uma hora somente para as instâncias de 10 pontos, em um conjunto de instâncias de até 100 pontos.

Abstract

The Traveling Salesman Problem (TSP) from a geometric point of view aims to find a simple polygon with minimum perimeter. It is possible to derive the problem so that the objective is to find a simple polygon whose enclosed area is maximum, such problem is known as Maximum Area Polygonization (MAXAP). The MAXAP is an NP-hard combinatorial optimization problem with practical applications in segments such as pattern recognition, image reconstruction, clustering and robotics. This work proposes new solution methodologies and mathematical formulations for MAXAP, aiming the implementation of algorithms for exact, approximate and heuristic solutions, as well as a computational study to evaluate the performance of the methodologies for a benchmark set of instances. Two mathematical models based on integer linear programming are proposed. In addition, two constructive heuristics, a GRASP metaheuristic, and a matheuristic are proposed for the solution of larger instances. Computacional experiments were conducted to compare the proposed methodologies among themselves and a $\frac{1}{2}$-approximation algorithm from literature. Comparative perfomance analysis were made on the results showing that the GRASP outperformed all other approaches with respect to solution quality. The constructive heuristics, on the other hand, outperformed the literature $\frac{1}{2}$-approximation algorithm. Finally, the proposed mathematical models have shown the hardness of exact solution for the MAXAP, finding optimal solutions in one hour only for the 10-vertices instances in a set of instances with 10, 25, 50 and 100 vertices.

Lista de Figuras

Lista de Tabelas

Sumário

Capítulo 1

Introdução

Dentre os problemas de otimização combinatória, o Problema do Caixeiro Viajante (*Traveling Salesman Problem - TSP*) é um dos mais investigados pela comunidade acadêmica [12]. Neste problema, dado um conjunto de cidades e o custo de viagem entre cada par de cidades, o objetivo é encontrar uma rota de comprimento mínimo que visita todas as cidades [1]. Do ponto de vista geométrico, o objetivo do TSP é encontrar um polígono simples[1], a partir de um dado conjunto de vértices, cujo perímetro é mínimo. Por outro lado, é possível avaliar um polígono por outra métrica: sua área interna [12]. Os problemas de minimizar e maximizar a área de um polígono são conhecidos na literatura como *Minimum Area Polygonization* (MINAP) e *Maximum Area Polygonization* (MAXAP), respectivamente.

Problemas do tipo MINAP e MAXAP surgem no contexto de reconhecimento de padrões, reconstrução de imagens e clusterização [13]. MAXAP, em particular, está intimamente relacionado com a Triangulação de Área Máxima, que é relevante no contexto da robótica [4, 14, 16] para explorar e triangularizar uma região com um enxame de robôs.

Este trabalho trata do Problema da Poligonização de Área Máxima (MAXAP). Fekete [13] prova que os problemas MINAP e MAXAP são da classe NP-difícil, o que implica que não podem ser resolvidos em tempo polinominal, a menos que P = NP.

Embora possuam aplicações práticas, MINAP e MAXAP são problemas recentes na literatura. Existem muitos estudos focados na geração de polígonos aleatórios. Alguns buscam estabelecer limites inferior e superior para a contagem de poligonizações simples, por exemplo (veja [19, 33]), e outros consideram tipos especiais de poligonizações simples, como *Monotone Polygons* e *Starshaped Polygons* [23, 10]. Contudo, poucos trabalhos abordam a geração de polígonos simples com área mínima ou máxima. O trabalho mais recente é dos autores Peethambaran, Parakkat e Muthuganapathy [26], que propõem uma heurística para encontrar soluções para os problemas MINAP e MAXAP dado um conjunto de pontos no plano.

Este trabalho apresenta formulações e algoritmos para o MAXAP. Além do algoritmo aproximado proposto por Fekete, seis novas metodologias são apresentadas, dividindo-se em métodos exatos e heurísticos, sendo: dois modelos de programação linear inteira (PLI); duas heurísticas construtivas, onde uma baseia-se na adição de triângulos vazios e a outra

[1] Um polígono é dito ser simples quando nenhum dos seus segmentos de reta (arestas) se cruzam.

é baseada na adição de envoltórias convexas; uma metaheurística (GRASP) e; por fim, uma hibridização entre um modelo PLI e a heurística de triangulação (*matheuristic*).

O presente trabalho está organizado da seguinte forma: o Capítulo 2 apresenta conceitos introdutórios relevantes que são utilizados no decorrer da dissertação; no Capítulo 3 são dadas as definições sobre o problema de poligonização de área máxima e adicionalmente são revisados os últimos trabalhos relacionados; o Capítulo 4 descreve conceitualmente as metodologias de solução propostas no trabalho; o Capítulo 5 apresenta o estudo computacional realizado com as metodologias, descrevendo o ambiente de desenvolvimento, o conjunto de instâncias utilizadas, os testes executados para parametrização da metaheurística GRASP e da *matheuristic*, bem como uma análise comparativa entre os resultados obtidos pelas metodologias; finalmente, o Capítulo 6 apresenta as conclusões e trabalhos futuros.

Capítulo 2

Conceitos básicos

O propósito deste capítulo é introduzir alguns conceitos básicos necessários para um melhor entendimento desta dissertação. A Seção 2.1 introduz brevemente conceitos relacionados à Otimização Combinatória. Na Seção 2.2 apresentamos uma das linguagens de modelagem mais utilizadas em problemas de otimização e, por fim, a Seção 2.3 mostra como encontrar desvios de otimalidade sobre a solução ótima para um problema de otimização.

2.1 Otimização combinatória

Otimização Combinatória é uma sub-área da otimização matemática, cujo objetivo se resume a encontrar, dentre todas as possíveis soluções de um conjunto de soluções factíveis, aquela cujo custo seja o menor (minimização) ou o maior (maximização) possível. Nesses problemas o conjunto de soluções factíveis pode ser representado por um valor inteiro, conjunto, permutação, grafo, etc.

Resende e Ribeiro [31] definem genericamente um problema de otimização combinatória como um conjunto finito $E = \{1, ..., N\}$, um conjunto de soluções factíveis $F \subseteq 2^{|E|}$ e uma função objetivo $f : 2^{|E|} \rightarrow \mathbb{R}$, definidos para cada problema específico. No caso de maximização, busca-se uma solução ótima $S^* \in F$ tal que $f(S^*) \geq f(S), \forall S \in F$.

Uma forma de resolver problemas de otimização combinatória é simplesmente enumerar todas as soluções possíveis e selecionar a melhor. Entretanto, para muitos problemas este método ingênuo torna-se impraticável, visto que o número de soluções possíveis pode ser de ordem exponencial (explosão combinatória) em relação aos dados de entrada.

Existem muitas técnicas na área de otimização combinatória para tratar esses problemas onde a simples enumeração não é viável. Dentre essas diversas técnicas, destacam-se:

- Programação Linear e Programação Inteira: são técnicas muito conhecidas no campo da otimização. A Programação Linear consiste em expressar uma função objetivo em termos de uma função linear definida sobre um conjunto de variáveis, as quais devem satisfazer um conjunto de restrições lineares (equações e inequações). Dada uma função objetivo que expressa como é calculado o custo a ser minimizado ou maximizado, aplica-se um algoritmo (tipicamente o *Simplex*) que resolve o problema.

Caso seja exigido que algumas ou todas as variáveis assumam apenas valores inteiros, constitui-se um problema de Programação Linear Inteira [24].

- Métodos Heurísticos: quando não se conhecem algoritmos eficientes para encontrar a solução ótima em problemas de otimização, uma solução "boa" pode ser suficiente. Métodos Heurísticos são algoritmos que não garantem encontrar a solução ótima para o problema, mas são capazes de retornar uma solução "boa" em um tempo aceitável. Pode se entender por solução "boa" como sendo uma solução não-trivial que é melhor do que as soluções obtidas na prática por métodos baseados em senso comum.

- Algoritmos Aproximados: métodos heurísticos não dão garantia a respeito da qualidade da solução encontrada. No entanto, há casos onde uma certificação de qualidade é necessária. Por exemplo, pode ser que a aplicação não precise da solução ótima, mas exija uma solução que esteja a um certo desvio máximo da solução ótima. Seja α esse desvio, um algoritmo α-aproximado para um problema de otimização é um algoritmo tempo de polinomial que, dado qualquer instância do problema, produz uma solução cujo valor está dentro do fator de α do valor de uma solução ótima [36].

2.2 Programação linear inteira

Um problema de Programação Linear (PL) consiste em minimizar ou maximizar uma função linear sujeita a um conjunto de restrições formadas por equações e/ou inequações lineares que atuam sobre as variáveis de decisão [3]. Considere o seguinte exemplo de um programa linear [37]:

$$MAX \quad \{c^T x : Ax \leq b, x \geq 0\}$$

onde x representa o vetor das variáveis de decisão, c é o vetor de *coeficientes de custo* associados a cada variável, b é o vetor de coeficientes do lado direito das desigualdades ou igualdades, e A é a matriz de coeficientes das restrições. Assim, um problema de programação linear pode ser formulado da seguinte forma [3]:

$$\begin{array}{rccccccccc}
MAX & c_1x_1 & + & c_2x_2 & + & ... & + & c_nx_n & & \\
sujeito\ a & a_{11}x_1 & + & a_{12}x_2 & + & ... & + & a_{1n}x_n & \geq & b_1 \\
 & a_{21}x_1 & + & a_{22}x_2 & + & ... & + & a_{2n}x_n & \geq & b_2 \\
 & \vdots & & \vdots & + & ... & + & \vdots & & \vdots \\
 & a_{m1}x_1 & + & a_{m2}x_2 & + & ... & + & a_{mn}x_n & \geq & b_m \\
 & x_1, & & x_2, & & ..., & & x_n & \geq & 0
\end{array}$$

A PL está entre as técnicas matemáticas mais bem-sucedidas e mais amplamente aplicadas em problemas de otimização combinatória, sendo utilizada em diversas áreas, como engenharias, transportes, energia, telecomunicações, economia, manufatura, logística entre outras. Alguns problemas de otimização combinatória podem ser resolvidos com

programação linear. No entanto, partindo do princípio que P $\neq$ NP, problemas NP-difíceis não podem ser resolvidos através de PL.

Na vida real, entretanto, é muito comum que algumas variáveis (ou todas) precisem assumir valores inteiros. O ramo da Programação Linear Inteira (PLI) trata desse tipo de problema. Adicionar a restrição de que as variáveis assumam valores inteiros acrescenta uma poderosa flexibilidade aos modelos de PLI. Em contrapartida, essa flexibilidade pode fazer o problema se tornar mais difícil.

No caso geral, a programação linear inteira é NP-difícil e, uma vez que variáveis discretas podem ser representadas como números inteiros, muitos problemas de otimização combinatória NP-difíceis são modelados por meio de programação linear inteira. Ao longo dos anos, técnicas foram desenvolvidas para tratar problemas de otimização combinatória através de PLI, destacando-se os métodos *branch-and-bound*, *branch-and-cut* e *branch-and-price*.

Para mais detalhes sobre conceitos e formulações de PL e PLI para problemas de otimização, são recomendadas as referências [3, 9, 25, 37].

2.3 Desvios de otimalidade

Em problemas de otimização combinatória, normalmente deseja-se obter o valor de uma solução ótima. Normalmente o valor ótimo é desconhecido, porém é possível definir um intervalo onde esse valor se encontra, delimitado por limitantes primal (L_P) e dual (L_D). No caso de problemas de maximização, o limitante superior é o limitante dual e o limitante inferior é o limitante primal. O inverso vale para um problema de minimização. Sem perda de generalidade, nesta seção consideraremos os problemas de maximização.

Na prática, L_D é um limitante superior para o valor de uma solução ótima, ou seja, se $c(s^*)$ representa o valor ótimo, $c(s^*) \leq L_D$. Analogamente, L_P é um limitante inferior para o valor de uma solução ótima, isto é, $c(s^*) \geq L_P$.

Uma das maneiras de se obter um limitante primal é encontrar uma solução viável para o problema, enquanto que limitantes duais podem ser obtidos através de relaxações do problema. Um problema é uma relaxação de um problema original se o primeiro possuir uma solução ótima cujo custo não é inferior ao custo da solução ótima do problema original para a mesma entrada. Portanto, resolver a relaxação nos dá uma garantia de limitante superior para o problema original.

Uma das relaxações mais comuns para um modelo de programação linear inteira é a sua relaxação linear. A relaxação linear permite que as variáveis inteiras do modelo PLI assumam valores fracionários, isto é, a restrição de integralidade das variáveis é removida. O modelo relaxado possui um conjunto de soluções viáveis que é superconjunto das soluções do modelo PLI. No entanto, para o modelo relaxado gerar bons limitantes duais, o seu valor ótimo deve se situar o mais próximo possível do valor ótimo do modelo PLI. Desenvolver um modelo que atenda esse requisito pode ser considerado uma das principais dificuldades na modelagem de problemas através de programação linear inteira.

Os limitantes primais e duais podem provar a otimalidade de soluções caso se igualem ao custo $c(s)$ destas soluções, isto é, $L_P = c(s) = L_D$, ou se forem suficientemente

próximos. Diante da dificuldade em resolver problemas de otimização até sua otimalidade, considera-se então métodos que encontrem "boas" soluções em tempos aceitáveis. Através dos limitantes primal e dual é possível definir um desvio de otimalidade $GAP = \frac{(L_D - L_P)}{L_P}$ que representa, no pior caso, o quão afastado o valor de uma solução pode estar do ótimo.

Capítulo 3

Problema da poligonização de área máxima

O problema da Poligonização de Área Máxima (MAXAP) foi primeiramente estudado por Fekete em sua tese de doutorado [12], estudo este continuado e publicado em [15, 13]. Nos últimos anos, problemas relacionados ganharam importância devido às suas relações com exploração e vigilância por enxames de robôs [4, 14, 16]. Isso torna importante desenvolver novos métodos para computar soluções ótimas para problemas como MAXAP.

3.1 Definições

Um *polígono* é uma sequência ordenada de pontos $\langle p_0, p_1, ..., p_{n-1}\rangle$ com $n \geq 3$ juntamente com o conjunto de segmentos $\langle \overline{p_0p_1}, \overline{p_1p_2}, ..., \overline{p_{n-2}p_{n-1}}\rangle$ denominados *arestas* [35]. O ponto onde duas arestas se encontram é denominado *vértice* [21]. Um polígono é *fechado* se o primeiro e o último ponto estão conectados por um segmento de linha. Um polígono *simples* é um polígono fechado sem interseções entre arestas, exceto aquelas em pontos finais comuns de arestas consecutivas [35].

Seja P um conjunto de pontos no plano Euclidiano. Uma *poligonização simples* de P é um polígono simples cujos vértices são compostos exclusivamente por P. A menos que todos os pontos sejam colineares, um conjunto de pontos no plano sempre permite uma poligonização simples [35]. É conhecido que a quantidade de poligonizações simples para um conjunto de n pontos é exponencial em n [18]. Se os pontos estão em posição convexa, o conjunto permite uma única poligonização, a *envoltória convexa* (do inglês *convex hull*) [35]. A envoltória convexa de um conjunto de pontos P é o menor polígono convexo $CH(P)$ no qual cada ponto em P está na borda de $CH(P)$ ou em seu interior [21].

Dado um polígono simples $\mathcal{P}$ (côncavo ou convexo) definidos por seus vértices $V_i = (x_i, y_i)$ para $i = (0, 1, ..., n)$, onde $V_n = V_0$, sua área sinalizada pode ser obtida da seguinte forma: Seja P um ponto qualquer, para cada aresta $e_i = V_iV_{i+1}$ de $\mathcal{P}$ construa o triângulo $\Delta_i = PV_iV_{i+1}$. A área $\mathcal{P}$ é igual a soma das áreas sinalizadas de todos os triângulos Δ_i para $i = (0, 1, ..., n-1)$. Por exemplo, na Figura 3.1, considerando a orientação do polígono no sentido anti-horário, os triângulos Δ_2 e Δ_{n-1} tem área positiva, contribuindo

positivamente para a área total do polígono $\mathcal{P}$. Entretanto, somente parte das áreas de Δ_2 e Δ_{n-1} estão dentro de $\mathcal{P}$. Por outro lado, os triângulos Δ_0 e Δ_1 têm áreas negativas, cancelando a área externa ao polígono de Δ_2 e Δ_{n-1}. Ao final, todas as áreas externas são canceladas, totalizando somente a área interna do polígono [34].

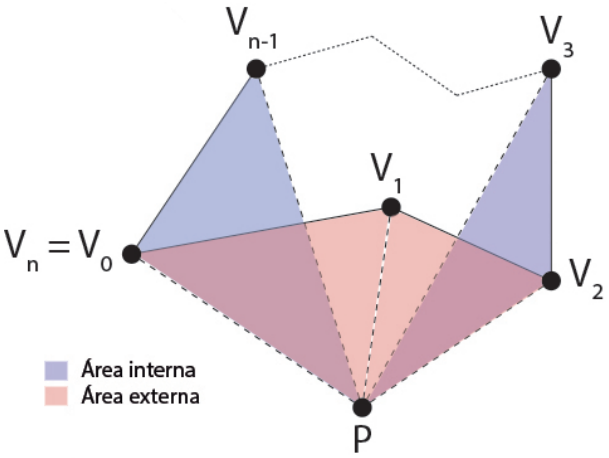


Figura 3.1: Exemplo da soma de áreas sinalizadas de triângulos para um polígono $\mathcal{P}$ [34].

A área sinalizada de um triângulo Δ_i pode ser encontrada dividindo-se o determinante da matriz de coordenadas de seus vértices por 2 [28].

$$A_{\Delta_i} = \frac{1}{2} \times \begin{bmatrix} x_0 & y_0 & 1 \\ x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \end{bmatrix} = \frac{x_0y_1 + y_0x_2 + x_1y_2 - y_1x_2 - x_0y_2 - y_0x_1}{2}$$

Considerando P como a origem, isto é, $P = (0, 0)$, então a fórmula para área sinalizada de um triângulo Δ_i se reduz a

$$A_{\Delta_i} = \frac{x_iy_{i+1} - y_ix_{i+1}}{2} \tag{3.1}$$

Logo, a área de um polígono simples pode ser obtida por meio da seguinte equação:

$$A_{\mathcal{P}} = \left| \frac{(x_0y_1 - y_0x_1) + (x_1y_2 - y_1x_2) + ... + (x_{n-1}y_0 - y_{n-1}x_0)}{2} \right| \tag{3.2}$$

Um modo alternativo para calcular a área de um polígono consiste em particioná-lo em triângulos [12]. Este tipo de partição é denominado *triangulação*. A triangulação de um polígono consiste em sua decomposição em triângulos vazios por meio da adição de um conjunto maximal de diagonais que não se interceptam [8] (Figura 3.2). Seja P um conjunto de pontos no plano, um triângulo vazio é um triângulo com vértices em P e nenhum outro ponto de P está em seu interior.

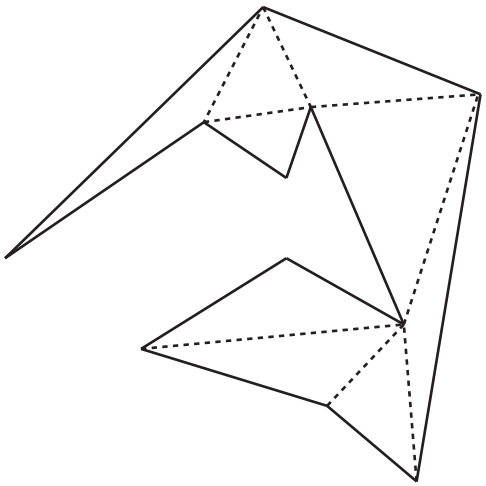


Figura 3.2: Exemplo de triangulação de um poligono [20].

A seguir será apresentada a definição do MAXAP [12].

Poligonização de Área Máxima: Seja P um conjunto finito de pontos no plano Euclidiano. Dentre todos os polígonos simples com um conjunto de vértices P, encontre um cuja área interna seja máxima.

Na literatura, a versão de minimização do problema é conhecida como *Minimum Area Polygonization* (MINAP) [12].

Pick propôs um teorema que dá uma interpretação combinatória para a determinação da área de um polígono simples construído sobre um *grid* (quadriculado) de pontos equidistantes, de forma que todos os vértices do polígono sejam *gridpoints* (pontos do quadriculado) [12]. Ele mostrou que qualquer ponto do *grid* que intercepta o polígono contribui com meia unidade de área, enquanto que os pontos no interior contribuem com uma unidade de área. Com base nisso, demonstrou o seguinte teorema [12]:

Teorema 1 *Seja $\mathcal{P}$ um polígono simples cujos vértices são gridpoints. Seja $b(\mathcal{P})$ o número de gridpoints que interceptam $\mathcal{P}$ e $i(\mathcal{P})$ o número de gridpoints no interior de $\mathcal{P}$. Então, a área de $\mathcal{P}$, $area(\mathcal{P})$, é dada pela Equação (3.1).*

$$area(\mathcal{P}) = \frac{b(\mathcal{P})}{2} + i(\mathcal{P}) - 1 \tag{3.3}$$

A Figura 3.3 apresenta um exemplo de um polígono simples construído sobre um *grid* de pontos equidistantes. Os pontos do *grid* em vermelho representam os pontos que interceptam o polígono, e os amarelos representam os pontos interiores ao polígono. Substituindo $b(\mathcal{P})$ e $i(\mathcal{P})$ na equação pelas quantidades de pontos vermelhos e amarelos, respectivamente, a área resultante do polígono ilustrado na figura é $\frac{14}{2} + 39 - 1 = 45$.

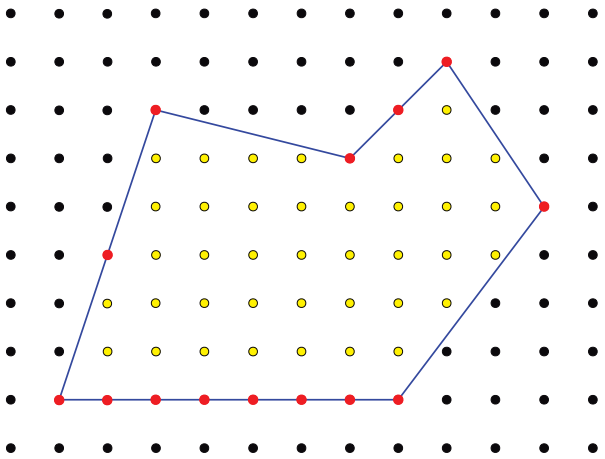


Figura 3.3: Polígono de área interna 45 construído sobre quadriculados de área unitária.

Com base nos conceitos definidos por Pick, o melhor que pode ser feito quando deseja-se minimizar a área é evitar que o polígono intercepte ou contenha quaisquer outros *gridpoints* além dos dados vértices de $\mathcal{P}$. O raciocínio análogo pode ser feito quando deseja-se maximizar a área do polígono [12]. Estes conceitos são usados no Teorema 2 demonstrado por Fekete [12], o qual estabelece limitantes inferior e superior para a área de um polígono.

Teorema 2 *Seja P um conjunto de n gridpoints, $h_i(P)$ o número de gridpoints não pertencentes a P que estão no interior da envoltória convexa de P, e $h_b(P)$ o número de gridpoints não pertencentes a P que interceptam a envoltória convexa de P. Para qualquer polígono simples $\mathcal{P}$ com conjunto de vértices P, as inequações abaixo (3.4) fornecem limitantes inferior e superior para $area(\mathcal{P})$.*

$$\frac{n}{2} - 1 \leq area(\mathcal{P}) \leq \frac{h_b(P)}{2} + h_i(P) + \frac{n}{2} - 1 \tag{3.4}$$

Estes limitantes sugerem uma versão do MINAP e do MAXAP para vértices definidos sobre *grids*, conhecidas como *Grid Avoiding Polygon* (GAP) e *Grid Approximating Polygon* (GAXP) [12].

Grid Avoiding Polygon (GAP): Dados n *gridpoints* no plano. Existe um polígono simples neste conjunto de vértices que não contém quaisquer outros *gridpoints* em sua borda ou em seu interior, e portanto tem um valor de área de $\frac{n}{2} - 1$?

Grid Approximating Polygon (GAXP): Dados n *gridpoints* no plano. Existe um polígono simples neste conjunto de vértices que contém tantos *gridpoints* quanto possível, e portanto tem um valor de área de $\frac{n}{2} + \frac{h_b(P)}{2} + h_i(P) - 1$?

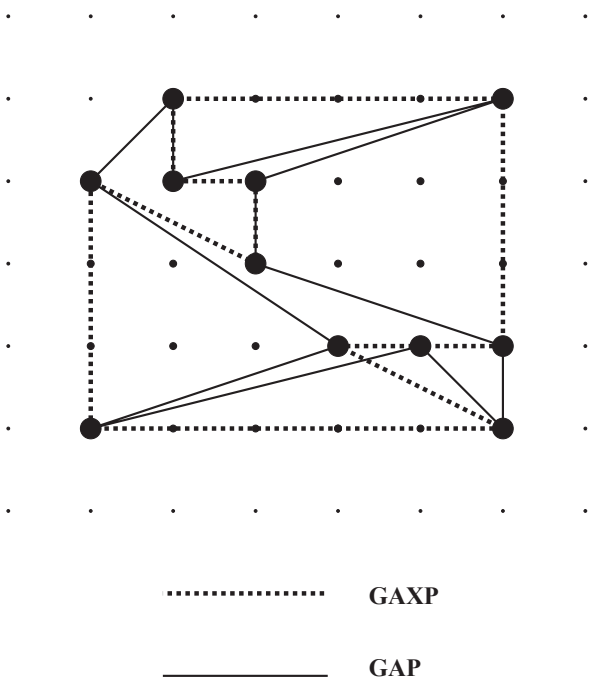


Figura 3.4: Um conjunto de *gridpoints* com GAP e GAXP. Adaptado de [13].

Fekete [12] provou que GAP e GAXP são ambos problemas NP-Completos. Uma vez que o GAP é um caso particular do MINAP, isso implica diretamente na NP-Completude do MINAP. Fekete também demonstrou, a partir do resultado do GAP, que o MAXAP é NP-Completo.

Para o MAXAP, Fekete [12] propôs um algoritmo aproximado (Seção 4.1) para obter um polígono simples para um conjunto de pontos P cuja área é maior ou igual à metade da área da envoltória convexa de P. Uma vez que a área da envoltória convexa é um limitante superior para qualquer solução do MAXAP, isto produz uma $\frac{1}{2}$-aproximação para o problema, e esse fator de aproximação é justo para o algoritmo proposto. Fekete também provou ser NP-Difícil encontrar uma $(\frac{2}{3}+\epsilon)$-aproximação, para qualquer $\epsilon \in (0, \frac{1}{3})$.

3.2 Trabalhos anteriores

Métodos para geração e contagem de polígonos simples para n pontos no plano têm sido amplamente estudados na literatura. No entanto, poucos trabalhos abordam a geração de polígonos simples com área mínima ou máxima. Alguns autores estudam a contagem de poligonizações simples, buscando estabelecer limites inferior e superior (veja [18, 33]). Outros consideram tipos especiais de poligonizações simples, conhecidas como *Monotone Polygons* e *Starshaped Polygons* [23, 10]. As principais referências para o problema da poligonização são os trabalhos de Fekete [12, 15, 13].

Trabalhos mais recentes voltaram a apresentar estudos sobre o MINAP e o MAXAP, como pode ser visto em [35, 26], intitulados respectivamente por *Approaching Minimum Area Polygonization* e *An Empirical Study on Randomized Optimal Area Polygonization of Planar Point Sets*. Um breve descrição destes trabalhos é dada a seguir.

O trabalho de Taranilla, Gagliardi e Hernández [35] aborda o MINAP. Os autores

apresentam um algoritmo para construir um polígono simples, baseado na heurística *Steady Growth* proposta por Auer e Held [2] para geração de polígonos simples aleatórios. Eles descrevem seis diferentes estratégias para minimizar a área do polígono durante sua construção. Além disso, são propostos três algoritmos para encontrar soluções heurísticas para o MINAP, sendo um deles a metaheurística *Ant Colony Optimization* - ACO. Por fim os autores apresentam os experimentos computacionais e os resultados obtidos por cada algoritmo.

Mais recentemente, Peethambaran, Parakkat e Muthuganapathy [26] publicaram um trabalho sobre os problemas MINAP e MAXAP. O estudo propõe uma heurística para encontrar uma solução para os problemas dado qualquer conjunto planar de pontos. Embora o pseudocódigo apresentado no trabalho se baseie na solução para o MINAP, os autores explicam que o mesmo pode ser aplicado ao MAXAP com uma simples modificação no código. O Algoritmo 1 apresenta o pseudocódigo descrito pelos autores já adaptado para maximizar a área do polígono.

Algoritmo 1: RAND_MAXAP(S,n)

Entrada: $S = (p^0, p^1, ..., p^{n-1}) \rightarrow$ conjunto de n pontos planares
Saída: $\mathcal{P} \rightarrow$ polígono simples com área máxima aproximada
// P_2 = conjunto de pontos iniciais do polígono
// P_i = conjunto de pontos do polígono na i-ésima iteração
// E_2 = conjunto de arestas iniciais do polígono
// E_i = conjunto de arestas do polígono na i-ésima iteração
// *line sweep* = algoritmo clássico para descobrir interseção entre arestas
// *hull_size* = número de pontos no polígono
1 **início**
2 Escolha aleatoriamente três pontos a partir de S. Sejam eles p_0, p_1 e p_2;
3 $P_2 \leftarrow \{p_0, p_1, p_2\}$;
4 $E_2 \leftarrow \{(p_0, p_1), (p_1, p_2), (p_2, p_0)\}$;
5 $hull\_size \leftarrow 3$;
6 **para** (i $\leftarrow$ 3 **até** $n-1$) **faça**
7 Selecione aleatoriamente um ponto p_i a partir de $S \setminus P_{i-1}$;
8 Construa L consistindo de todas as arestas formadas por $p_i \cup P_{i-1}$;
9 Aplique o *line sweep* em $E_{i-1} \cup L$ e atualize L com as arestas sem interseção;
10 **se** (p_i está no interior de P_{i-1}) **então**
11 Percorra P_{i-1} e remova o triângulo $\Delta_{p_q p_i p_r}$ de área mínima;
12 **senão**
13 Percorra P_{i-1} e adicione o triângulo $\Delta_{p_q p_i p_r}$ de área máxima;
14 $L \leftarrow \{\emptyset\}$;
15 $hull\_size \leftarrow hull\_size + 1$;
16 $P_i \leftarrow \{p_0, p_1, p_2, ..., p_q, p_i, p_r, ..., p_{(hull\_size-1)}\}$;
17 $E_i \leftarrow \{(p_0, p_1), (p_1, p_2), (p_2, p_3), ..., (p_q, p_i), (p_i, p_r)\}$;
18 **retorna** P_{n-1} e E_{n-1};

Adaptado de [26].

O primeiro passo do algoritmo é selecionar três pontos aleatórios de S e construir um triângulo que representa o polígono inicial P_2, com o conjunto de arestas $E_2 = \{e_0, e_1, e_2\}$ (linhas 2 à 5). Na i-ésima iteração, *hull_size* representa o número de pontos de P_{i-1}. No restante das $n-3$ iterações, o algoritmo realiza os seguintes passos (linhas 6 à 17) [26]:

1. Selecione aleatoriamente um ponto p_i a partir de $S \setminus P_{i-1}$.

2. Verifique se o ponto se encontra no interior ou exterior de P_{i-1}.

3. Se o ponto está no interior de P_{i-1}, encontre um triângulo $\Delta_i = \{p_q, p_i, p_r\}$ de menor área que o ponto p_i forma com as arestas de P_{i-1}, tal que Δ_i não intercepta P_{i-1} e $p_q, p_r \in P_{i-1}$. Remova Δ_i de P_{i-1} adicionando as arestas e o ponto p_i na posição correta.

4. Se o ponto está no exterior de P_{i-1}, encontre um triângulo $\Delta_i = \{p_q, p_i, p_r\}$ de maior área que o ponto p_i forma com as arestas de P_{i-1}, tal que Δ_i não intercepta P_{i-1} e $p_q, p_r \in P_{i-1}$. Adicione Δ_i a P_{i-1} adicionando as arestas e o ponto p_i na posição correta.

5. Atualize os conjuntos de arestas e vértices como $E_i = E_{i-1} \cup \{p_q, p_i\} \cup \{p_i, p_r\} \setminus \{p_q, p_r\}$ e $P_i = P_{i-1} \cup p_i$.

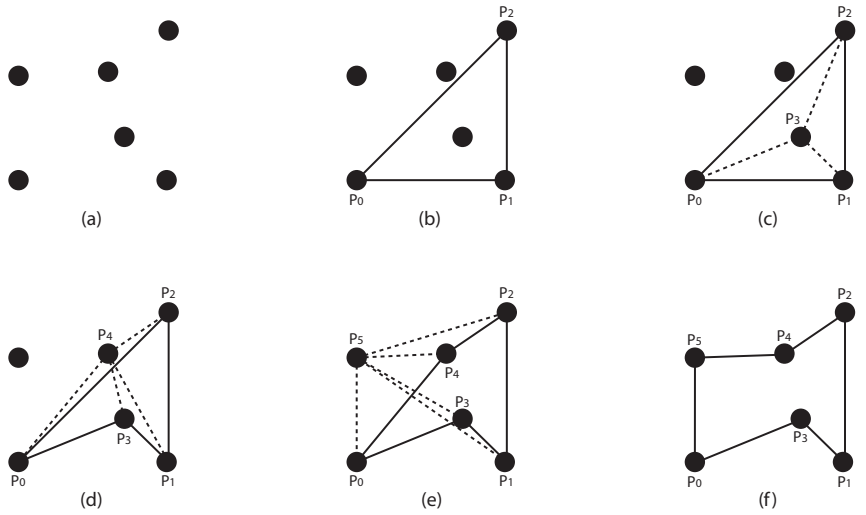


Figura 3.5: Passo a passo da heurística. (a) Conjunto de pontos; (b) Triângulo aleatório inicial; (c)-(e) Construção incremental; (f) Polígono final. Adaptado de [26].

O pseudocódigo do algoritmo para construir uma solução minimizando a área é o mesmo. A única diferença está na remoção ou inclusão de triângulos em cada iteração. Ao invés de remover o triângulo de área mínima caso o ponto esteja no interior do polígono atual, no MINAP remove-se o de área máxima. Similarmente, se o ponto é exterior ao polígono atual, a versão de minimização inclui o triângulo de área mínima.

Os autores provam a complexidade $O(n^2 \log n)$ da heurística e, por fim, apresentam um estudo experimental comparando ambas abordagens em instâncias criadas aleatoriamente e também algumas fornecidas pela *Traveling Salesman Problem Library* (TSPLIB).

Mais detalhes sobre os trabalhos descritos nesta seção podem ser vistos em [35, 26].

Capítulo 4

Metodologias de solução

Este capítulo descreve as metodologias de solução para o MAXAP investigadas neste trabalho. A Seção 4.1 apresenta o algoritmo aproximado proposto por Fekete; na Seção 4.2 são propostos dois modelos matemáticos para a solução exata do MAXAP, ambos formulados em PLI; por fim, a Seção 4.3 apresenta três novas abordagens de solução para o MAXAP: duas heurísticas e uma metaheurística.

4.1 Algoritmo aproximado

O algoritmo $\frac{1}{2}$-aproximado proposto por Fekete foi a primeira metodologia de solução investigada neste trabalho. Como não existem resultados práticos na literatura, um dos objetivos e também o ponto de partida desta pesquisa era o estudo computacional do mesmo. A seguir será descrito em mais detalhes seu funcionamento.

O algoritmo $\frac{1}{2}$-aproximado resulta da demonstração do seguinte teorema proposto por Fekete [12]:

Teorema 3 *Seja P um conjunto de $n + 1$ pontos no plano. Podemos determinar um polígono simples $\mathcal{P}$ com vértices em P cuja área é maior do que metade da área da envoltória convexa de P. Isto pode ser feito em tempo $O(n\ log\ n)$.*

Prova [12]: Seja p_0 um ponto na envoltória convexa de P. Em tempo $O(n\ log\ n)$, ordene os pontos p_i de P pelo coeficiente angular da linha $l(p_0, p_i)$, de modo que os vizinhos de p_0 na envoltória convexa de P ($\mathcal{H} = convexHull(P)$) sejam o primeiro ponto (p_1) e último ponto (p_n) respectivamente, ou seja, $\mathcal{H} = \langle p_0, p_1, ..., p_n, p_0 \rangle$. Se existe um conjunto de pontos para os quais o coeficiente angular é o mesmo, ordene-os de forma crescente pela distância de p_0, exceto quando tais pontos possuem coeficiente angular mínimo, caso em que deve-se ordená-los de modo decrescente pela distância de p_0. Ao conectar os pontos p_i nesta ordem, gera-se um polígono simples $\mathcal{P}_1$ a partir de P, como mostra a Figura 4.1.

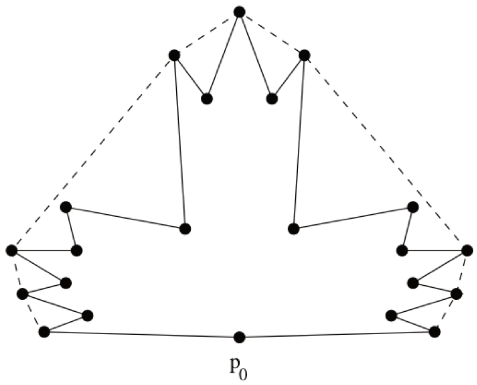


Figura 4.1: Polígono simples $\mathcal{P}_1$ gerado a partir de P [13].

Se a $area(\mathcal{P}_1) > \frac{1}{2}area(\mathcal{H})$, o algoritmo terminou. Caso contrário, a família de polígonos $\mathcal{Q} = \mathcal{H} \setminus \mathcal{P}_1$, resultante da subtração da envoltória convexa de P pelo polígono $\mathcal{P}_1$, tem área maior ou igual a $\frac{1}{2}area(\mathcal{H})$.

Seja $\overline{P} \subseteq P$ o conjunto de pontos da borda de $\mathcal{H}$. Referimo-nos aos pontos q_i contidos em $\overline{P}$ como $\overline{q}_i$. A família $\mathcal{Q} = (Q_1, ..., Q_h)$ consiste de h polígonos, tal que $Q_j := \langle \overline{q}_1^{(j)}, q_2^{(j)}, ..., q_{z_j-1}^{(j)}, \overline{q}_{z_j}^{(j)}, \overline{q}_1^{(j)} \rangle$, onde $\overline{q}_1^{(j)}$ e $\overline{q}_{z_j}^{(j)}$ são os únicos vértices de Q_j que pertencem a $\overline{P}$.

Fekete [12] mostra que o polígono

$$\mathcal{P}_2 = \langle \overline{p}_0, q_2^{(1)}, ..., q_{z_1-1}^{(1)}, q_2^{(2)} ..., q_{z_j-1}^{(j)}, q_2^{(j+1)}, ..., q_2^{(h)}, ..., q_{z_h-1}^{(h)}, \overline{p}_n, ..., \overline{p}_1, \overline{p}_0 \rangle$$

é simples. Em termos mais claros, o polígono $\mathcal{P}_2$ é obtido percorrendo, na mesma ordem definida por $\mathcal{P}_1$, todos os pontos no interior de $\mathcal{H}$ e, em seguida, voltando ao longo dos pontos de $\mathcal{H}$ (Figura 4.2), na ordem inversa definida por $\mathcal{P}_1$.

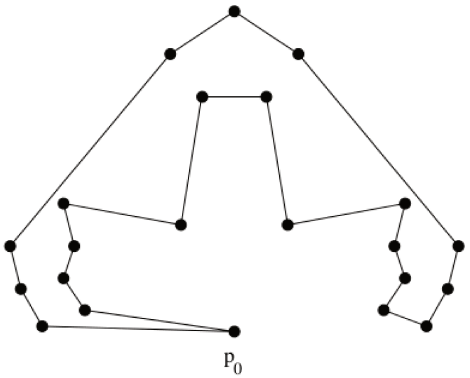


Figura 4.2: O polígono simples $\mathcal{P}_2$ [13].

Uma vez que $\mathcal{P}_2$ contém todos os polígonos de $\mathcal{Q}$ se $area(\mathcal{P}_1) < \frac{1}{2}area(\mathcal{H})$, então $area(\mathcal{P}_2) \geqslant \frac{1}{2}area(\mathcal{H})$, concluindo a prova. □

Vale ressaltar que não existe um estudo empírico que sugere quais seriam as propriedades que tornam um vértice da envoltória uma boa escolha como ponto de partida para o algoritmo. Inclusive, Fekete [12] mostrou que o fator aproximativo do algoritmo não pode ser melhorado mesmo tentando todos os possíveis pontos de partida p_0 da envoltória convexa.

O Algoritmo 2 ilustra o pseudocódigo do algoritmo $\frac{1}{2}$-aproximado proposto por Fekete [12].

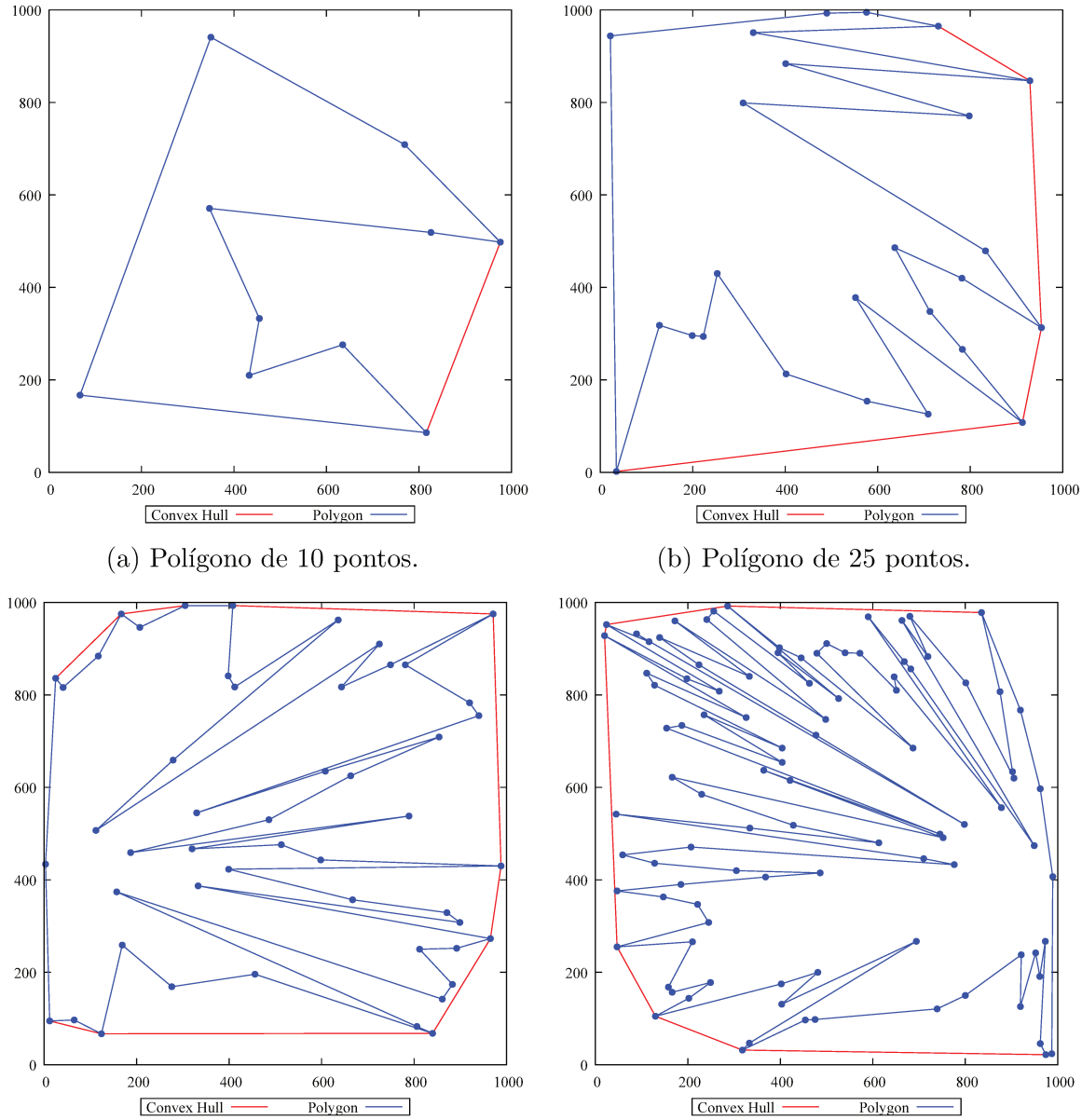
Algoritmo 2: MAXAP_Aproximado

Entrada: $P = (p_0, p_1, ..., p_n) \rightarrow$ conjunto de $n+1$ pontos no plano
Saída: $\mathcal{P} \rightarrow$ polígono simples tal que $area(\mathcal{P}) \geq \frac{area(\mathcal{H})}{2}$

início
  $\mathcal{H} = \langle \overline{p}_0, \overline{p}_1, ..., \overline{p}_i, \overline{p}_0 \rangle \leftarrow$ convexHull(P);
  Seja $\overline{p}_0$ algum ponto em $\mathcal{H}$;
  Ordene os p_i pontos em ordem crescente de $slope(\overline{p}_0, p_i)$. Em caso de empate, se o *slope* for mínimo, ordene em ordem decrescente de $dist(\overline{p}_0, p_i)$; caso contrário, ordene em ordem crescente de $dist(\overline{p}_0, p_i)$;
  $\mathcal{P}_1 \leftarrow \langle \overline{p}_0, p_1, ...p_n, \overline{p}_0 \rangle$;
  se $\left(area(\mathcal{P}_1) < \frac{area(\mathcal{H})}{2}\right)$ **então**
    $\mathcal{Q} = (Q_1, ..., Q_h) \leftarrow \mathcal{H} \setminus \mathcal{P}_1$; // $\mathcal{Q}$ é um conjunto de h polígonos
    para (j $\leftarrow 1$ **até** h) **faça**
      Ordene os vértices de $Q_j = \langle \overline{q}_1^j, q_2^j, ..., q_{z_j-1}^j, \overline{q}_{z_j}^j, \overline{q}_1^j \rangle$ de modo que $\overline{q}_1^j, \overline{q}_{z_j}^j \in \mathcal{H}$;
    $\mathcal{P}_2 \leftarrow \langle \overline{p}_0, q_2^1, ..., q_{z_1-1}^1, ..., q_{z_j-1}^j, q_2^{j+1}, q_2^h, ..., q_{z_h-1}^h, \overline{p}_n, ..., \overline{p}_1, \overline{p}_0 \rangle$;
    retorna $\mathcal{P}_2$;
  retorna $\mathcal{P}_1$;

Adaptado de [12].

A Figura 4.3 mostra alguns exemplos de polígonos gerados pelo algoritmo aproximado.



(a) Polígono de 10 pontos.

(b) Polígono de 25 pontos.

(c) Polígono de 50 pontos.

(d) Polígono de 100 pontos.

Figura 4.3: Exemplos de polígonos gerados pelo algoritmo aproximado.

4.2 Modelos matemáticos

Este trabalho propõe duas formulações PLI para o MAXAP, a primeira baseia-se na fórmula para cálculo de área de um polígono simples, e a segunda é baseada no cálculo de área por triangulação, conforme visto na Seção 3.1.

4.2.1 Modelo baseado no cálculo de área de um polígono simples

Baseado na equação 3.2, propõem-se o seguinte modelo para o MAXAP:

$$(MAXAP)$$

$$MAX \quad \frac{1}{2}\sum_{ij\in E}(a_i b_j - a_j b_i)x_{ij} \tag{4.1}$$

s.a.

$$x_{ij} + x_{ji} \leq 1 \qquad ij, ji \in E \tag{4.2}$$

$$\sum_{j=1}^{|V|} x_{ij} = 1 \qquad i \in V \tag{4.3}$$

$$\sum_{i=1}^{|V|} x_{ij} = 1 \qquad j \in V \tag{4.4}$$

$$\sum_{i,j\in S} x_{ij} \leq |S| - 1 \qquad \forall S \subset V \tag{4.5}$$

$$x_{ij} + x_{uv} \leq 1 \qquad ((i,j),(u,v)) \in \mathcal{L} \tag{4.6}$$

$$x_{ij} \in \{0,1\} \qquad ij \in E \tag{4.7}$$

onde,

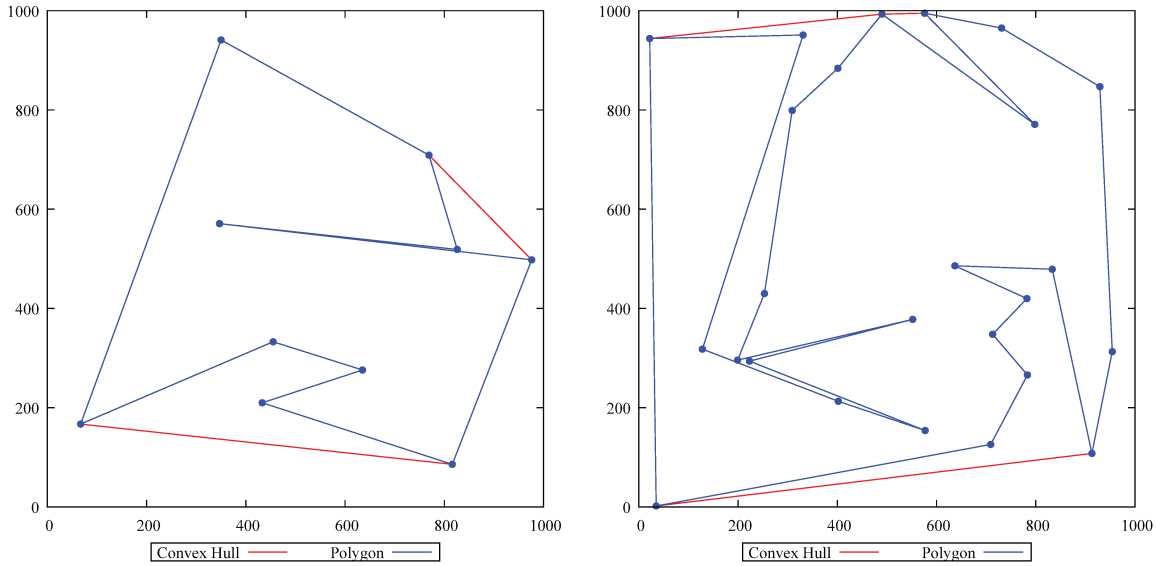
- $a_i b_i$ e $a_j b_j$ = coordenadas dos pontos que compõem o arco ij.
- $x_{ij} = 1$ se o arco (i,j) pertence à solução. $x_{ij} = 0$ caso contrário.
- $\mathcal{L}$ = família de pares de arcos que se interceptam.

A restrição 4.2 garante a unidirecionalidade da rota, ou seja, os arcos escolhidos para compor a solução devem seguir um único sentido (horário ou anti-horário). As restrições 4.3 e 4.4 asseguram a continuidade da rota, isto é, cada vértice deve possuir apenas um arco de entrada e um arco de saída. A restrição 4.5 garante a eliminação de subciclos, e a restrição 4.6 define que não há cruzamento de arcos (polígono simples).

Em 2015, Fekete *et al.* [11] propuseram um modelo semelhante, apresentando resultados computacionais e discutindo, inclusive, melhorias que evidenciaram ganhos práticos. Vale ressaltar que, apesar dos autores apresentarem esta formulação recentemente, o modelo proposto neste trabalho foi desenvolvido de maneira independente.

Cabe mencionar que se tentou inserir as restrições 4.6 sob demanda (*lazy constraints*), mas experimentos preliminares mostraram que essa estratégia não foi bem sucedida.

Dois exemplos de polígonos gerados pelo modelo são apresentados na Figura 4.4. A Figura 4.4b ilustra um polígono que não obteve certificado de otimalidade dentro do limite de tempo de execução.



(a) Polígono ótimo de 10 pontos. (b) Polígono de 25 pontos.

Figura 4.4: Exemplos de polígonos gerados pelo modelo.

4.2.2 Modelo baseado em triangulação

O modelo apresentado nesta seção utilizou como ponto de partida a formulação PLI aplicada ao problema da triangulação de menor peso (*Minimum Weight Triangulation Problem* – MWTP), cujo objetivo é encontrar uma triangulação que minimiza o peso das arestas dos triângulos [27].

Seja P um conjunto de pontos no plano. Um triângulo vazio com vértices em P é representado pela tripla ijk. Seja $\Delta(P)$ o conjunto de triângulos vazios sobre P, $L^+(ij)$ e $L^-(ij)$ são os dois semi-planos definidos pela reta contendo o segmento ij.

O modelo formulado para o MWTP é apresentado a seguir.

$$(MWTP)$$

$$MIN \quad \sum_{ijk \in \Delta(P)} c_{ijk} t_{ijk} \tag{4.8}$$

s.a.

$$\sum_{ijk \in \Delta(P)} t_{ijk} = 1 \qquad ij \in E_H \tag{4.9}$$

$$\sum_{\substack{ijk \in \Delta(P) \\ ijk \subset L^+(ij)}} t_{ijk} = \sum_{\substack{ijk \in \Delta(P) \\ ijk \subset L^-(ij)}} t_{ijk} \qquad ij \in E \setminus E_H \tag{4.10}$$

$$t_{ijk} \in \{0, 1\} \qquad ijk \in \Delta(P) \tag{4.11}$$

onde,

- c_{ijk} = peso do triângulo ijk.
- $t_{ijk} = 1$ se ijk pertence à triangulação, $t_{ijk} = 0$ caso contrário.

Visto que as arestas na envoltória convexa E_H estão presentes em cada triangulação planar, a restrição 4.9 garante que um triângulo contendo uma dessas arestas está na triangulação. A restrição 4.10 define que o número de triângulos contendo uma aresta ij deve ser o mesmo em ambos os semi-planos $L^+(ij)$ e $L^-(ij)$.

A partir do modelo para o MWTP, propõem-se o seguinte modelo para o MAXAP:

$$(MAXAP)$$

$$MAX \quad \sum_{ijk \in \Delta(P)} a_{ijk} t_{ijk} \tag{4.12}$$

s.a.

$$x_{ij} \geq \sum_{\substack{ijk \in \Delta(P) \\ ijk \subset L^+(ij)}} t_{ijk} - \sum_{\substack{ijk \in \Delta(P) \\ ijk \subset L^-(ij)}} t_{ijk} \qquad ij \in E \setminus E_H \tag{4.13}$$

$$x_{ij} \geq \sum_{\substack{ijk \in \Delta(P) \\ ijk \subset L^-(ij)}} t_{ijk} - \sum_{\substack{ijk \in \Delta(P) \\ ijk \subset L^+(ij)}} t_{ijk} \qquad ij \in E \setminus E_H \tag{4.14}$$

$$\sum_{ijk \in \Delta(P)} t_{ijk} \geq x_{ij} \qquad ij \in E \setminus E_H \tag{4.15}$$

$$\sum_{ijk \in \Delta(P)} t_{ijk} \leq 2 - x_{ij} \qquad ij \in E \setminus E_H \tag{4.16}$$

$$\sum_{ijk \in \Delta(P)} t_{ijk} = x_{ij} \qquad ij \in E_H \tag{4.17}$$

$$t_{ijk} + t_{i'j'k'} \leq 1 \qquad (ijk, i'j'k') \in T \tag{4.18}$$

$$\sum_{i,j \in S} x_{ij} \leq |S| - 1 \qquad \forall S \subset P \tag{4.19}$$

$$\sum_{ij \in \delta(i)} x_{ij} = 2 \qquad i \in P \tag{4.20}$$

$$t_{ijk} \in \{0, 1\} \qquad ijk \in \Delta(P) \tag{4.21}$$

$$x_{ij} \in \{0, 1\} \qquad ij \in E \tag{4.22}$$

onde,

- a_{ijk} = área do triângulo ijk.
- $t_{ijk} = 1$ se ijk pertence à solução, 0 caso contrário.
- $x_{ij} = 1$ se a aresta (i, j) pertence à solução, $x_{ij} = 0$
- T = família de pares de triângulos vazios que se interceptam.
- $\delta(i)$ = família de arestas não direcionadas ligadas ao vértice i.

As restrições 4.13 e 4.14 garantem que, para cada aresta (i, j) não pertencente à envoltória convexa, se $x_{ij} = 0$, a quantidade de triângulos no semi-plano $L^+(ij)$ presentes

na solução é igual à quantidade de triângulos no semi-plano $L^-(ij)$. Caso contrário, se $x_{ij} = 1$, a quantidade destes triângulos nos dois semi-planos pode diferir em uma unidade.

As restrições 4.15 e 4.16 garantem que, para cada aresta (i, j) não pertencente à envoltória convexa, se $x_{ij} = 1$, então a solução deve conter estritamente um triângulo $ijk \in \Delta(P)$. Caso contrário, se $x_{ij} = 0$, então a solução pode conter no máximo dois triângulos $ijk \in \Delta(P)$ e $ijk' \in \Delta(P)$ tal que $k \neq k'$.

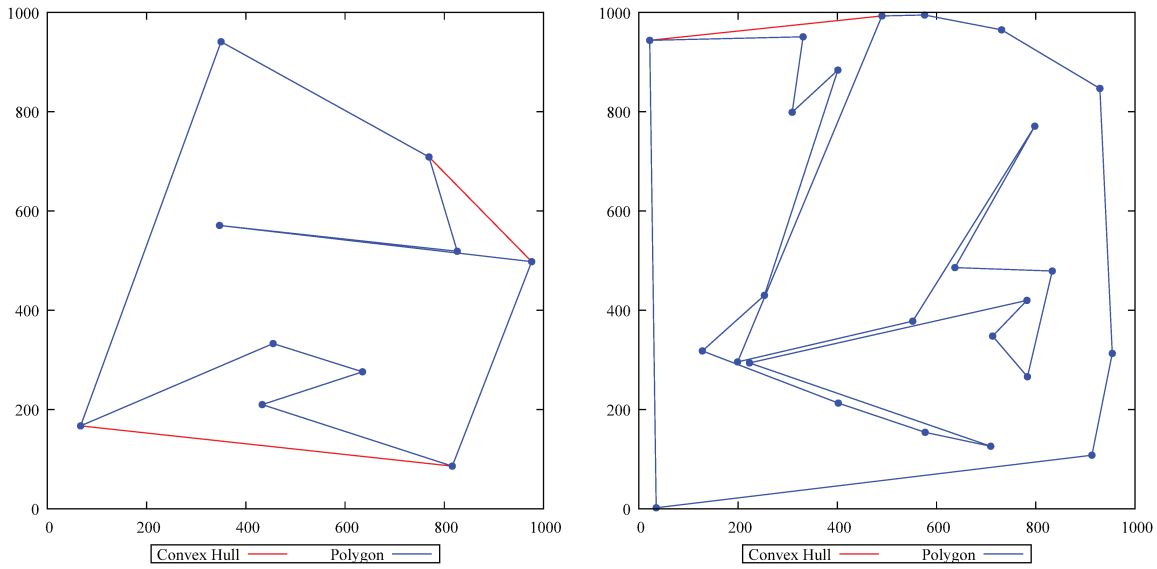
A restrição 4.17 assegura que se a aresta (i, j) da envoltória convexa pertence à solução, então a solução deve conter estritamente um triângulo $ijk \in \Delta(P)$.

A restrição 4.18 garante que, para cada par de triângulos vazios que se interceptam, no máximo um deles pode estar na solução.

A restrição 4.19 define que a solução deve conter apenas um polígono simples (restrição para eliminação de sub-ciclos).

Finalmente, a restrição 4.20 garante que o polígono gera todos os vértices, ou seja, cada vértice deve ter estritamente duas arestas.

A Figura 4.5 apresenta dois polígonos gerados pelo modelo descrito acima. O polígono da Figura 4.5b não obteve certificado de otimalidade dentro do limite de tempo de execução.



(a) Polígono ótimo de 10 pontos. (b) Polígono de 25 pontos.

Figura 4.5: Exemplos de polígonos gerados pelo modelo.

4.3 Heurísticas e metaheurística

Um dos objetivos deste trabalho é propor, além dos modelos PLI, metodologias heurísticas para solução de instâncias intratáveis por metodologias exatas. As Seções 4.3.1, 4.3.2 e 4.3.3 apresentam três novas metodologias para solução do MAXAP, duas heurísticas construtivas e uma metaheurística, respectivamente.

4.3.1 Heurística baseada em triangulação

A primeira heurística proposta trata-se de uma heurística construtiva gulosa, a qual baseia-se na adição de triângulos vazios de área máxima. É necessário um pré-processamento para encontrar o conjunto de triângulos vazios que compõem a instância (Figura 4.6). O polígono inicia com o triângulo vazio de maior área. Uma vez que o triângulo vazio inicial foi escolhido, para garantir que o polígono gerado seja simples, durante $n-3$ iterações faz-se a adição do triângulo vazio de maior área adjacente a alguma das arestas do polígono em construção, e todos aqueles que cruzam alguma aresta do polígono atual são removidos do conjunto de triângulos vazios elegíveis. A heurística itera $n-3$ vezes visto que é conhecido que toda triangulação de um polígono de n lados possui $n-2$ triângulos.

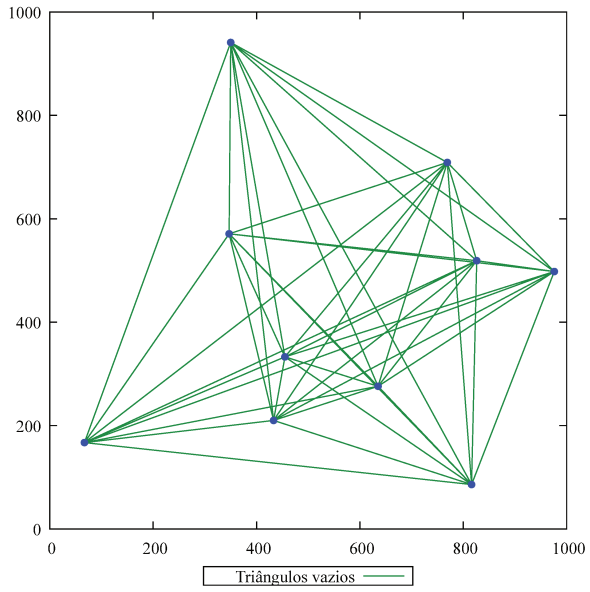
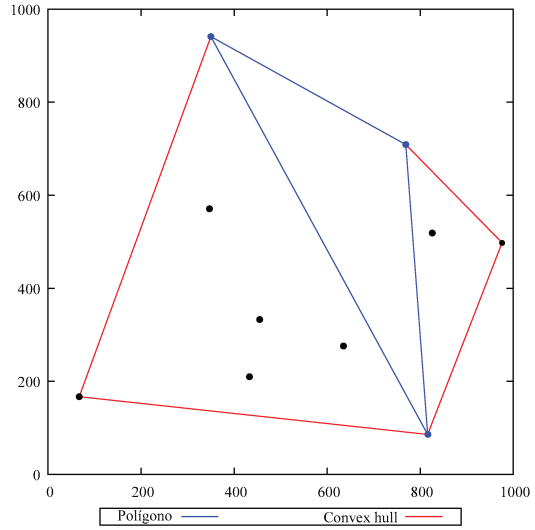
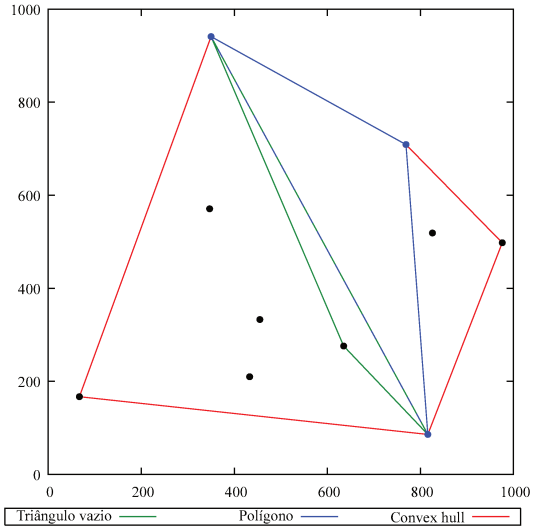


Figura 4.6: Conjunto de arestas pertencentes a triângulos vazios para uma instância de 10 pontos.

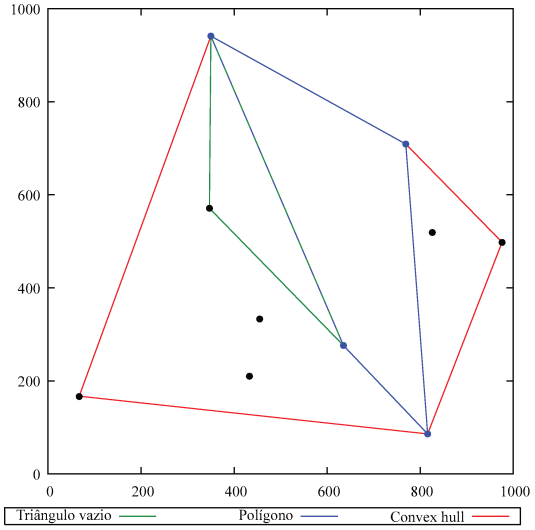
As Figuras 4.7 e 4.8 ilustram o funcionamento do algoritmo a cada iteração para uma instância de tamanho 10. O polígono final resultante é mostrado na Figura 4.9.



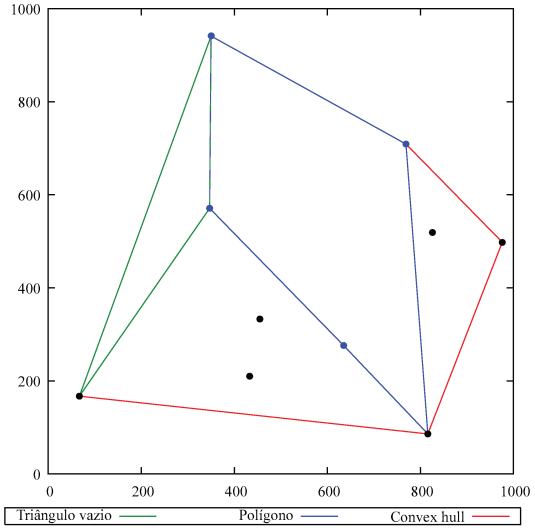
(a) Polígono inicial.



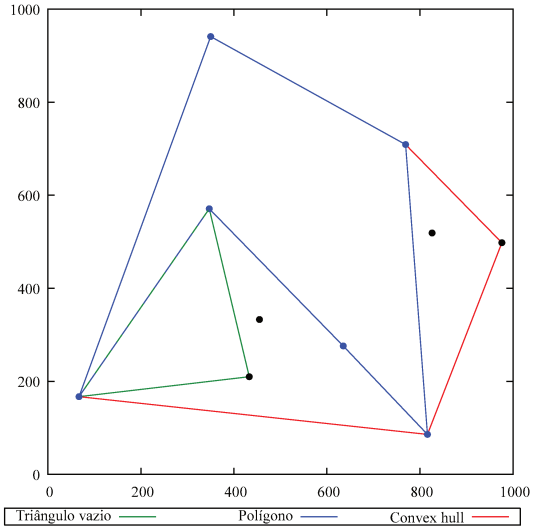
(b) Iteração 1.



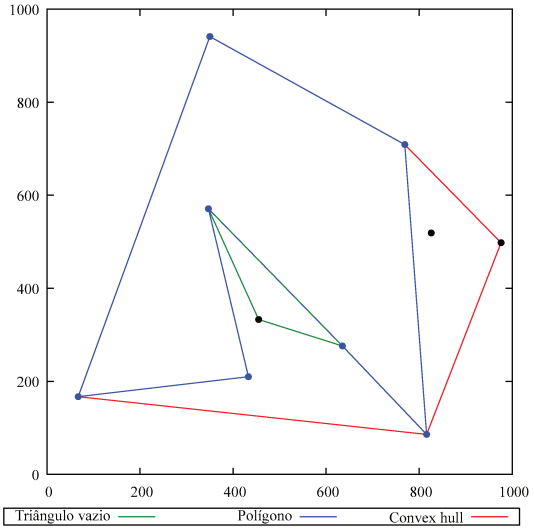
(c) Iteração 2.



(d) Iteração 3.

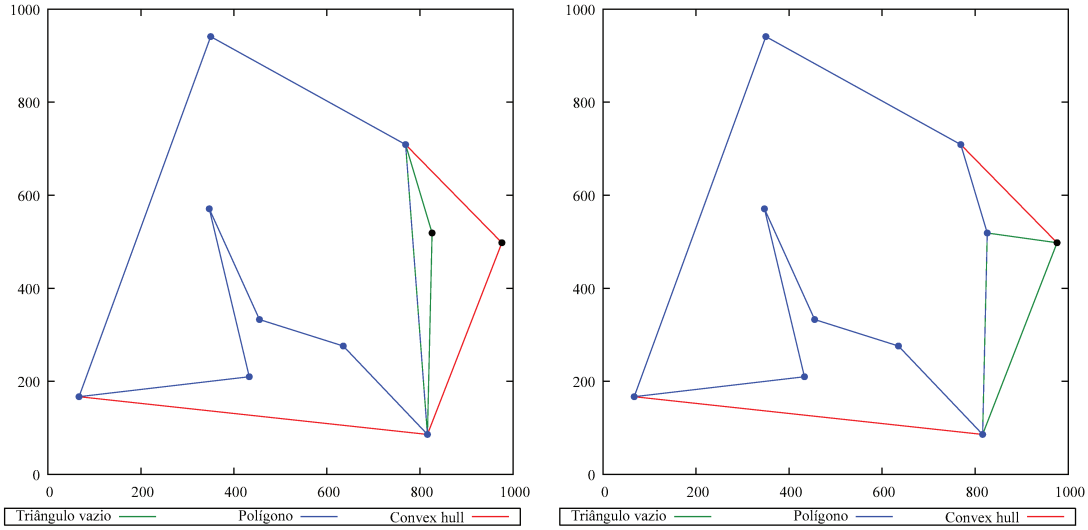


(e) Iteração 4.



(f) Iteração 5.

Figura 4.7: Passo a passo da heurística para uma instância de 10 pontos.



(a) Iteração 6. (b) Iteração 7.

Figura 4.8: Passo a passo da heurística para uma instância de 10 pontos (cont.).

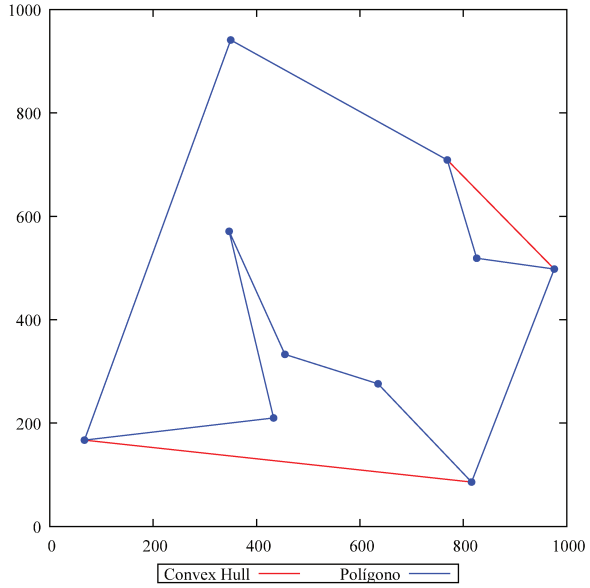


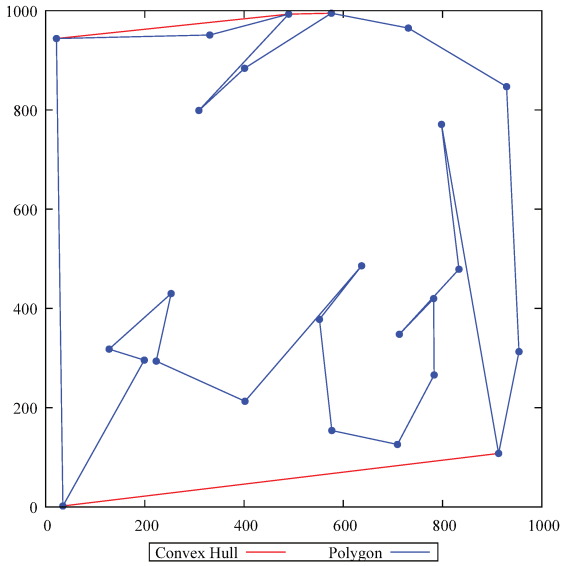
Figura 4.9: Polígono final gerado.

O Algoritmo 3 apresenta o pseudocódigo da heurística.

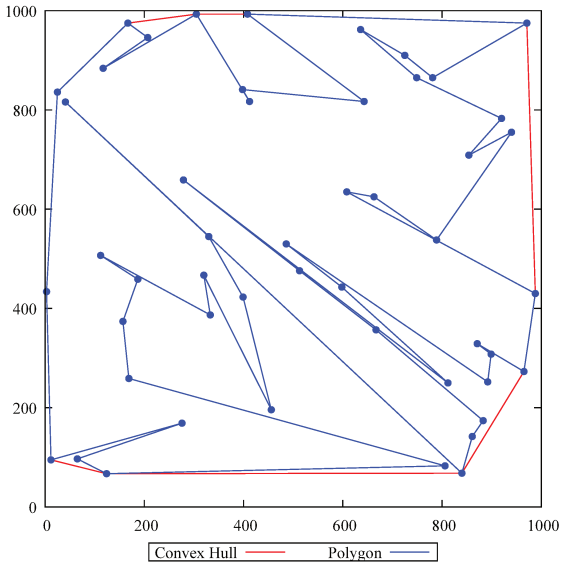
Algoritmo 3: MAXAP_TriangulosVazios(P)

Entrada: $P = (p_1, p_2, ..., p_n) \rightarrow$ conjunto de n pontos no plano
Saída: $\mathcal{P} \rightarrow$ polígono simples
// $\Delta(P)$ = conjunto de todos os triângulos vazios sobre P
// $best_\Delta$ = triângulo de maior área a ser adicionado a $\mathcal{P}$
// $\Delta_{cross}(best_\Delta)$ = triângulos vazios que cruzam $best_\Delta$
// $\mathcal{N}_\mathcal{T}(\mathcal{P})$ = conjunto de triângulos vazios em $\mathcal{T}$ adjacentes a $\mathcal{P}$
1 **início**
2 $\quad \mathcal{P} \leftarrow \emptyset$;
3 $\quad \mathcal{T} \leftarrow \Delta(P)$;
4 $\quad best_\Delta \leftarrow \arg\max_\Delta \{Area(\Delta) \mid \Delta \in \Delta(P)\}$;
5 $\quad \mathcal{P} \leftarrow \mathcal{P} \cup \{best_\Delta\}$;
6 $\quad \mathcal{T} \leftarrow \mathcal{T} \setminus \{\Delta \in \mathcal{T} \mid \Delta \in \Delta_{cross}(best_\Delta)\}$;
7 $\quad \mathcal{T} \leftarrow \mathcal{T} \setminus \{best_\Delta\}$;
8 $\quad$ **enquanto** $|\mathcal{P}| < n - 2$ **faça**
9 $\quad\quad best_\Delta \leftarrow \arg\max_\Delta \{Area(\Delta) \mid \Delta \in \mathcal{N}_\mathcal{T}(\mathcal{P})\}$;
10 $\quad\quad \mathcal{P} \leftarrow \mathcal{P} \cup \{best_\Delta\}$;
11 $\quad\quad \mathcal{T} \leftarrow \mathcal{T} \setminus \{\Delta \in \mathcal{T} \mid \Delta \in \Delta_{cross}(best_\Delta)\}$;
12 $\quad\quad \mathcal{T} \leftarrow \mathcal{T} \setminus \{best_\Delta\}$;
13 $\quad$ **retorna** $\mathcal{P}$;

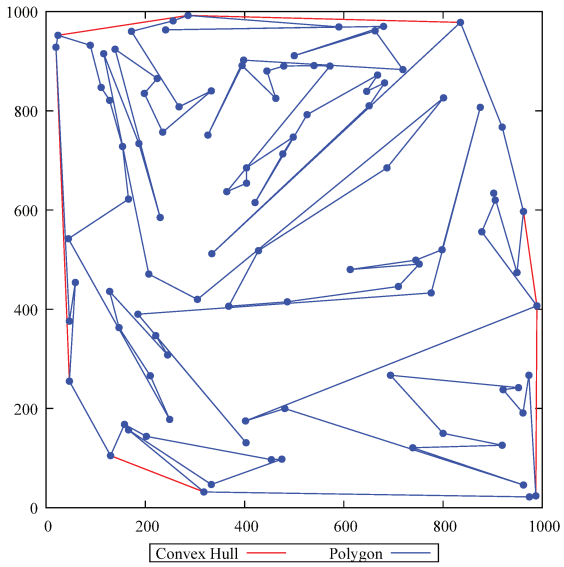
As linhas 3 a 7 referem-se à adição do primeiro triângulo à solução. O laço de repetição das linhas 8 a 12 adicionam o triângulo de maior área adjacente a $\mathcal{P}$ a cada iteração. O laço termina quando o tamanho do polígono (número de triângulos) for igual à quantidade de vértices menos 2. A adição somente de triângulos vazios adjacentes a $\mathcal{P}$ (linhas 9 e 10) e a remoção daqueles que interceptam o triângulo adicionado (linhas 6 e 11) garantem que o polígono resultante será simples. A Figura 4.10 apresenta exemplos de soluções encontradas pela heurística.



(a) Polígono de 25 pontos.



(b) Polígono de 50 pontos.



(c) Polígono de 100 pontos.

Figura 4.10: Exemplos de soluções obtidas pela heurística de triangulação.

Embora o RAND_MAXAP (Algoritmo 1) possua algumas semelhanças com a heurística descrita acima, é importante ressaltar que a mesma foi elaborada de maneira independente. Mais relevante que isso está o fato do RAND_MAXAP trabalhar com triângulos não necessariamente vazios, enquanto que o MAXAP_TriangulosVazios (Algoritmo 3) trabalha apenas com triângulos vazios.

Análise de complexidade

A complexidade de tempo do algoritmo, no pior caso, é $O(n^4)$. Isso decorre do fato de que o laço da linha 8 repete-se por $\Theta(n)$ iterações, e a cada iteração realiza-se uma busca pela lista de triângulos vazios $O(n^3)$.

Uma vez que é necessário armazenar os triângulos vazios, a complexidade de espaço

do algoritmo é $O(n^3)$.

4.3.2 Heurística baseada em envoltórias convexas

A segunda heurística proposta também é uma heurística construtiva gulosa, e baseia-se na concatenação de uma família de envoltórias convexas que compõem o conjunto de vértices. A geração dessa família de envoltórias convexas consiste em iterativamente gerar a envoltória convexa dos vértices atuais e remover dos vértices atuais aqueles que formam a última envoltória convexa. Esse processo é chamado de *onion peeling*, e o conjunto de envoltórias convexas resultante é conhecido como *onion decomposition* [22]. Em contraste com a heurística de triangulação, o polígono é construído de fora para dentro, isto é, inicia-se da envoltória convexa mais externa para a mais interna. O polígono inicial é formado pelas arestas da envoltória convexa mais externa (Figura 4.11).

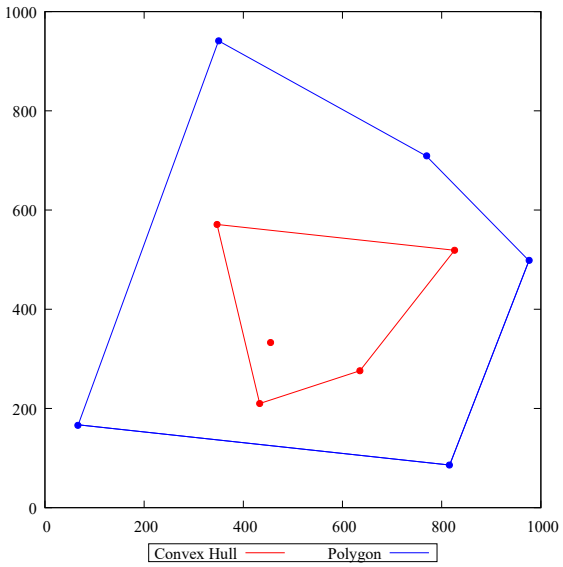
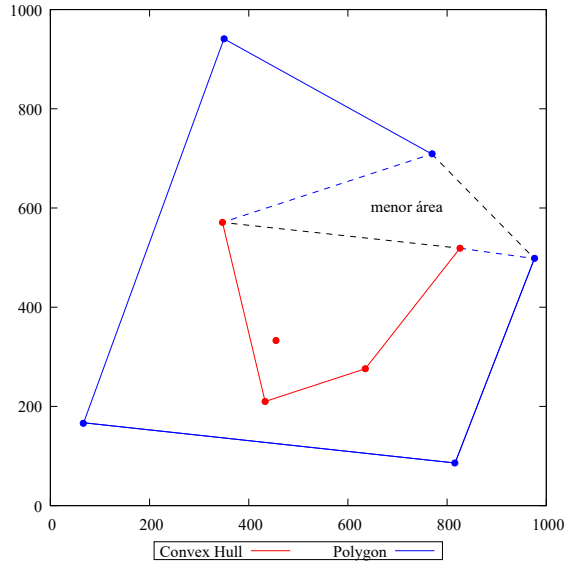
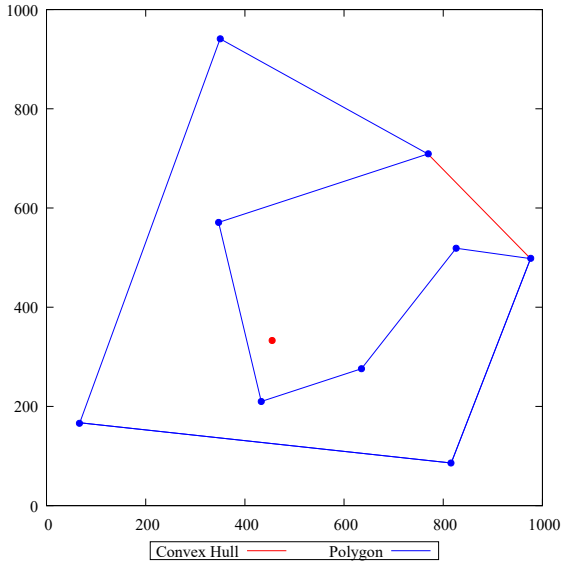


Figura 4.11: Conjunto das envoltórias convexas e polígono inicial.

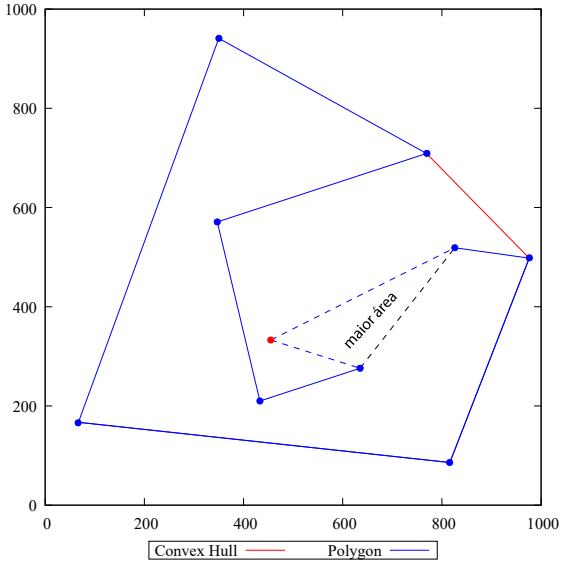
A cada iteração o algoritmo concatena duas envoltórias convexas, a externa e a interna. O critério guloso está na escolha dos pares de arestas que serão utilizadas para concatenar as duas envoltórias convexas. As arestas escolhidas devem pertencer a um setor de área mínima nas iterações ímpares (Figura 4.12a), e a um setor de área máxima nas iterações pares (Figura 4.12c). Isto é feito pois, nas iterações ímpares, a área do setor escolhido é excluída do polígono, enquanto que nas iterações pares ocorre a inclusão da área do setor no polígono. A Figura 4.12 exemplifica o funcionamento do algoritmo para a mesma instância utilizada como exemplo na seção anterior.



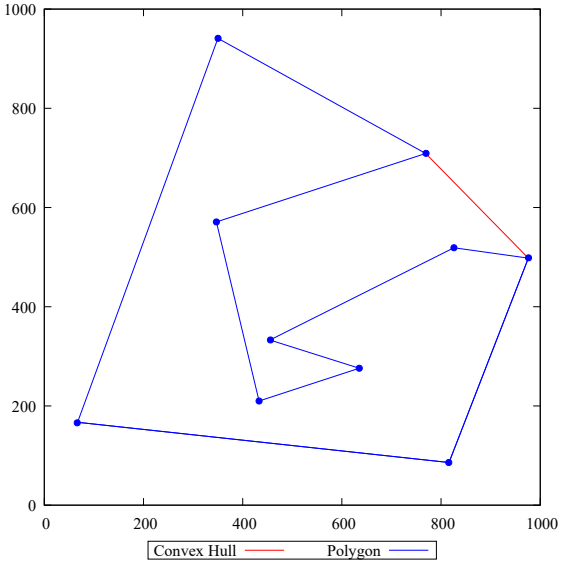
(a) Iteração 1 - Escolha do setor



(b) Iteração 1 - Atualização do polígono



(c) Iteração 2 - Escolha do setor



(d) Iteração 2 - Atualização do polígono

Figura 4.12: Passo a passo da heurística para uma instância de 10 pontos.

O pseudocódigo da heurística baseada em envoltórias convexas é apresentado no Algoritmo 4.

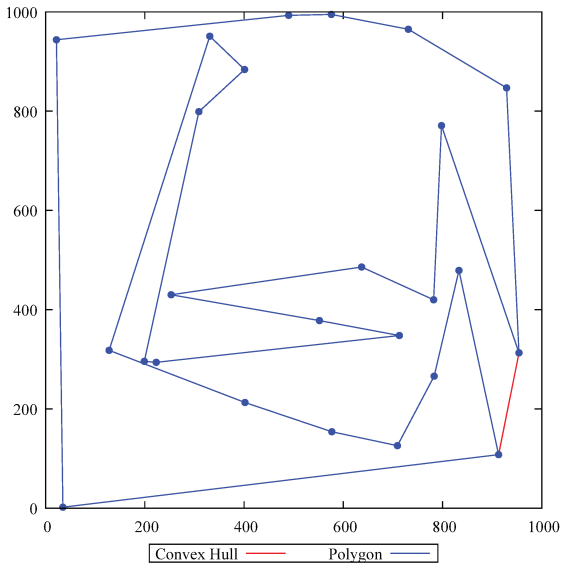
Algoritmo 4: MAXAP_ConvexHulls(P)

Entrada: $P = (p_1, p_2, ..., p_n) \rightarrow$ conjunto de n pontos no plano
Saída: $\mathcal{P} \rightarrow$ polígono simples
// $\mathcal{H}$ = conjunto de todas as envoltórias convexas sobre P

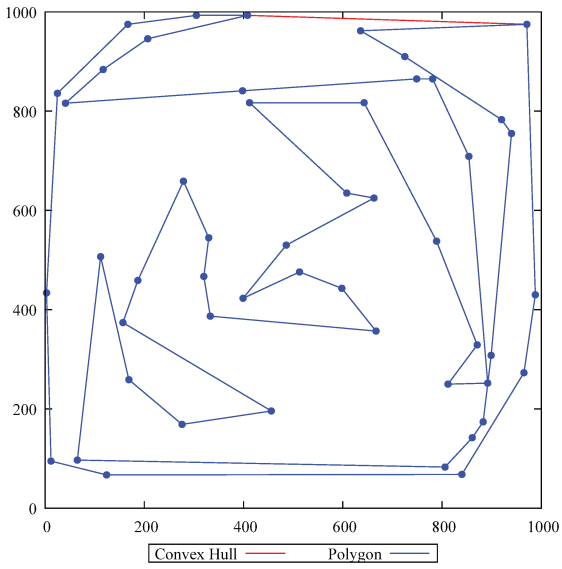
```
1  início
2      H ← conjunto de envoltórias convexas sobre P;
3      P ← H_1;
4      se (|H| = 1) então
5          retorna P;
6      para (i ← 1 até |H − 1|) faça
7          se (i % 2 ≠ 0) então
8              A_opt ← +∞;
9          senão
10             A_opt ← −∞;
11         para (e_1 ∈ {H_i ∩ P}) faça
12             para (e_2 ∈ H_{i+1}) faça
13                 e_3, e_4 ← arestas complementares a e_1 e e_2;
14                 se (e_3 e e_4 não interceptam H_{i+1}) então
15                     A ← Area(e_1, e_2, e_3, e_4);
16                     se (i % 2 ≠ 0) então
17                         se (A < A_opt) então
18                             A_opt ← A;
19                             best_e1 ← e_1;
20                             best_e2 ← e_2;
21                             best_e3 ← e_3;
22                             best_e4 ← e_4;
23                     senão
24                         se (A > A_opt) então
25                             A_opt ← A;
26                             best_e1 ← e_1;
27                             best_e2 ← e_2;
28                             best_e3 ← e_3;
29                             best_e4 ← e_4;
30         P ← {P \ {best_e1}} ∪ {H_{i+1} \ {best_e2}} ∪ {best_e3, best_e4};
31     retorna P;
```

A linha 2 encontra as envoltórias convexas do conjunto de pontos de entrada. O polígono $\mathcal{P}$ inicia-se com as arestas da envoltória convexa mais externa $\mathcal{H}_1$ (linha 3). Caso exista uma única envoltória convexa no conjunto de pontos, o algoritmo é encerrado e $\mathcal{P} = \mathcal{H}_1$ (linhas 4 e 5). O laço da linha 6 é responsável por realizar a concatenação das envoltórias convexas. O laço da linha 11 itera sobre cada aresta e_1 da envoltória convexa externa $\mathcal{H}_i$ que faz parte da solução, e o laço mais interno na linha 12 itera sobre as arestas e_2 da envoltória interna $\mathcal{H}_{i+1}$ (a que será concatenada). Na linha 13 obtêm-se as arestas e_3 e e_4 complementares a e_1 e e_2. Se e_3 e e_4 não interceptam a envoltória convexa interna (linha 14), calcula-se a área do setor formado por e_1, e_2, e_3 e e_4 (linha 15). Caso a iteração do laço mais externo seja ímpar, armazena-se o setor de área mínima, e o de área máxima

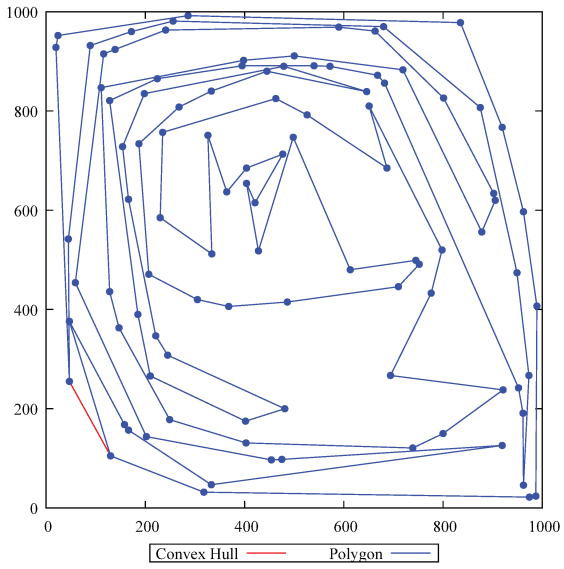
caso contrário (linhas 16 a 29). Ao final de uma iteração do laço da linha 6 as arestas complementares $best_{e3}$ e $best_{e4}$ e as pertencentes a $\mathcal{H}_{i+1} \setminus \{best_{e2}\}$ são incluídas em $\mathcal{P}$, e $best_{e1} \in \mathcal{P}$ é removida do polígono (linha 30). Alguns exemplos de soluções obtidas pelo algoritmo podem ser vistos na Figura 4.13.



(a) Polígono de 25 pontos.



(b) Polígono de 50 pontos.



(c) Polígono de 100 pontos.

Figura 4.13: Soluções obtidas pelo algoritmo *convex hulls*.

Análise de complexidade

Na linha 2 realiza-se o pré-processamento *onion peeling* para encontrar as envoltórias convexas no conjunto de pontos. Tal procedimento pode ser implementado de forma que sua complexidade de tempo no pior caso é $O(n \log n)$ [6].

Supondo que o número de arestas em cada envoltória convexa é $O(n)$, em contrapartida a quantidade de envoltórias convexas é constante. Neste sentido, o laço principal (linha

9) executa $O(1)$ vezes e cada um dos laços nas linhas 10 e 11 iteram sobre $O(n)$ arestas. Encontrar as arestas complementares (linha 12) é $O(1)$, e verificar se elas não interceptam a envoltória convexa interna (linha 13) pode ser feito em $O(log\, n)$ [7], o que resulta em uma complexidade $O(n\, log\, n)$ no pior caso para o laço da linha 11. Em resumo, a complexidade final de tempo é $O(n\, log\, n) + O(1) \times O(n) \times O(n\, log\, n) = O(n^2\, log\, n)$.

Em termos de espaço, a heurística exige armazenar todas as envoltórias convexas, portanto a complexidade de espaço no pior caso é $O(n^2)$.

4.3.3 GRASP

GRASP (*Greedy Randomized Adaptive Search Procedure* - Procedimento de Busca Guloso Adaptativo e Aleatorizado) é uma metaheurística multi-partida ou iterativa para problemas de otimização combinatória, na qual cada iteração consiste em duas fases: heurística construtiva e busca local. A heurística construtiva obtém uma solução inicial e, caso esta não seja factível, aplica-se um procedimento reparador para atingir a factibilidade. Obtida uma solução factível, sua vizinhança é explorada até atingir, no caso de maximização, um máximo local na fase de busca local. Por fim, a melhor solução é mantida como resultado [31].

A metaheurística GRASP é assim chamada devido à combinação aplicada das estratégias gulosa (*Greedy*), aleatorizada (*Randomized*) e adaptativa (*Adaptative*) em cada iteração, as quais podem ser descritas da seguinte maneira [17, 30, 31]:

a) *Greedy*: Fase construtiva considera a construção de uma solução factível onde elementos são incrementados à solução em construção de acordo com uma função de avaliação gulosa. A avaliação dos elementos por esta função leva à criação de uma lista de candidatos restrita (*Restricted Candidate List* - RCL) formada pelos melhores elementos, isto é, aqueles cuja incorporação à atual solução parcial resulta nos menores (ou maiores) custos incrementais.

b) *Randomized*: Dentre os melhores candidatos (lista restrita de candidatos), escolhe-se aleatoriamente um deles para incorporar à solução parcial em construção.

c) *Adaptative*: Assim que um elemento é incorporado à solução, atualiza-se a lista de candidatos e recalcula-se os custos incrementais.

O Algoritmo 5 ilustra o pseudocódigo da metaheurística GRASP.

Algoritmo 5: GRASP(MaxIteracoes)

```
1 início
2     para (k ← 1 até MaxIteracoes) faça
3         Solucao ← FaseConstrutiva();
4         se (Solucao não é factível) então
5             Solucao ← Repara(Solucao);
6         Solucao ← BuscaLocal(Solucao);
7         AtualizaSolucao(Solucao, MelhorSolucao);
8     retorna MelhorSolucao;
```

Adaptado de [31].

Na fase construtiva, seja E o conjunto de elementos de entrada do GRASP e CL (do inglês *candidate list*) o conjunto dos elementos candidatos, o algoritmo inicia considerando todos os elementos pertencentes a E como sendo candidatos. A seleção do próximo elemento para compor a solução é determinada por meio da avaliação de todos os elementos da lista de candidatos de acordo com alguma função de avaliação gulosa [17]. Esta função normalmente representa o custo incremental resultante da inclusão do elemento na solução [31]. A avaliação dos candidatos leva a criação da lista restrita de candidatos (RCL), formada pelos melhores avaliados.

Segundo Resende e Ribeiro [31], a RCL é formada por todos os elementos candidatos que podem ser inseridos na solução sem destruir sua factibilidade e cuja qualidade é superior a um valor limiar. Este valor normalmente é calculado da seguinte forma:

$$c_{min} + \alpha \times (c_{max} - c_{min}) \tag{4.23}$$

onde $\alpha \in [0, 1]$ e c_{min} e c_{max} são, respectivamente, o menor e maior custos incrementais. Neste sentido, $\alpha = 0$ corresponde a construir uma solução gulosa, enquanto $\alpha = 1$ equivale a um algoritmo puramente aleatório.

O elemento a ser incluído na solução parcial é selecionado aleatoriamente da RCL. Uma vez feito isso, a lista de candidatos é atualizada e os custos incrementais reavaliados [31]. Este processo é repetido até que não existam mais candidatos a serem incorporados na solução. O pseudocódigo do procedimento construtivo descrito acima é exibido no Algoritmo 6.

Algoritmo 6: FaseConstrutiva()

```
1  início
2      Solucao ← ∅;
3      Inicialize o conjunto dos elementos candidatos;
4      Avalie os custos incrementais de cada candidato;
5      enquanto (existe pelo menos um elemento candidato) faça
6          Construa a lista restrita de candidatos (RCL);
7          Selecione aleatoriamente um elemento s ∈ RCL;
8          Solucao ← Solucao ∪ {s};
9          Atualize o conjunto dos elementos candidatos;
10         Reavalie os custos incrementais;
11     retorna Solucao;
```

Adaptado de [31].

As soluções geradas na fase construtiva não são necessariamente ótimas. A fase de busca local é usada para melhorar a solução construída inicialmente. O algoritmo de busca local funciona de forma iterativa, substituindo-se sucessivamente a solução atual por uma solução melhor na vizinhança, e termina quando encontra uma solução ótima local, isto é, quando nenhuma solução melhor é encontrada na vizinhança [31]. O pseudocódigo da fase de busca local é apresentado no Algoritmo 7.

Algoritmo 7: BuscaLocal(*Solucao*)

```
   // N = subconjunto de soluções vizinhas (vizinhança)
1  início
2      enquanto (Solucao não é localmente ótima) faça
3          Encontre s' ∈ N(Solucao) com f(s') > f(Solucao);
4          Solucao ← s';
5      retorna Solucao;
```

Adaptado de [31].

O GRASP é uma metaheurística de fácil implementação, e tem sido utilizado na resolução de problemas combinatórios das mais diferentes naturezas, como exemplos [31]: problemas de roteamento, problemas de lógica, problemas de cobertura e particionamento, localização, otimização em redes (grafos), tabela de horários, produção, sequenciamento/escalonamento, transporte, sistemas de potência, telecomunicações, projeto de circuitos VLSI, etc. Mais detalhes sobre a metaheurística podem ser encontrados em [17, 30, 31].

Abordagem GRASP para o MAXAP

Esta seção descreve uma nova metodologia de solução para o MAXAP, baseada na metaheurística GRASP descrita acima. Aqui são apresentados os detalhes da fase construtiva e os motivos da não implementação da fase de busca local para a solução do MAXAP.

A presente abordagem da metaheurística GRASP para o MAXAP baseia-se na heurística apresentada na Seção 4.3.1, a qual constrói uma solução a partir da adição de triângulos vazios.

Antes de iniciar o GRASP, assim como na heurística 3 realiza-se um pré-processamento para construir o conjunto de todos os triângulos vazios que compõem o conjunto de pontos. O Algoritmo 8 apresenta o pseudocódigo da abordagem GRASP para o MAXAP.

Algoritmo 8: MAXAP_GRASP(MaxIteracoes, α)

```
1  início
2      T ← Δ(P);
3      para (k ← 1 até MaxIteracoes) faça
4          Solucao ← MAXAP_FaseConstrutiva(T, α);
5          AtualizaSolucao(Solucao, Melhor_Solucao);
6      retorna Melhor_Solucao;
```

A linha 2 refere-se ao pré-processamento que gera o conjunto dos triângulos vazios. O laço da linha 3 se assemelha ao do algoritmo genérico apresentado no início da seção. Uma diferença é que nesta abordagem não utiliza-se um método reparador pois o polígono resultante da fase construtiva já é simples. Outro ponto é a ausência da busca local. Como a fase construtiva utiliza triângulos vazios para construir uma solução inicial, uma opção para a busca local seria selecionar um triângulo da solução inicial e explorar sua vizinhança para tentar encontrar uma solução melhor. O problema desta abordagem é encontrar um

triângulo vizinho que não implique em interseções no polígono, e garantir que isso não ocorra torna a busca local muito pesada. Outra abordagem seria utilizar a heurística *2-opt* para trocar pares de arestas. No entanto, a vizinhança *2-opt* não traria muitos benefícios uma vez que a solução inicial obtida na fase construtiva já corresponde a um polígono simples. Portanto, a implementação do GRASP proposta neste trabalho não utiliza busca local.

Fase construtiva

A fase construtiva baseia-se no Algoritmo 3 e inicia-se com todos os triângulos vazios sendo candidatos. A abordagem proposta seleciona aleatoriamente o primeiro triângulo a ser incluído na solução. Feito isso, a lista de candidatos é atualizada e o procedimento iterativo para adicionar os outros triângulos se inicia. A atualização da lista de candidatos é realizada conforme o Algoritmo 3, ou seja, a cada triângulo adicionado à solução, todos os triângulos candidatos que o interceptam são removidos da lista de candidatos.

A cada iteração, a RCL é composta por todos os triângulos vazios cujo valor de área seja maior ou igual ao valor limiar calculado a partir da equação 4.23, onde c_{min} e c_{max} são o menor e maior valor de área dos triângulos candidatos da solução parcial, respectivamente. Nesse aspecto, o papel do valor de α na equação 4.23 é invertido, ou seja, $\alpha = 0$ constrói uma solução puramente aleatória, enquanto $\alpha = 1$ equivale a um algoritmo guloso.

O valor de α foi definido a partir de testes exaustivos para encontrar o melhor parâmetro. Observou-se que a metaheurística obtinha melhores resultados a medida que o valor de α aumentava (escolhas mais gulosas), constatando-se o valor ideal no intervalo [0,9, 1]. A Seção 5.3 dá maiores detalhes sobre a metodologia de testes utilizada para escolha do α. O Algoritmo 9 apresenta o pseudocódigo da fase construtiva aplicada ao MAXAP.

Algoritmo 9: MAXAP_FaseConstrutiva($\mathcal{T}$, α)

```
1  início
2      P ← ∅;
3      CL ← T;
4      rand_Δ ← random(Δ) | Δ ∈ T;
5      P ← P ∪ {rand_Δ};
6      T ← T \ {Δ_cross(rand_Δ) | Δ_cross ∈ T)};
7      T ← T \ {rand_Δ};
8      tem_candidato ← true;
9      enquanto |P| < n − 2 faça
10         RCL ← MAXAP_ConstroiRCL(CL, P, α);
11         se (RCL ≠ ∅) então
12             rand_Δ ← random(Δ) | Δ ∈ RCL;
13             P ← P ∪ {rand_Δ};
14             CL ← CL \ {Δ_cross(rand_Δ) | Δ_cross ∈ CL};
15             CL ← CL \ {rand_Δ};
16         senão
17             tem_candidato ← false;
18     retorna P;
```

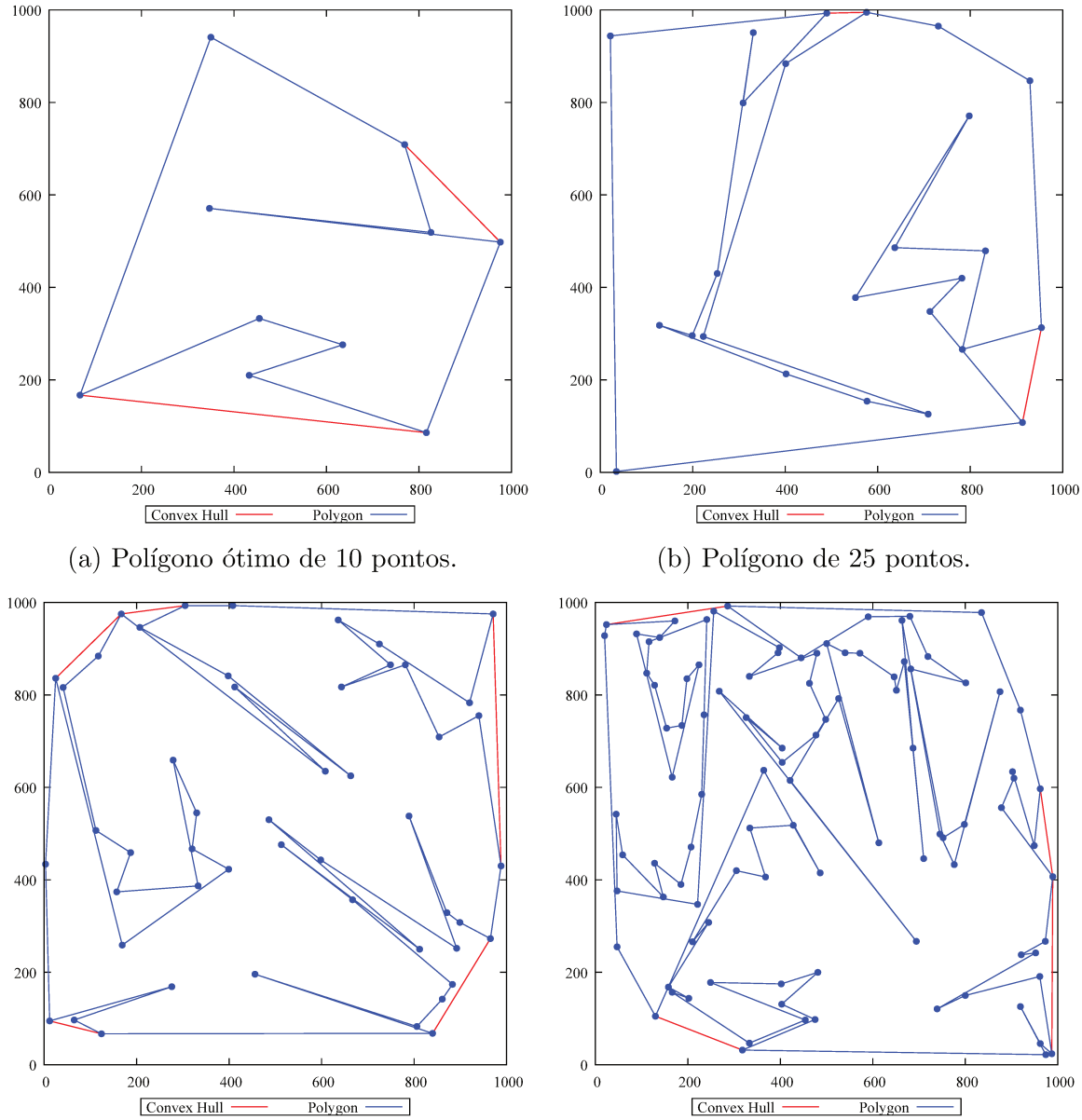
As linhas 2 a 7 inicializam a lista de candidatos (linha 3) e selecionam aleatoriamente um candidato (linha 4) para incorporar o polígono parcial. O laço na linha 9 executa enquanto a lista restrita de candidatos possuir elementos, isto é, enquanto existirem triângulos candidatos para compor a solução. A linha 10 chama o procedimento que constrói a lista restrita de candidatos, e caso esta possua elementos candidatos, seleciona-se aleatoriamente um deles para fazer parte da solução (linhas 11 a 15), do contrário o polígono está completo e o laço termina. A rotina que constrói a lista restrita de candidatos é apresentada no Algoritmo 10.

Algoritmo 10: MAXAP_ConstroiRCL(CL, $\mathcal{P}$, α)

```
1  início
2      c_min ← min(Area Δ) | Δ ∈ CL;
3      c_max ← max(Area Δ) | Δ ∈ CL;
4      threshold ← c_min + α × (c_max − c_min);
5      para (Δ ∈ CL) faça
6          se ((i, j) ∈ P ou (j, k) ∈ P ou (j, k) ∈ P | i, j, k ∈ Δ) então
7              se (Δ não está contido no interior de P) então
8                  se (Area_Δ ≥ threshold) então
9                      RCL ← RCL ∪ {Δ};
10     retorna RCL;
```

As linhas 2 e 3 do algoritmo armazenam os valores de área dos triângulos candidatos de menor e maior área, respectivamente. Na linha 4 calcula-se o valor limiar utilizado na seleção dos melhores candidatos para compor a RCL. O laço na linha 5 percorre todos os triângulos candidatos e inclui na RCL aqueles que são adjacentes ao polígono parcial, não estão contidos no interior do polígono e cujo valor de área seja maior ou igual ao valor limiar calculado.

A Figura 4.14 ilustra alguns exemplos de soluções obtidas pela metaheurística proposta.



(a) Polígono ótimo de 10 pontos.

(b) Polígono de 25 pontos.

(c) Polígono de 50 pontos.

(d) Polígono de 100 pontos.

Figura 4.14: Exemplos de soluções obtidas pelo GRASP.

Análise de complexidade

Uma vez que a metaheurística proposta se baseia na heurística de triangulação, a complexidade de tempo e espaço são as mesmas, $O(n^4)$ e $O(n^3)$, respectivamente. A diferença está no número de iterações que o GRASP faz, que seria um fator constante na complexidade de tempo final, permanecendo assim $O(n^4)$.

4.4 Matheuristic

Recentemente, pesquisadores têm dado atenção à hibridização de heurísticas e metaheurísticas com métodos exatos [5]. Esta técnica é normalmente conhecida na literatura como *Matheuristic*. Nesta seção o termo *heurísticas* remete também às *metaheurísticas*.

De um modo geral, algoritmos híbridos apresentam uma estrutura também chamada de *master-slave* (ou mestre-escravo). Nessa estrutura existem duas possibilidades: (i) a heurística atua em um nível superior e controla as chamadas ao método exato; (ii) o modelo exato atua como mestre e chama e controla a heurística [5].

Caserta e Voß [5] comentam que em abordagens do tipo (i), a heurística **define** a vizinhança, enquanto que o método exato **explora**. Desse ponto de vista, a heurística atua como mestre, definindo o tamanho e limites da vizinhança e controlando as chamadas ao método exato para exploração da mesma.

Por outro lado, em abordagens do tipo (ii), a heurística é embutida no *solver*. Um exemplo são os modernos métodos *branch-and-cut* que exploram os potenciais das heurísticas para obter de forma mais rápida boas soluções, principalmente nos estágios anteriores a árvore de exploração [5].

Essas duas abordagens são também classificadas por Puchinger e Raidl [29] como:

- Combinações colaborativas (*collaborative combinations*): No cenário colaborativo, heurísticas e métodos exatos trocam informações, mas não fazem parte um do outro (abordagem (i)). Podem ser executados sequencialmente, entrelaçados ou em paralelo.

- Combinações integrativas (*integrative combinations*): Neste cenário, uma técnica é um componente subordinado embutido na outra técnica (abordagem (ii)). Existe um algoritmo mestre distinto, que pode ser um método exato ou heurístico, e pelo menos um escravo integrado.

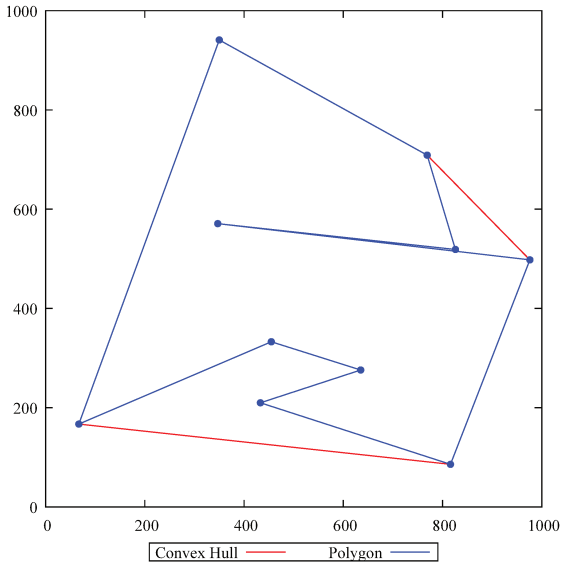
Com base nesses conceitos, o presente trabalho propõe uma *matheuristic* para o MA-XAP, a qual é uma hibridização da heurística baseada em triangulação (Algoritmo 3) com o modelo PLI 4.12.

A *matheuristic* proposta usa a solução obtida pelo Algoritmo 3 como solução inicial para o modelo, limitando o espaço de busca do mesmo, reduzindo o esforço computacional gasto no processo de busca. No entanto, uma vez limitado o espaço de busca, o modelo não garante otimalidade. Nesta abordagem o modelo é relaxado de forma que a solução encontrada deve ter uma quantidade mínima de triângulos da solução obtida pela heurística. Segue abaixo uma descrição formal da *matheuristic*.

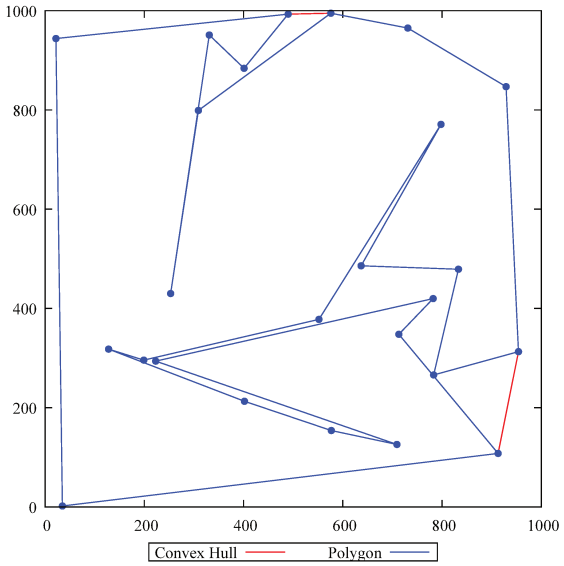
Seja $T(\mathcal{S})$ o conjunto de triângulos vazios de uma solução $\mathcal{S}$ obtida pelo Algoritmo 3. A restrição 4.24 ao ser incluída no conjunto de restrições do modelo PLI (4.12) força que a solução contenha pelo menos $|T(\mathcal{S})| - k$ triângulos vazios da solução $\mathcal{S}$, onde $k \leq |T(\mathcal{S})|$.

$$\sum_{ijk \in T(\mathcal{S})} x_{ijk} \geq |T(\mathcal{S})| - k \qquad T(\mathcal{S}) \subseteq \Delta(P) \tag{4.24}$$

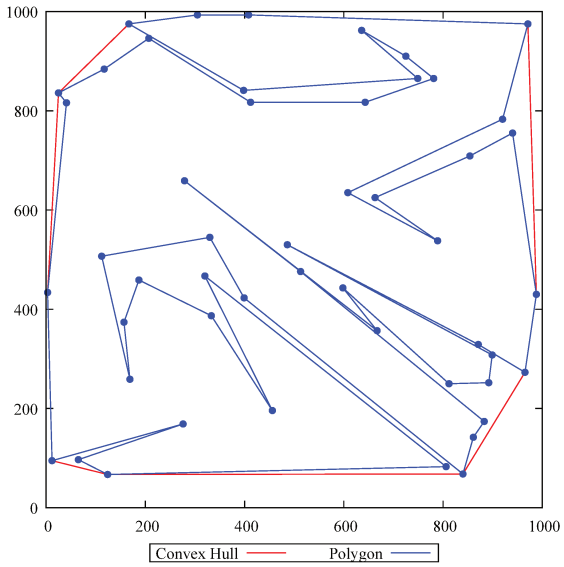
Na Figura 4.15 estão ilustrados alguns exemplos de soluções encontradas pela *matheuristic* descrita acima.



(a) Polígono ótimo de 10 pontos.



(b) Polígono de 25 pontos.



(c) Polígono de 50 pontos.

Figura 4.15: Exemplos de soluções obtidas pela *matheuristic*.

É válido ressaltar que a restrição 4.24 pode ser aplicada nas arestas de forma análoga. Seja $E(\mathcal{S})$ o conjunto de arestas que compõem uma solução heurística $\mathcal{S}$, a restrição 4.25 adicionada ao Modelo 4.12 obriga que a solução otimizada contenha pelo menos $|E(\mathcal{S})|-k$ arestas da solução $\mathcal{S}$.

$$\sum_{ij \in E(\mathcal{S})} y_{ij} \geq |E(\mathcal{S})| - k \qquad\qquad E(\mathcal{S}) \subseteq E \tag{4.25}$$

A Seção 5.4 descreve os testes realizados para análise e como foi realizada a escolha do melhor valor de k para a *matheuristic* proposta.

Capítulo 5

Experimentos computacionais

Este capítulo apresenta os experimentos computacionais realizados no trabalho. A Seção 5.1 descreve as ferramentas utilizadas na implementação dos algoritmos e a metodologia aplicada na experimentação. A Seção 5.2 fornece detalhes da geração do conjunto de instâncias utilizado nos experimentos. Nas Seções 5.3 e 5.4 são apresentados os resultados dos testes empíricos realizados para escolha do melhor valor de α para a metaheurística GRASP e o melhor valor de k para a *matheuristic*, respectivamente. Por fim, a Seção 5.5 apresenta os resultados finais dos experimentos, realizando um comparativo entre as metodologias de solução implementadas.

5.1 Ambiente de desenvolvimento e metodologia de experimentação

Esta seção apresenta as ferramentas computacionais utilizadas para a implementação e execução das metodologias de solução propostas, bem como os métodos usados para a realização dos experimentos computacionais.

Todos os algoritmos foram implementados na linguagem C++. A máquina utilizada nos experimentos computacionais é composta por Processador Intel® Core™ i7-4771 CPU @ 3.5GHz, 8GB de memória RAM e sistema operacional Windows 10 64 bits. A solução dos modelos matemáticos foi realizada pelo *solver* Gurobi[1] versão 7.5.

Os experimentos foram realizados sob o mesmo conjunto de instâncias para todas as metodologias de solução, e todos os valores apresentados foram limitados a três casas decimais. Para os modelos exatos, *matheuristic* e GRASP, estabeleceu-se o tempo de execução limitado a uma hora.

5.2 Geração das instâncias

Poucos estudos na literatura abordam a poligonização de área máxima. Após os trabalhos de Fekete [12, 15, 13], o mais recente foi o publicado por Peethambaran, Parakkat

[1]Gurobi é um *solver* comercial de programação linear inteira. Mais detalhes em www.gurobi.com.

e Muthuganapathy [26]. Diante disso, não encontram-se instâncias elaboradas especificamente para o MAXAP. Fekete em seus trabalhos não realizou experimentos computacionais, e o mais recente utilizou instâncias da TSPLIB e de autoria própria.

O principal problema de utilizar instâncias da TSPLIB para o MAXAP (e problemas geométricos derivados) é que são instâncias peculiares para o TSP do ponto de vista não geométrico, possuindo propriedades que podem atrapalhar o desempenho de algoritmos geométricos. Um exemplo disso é o fato do conjunto de coordenadas (ou pontos) das instâncias da TSPLIB não necessariamente estarem em *posição geral*. Um conjunto de objetos está em **posição geral** se todos os elementos do conjunto satisfazem determinada condição. Quando um ou mais elementos violam a condição, o subconjunto é chamado **degenerado** e, portanto, o conjunto não está em posição geral [32].

Muitos problemas geométricos assumem que a entrada está em posição geral a fim de reduzir o número de casos que devem ser considerados. Para Rosen [32], a hipótese de posição geral depende do problema. Por exemplo, em problemas que envolvem conjuntos de pontos planares, a suposição pode ser que dois pontos não têm as mesmas coordenadas x ou y, não existe colineariedade em subconjuntos de três pontos ou nenhum subconjunto de quatro pontos se encontram no mesmo círculo.

Sendo assim, em meio às características das metodologias de solução propostas e às especificidades do MAXAP, os experimentos computacionais deste trabalho foram realizados para um conjunto de 40 instâncias em posição geral geradas aleatoriamente, divididas em subconjuntos de 10 instâncias com 10, 25, 50 e 100 pontos respectivamente, com coordenadas inteiras no intervalo [0, 1000].

5.3 Parametrização do GRASP

Os principais parâmetros para a metaheurística GRASP são dois: número máximo de iterações e um valor $\alpha \in [0, 1]$ utilizado na função de avaliação dos elementos candidatos. Como supracitado, o valor escolhido para α implica diretamente na aleatoriedade da fase construtiva, isto é, se o algoritmo é mais guloso ou aleatório. Normalmente fazem-se estudos empíricos para determinar o valor de α que resulta nos melhores resultados.

Neste trabalho foram testados diversos valores de α, a partir de 0,005. Os primeiros testes foram com os valores {0,005, 0,01, 0,02, 0,03, 0,04, 0,05, 0,06, 0,07, 0,08, 0,09, 0,1, 0,15, 0,2, 0,25}. Observou-se que a medida que o valor de α aumentava o GRASP obtinha resultados melhores. Realizou-se então uma nova bateria de testes, agora com o valores {0,3, 0,35, 0,4, 0,45, 0,5, 0,55, 0,6, 0,65, 0,7, 0,75, 0,8, 0,85, 0,9, 0,95, 1,0}. Feito isso, constatou-se que para as instâncias maiores o algoritmo obteve melhores resultados usando uma abordagem quase puramente gulosa, com um valor de $\alpha \in [0,9, 1,0]$. Sendo assim, uma última rodada de testes foi realizada, agora com os valores {0,9, 0,91, 0,92, 0,93, 0,94, 0,95, 0,96, 0,97, 0,98, 0,99, 1,0} para as instâncias maiores. Em todos os testes a metaheurística foi executada durante o período de uma hora. A Tabela 5.1 apresenta os resultados do último teste realizado. Os valores em negrito destacam as melhores soluções obtidas para cada instância e para cada valor de α. A última linha da tabela contém a média das áreas encontradas para cada valor de α, destacando em negrito a melhor.

Tabela 5.1: Resultados obtidos pelo GRASP para α variando de 0,9 a 1,0.

Instância	A_{CH}	**0,9**	**0,91**	**0,92**	**0,93**	**0,94**	**0,95**	**0,96**	**0,97**	**0,98**	**0,99**	**1**
input25_1	548433,0	477043,0	477043,0	478109,0	477043,0	478109,0	**479465,0**	**479465,0**	**479465,0**	477043,0	**479465,0**	477043,0
input25_2	821460,5	737786,0	**742243,0**	722879,5	**742243,0**	737786,0	737786,0	**742243,0**	726061,0	726061,0	730518,0	730518,0
input25_3	731502,0	**674916,0**	**674916,0**	**674916,0**	673071,0	**674916,0**	**674916,0**	674598,0	674598,0	674598,0	**674916,0**	669153,5
input25_4	669933,5	**581302,0**	**581302,0**	574709,5	**581302,0**	**581302,0**	581050,5	**581302,0**	**581302,0**	**581302,0**	**581302,0**	**581302,0**
input25_5	739425,5	627351,5	**632963,0**	627035,5	627035,5	627035,5	627035,5	623828,0	627035,5	627035,5	627035,5	623828,0
input25_6	767510,0	**697738,0**	697247,5	697550,0	697247,5	696458,0	696458,0	694526,5	691121,0	697247,5	697247,5	697247,5
input25_7	759767,0	693876,5	693876,5	693680,5	693680,5	693876,5	692828,0	693680,5	**694065,0**	687655,5	687655,5	687655,5
input25_8	668515,0	604060,0	**608143,0**	604060,0	593769,0	604060,0	604060,0	604060,0	604060,0	589686,0	589686,0	585169,0
input25_9	748495,5	**676297,5**	**676297,5**	**676297,5**	**676297,5**	**676297,5**	**676297,5**	**676297,5**	666139,5	**676297,5**	663604,5	663604,5
input25_10	639718,5	**559283,5**	557085,0	**559283,5**	557085,0	**559283,5**	557085,0	557085,0	557085,0	557085,0	557085,0	557085,0
input50_1	842095,0	763055,0	759546,5	764082,5	756544,5	761904,5	**764233,5**	756216,5	755458,0	755458,0	755458,0	758907,0
input50_2	772040,5	657213,0	655284,0	**659000,5**	656923,5	645161,0	**659000,5**	653787,0	**659000,5**	646924,5	**659000,5**	**659000,5**
input50_3	861561,5	750270,5	750567,0	**754275,5**	753911,0	750567,0	752568,0	752568,0	750567,0	752568,0	752568,0	752568,0
input50_4	786575,5	679582,5	681939,0	675733,0	681080,0	677057,5	679812,0	684660,5	691706,5	681285,0	**695103,5**	681903,0
input50_5	671751,5	**588261,0**	584695,5	578819,0	585008,5	583673,5	586317,5	586075,5	580137,0	584663,0	586317,5	581067,5
input50_6	832342,0	752008,0	752354,0	752354,0	749875,0	748266,0	751341,0	**762535,5**	757495,0	752354,0	752354,0	752354,0
input50_7	841551,5	702555,5	**703922,0**	696298,0	696298,0	687616,5	701779,0	696298,0	696298,0	698229,0	696298,0	698229,0
input50_8	804230,5	672065,5	672809,0	666714,5	**677016,5**	672504,5	666979,0	671075,5	672504,5	666906,0	672504,5	666979,0
input50_9	761448,0	648018,0	666170,5	663056,0	**669836,0**	647578,5	655400,5	647555,5	654357,0	648266,0	651102,5	651102,5
input50_10	857377,0	747919,5	740119,5	745559,0	750419,0	744154,0	748094,5	746640,0	**760201,5**	737745,5	746622,5	747516,0
input100_1	912330,0	755297,5	749697,5	754447,0	757536,5	758181,0	753905,5	756710,0	759692,5	748490,5	**764377,0**	750159,0
input100_2	838407,0	**713137,5**	705583,5	694615,5	710484,0	712854,5	696439,0	695254,0	690364,0	697292,5	693601,5	698513,0
input100_3	893201,5	744776,5	736625,0	740064,5	**747854,0**	743314,0	741965,5	733943,5	745621,0	741196,0	727402,0	733259,5
input100_4	853128,5	716693,0	**720044,5**	714623,0	717991,5	714040,5	710054,5	713354,0	718912,0	715564,0	710043,0	719996,5
input100_5	885114,5	763360,0	773779,0	773082,0	765931,0	773367,0	777536,5	772281,0	772876,5	777730,0	773942,5	**778789,5**
input100_6	888872,5	**746225,5**	737534,5	739775,0	736142,0	734542,0	737925,5	738825,0	732705,0	744814,0	740224,0	739326,0
input100_7	899917,0	**772548,0**	756192,0	768521,0	760378,0	771561,0	761267,5	767004,5	759455,0	771731,5	757500,0	754084,5
input100_8	901445,5	740842,0	**767754,5**	757437,0	746500,0	750022,5	750882,0	749929,0	746177,0	743596,0	747940,0	736927,0
input100_9	843688,0	693372,0	693120,0	690361,5	690010,5	**698232,0**	696237,5	688509,0	698323,0	694983,5	697104,0	693910,0
input100_10	928504,5	767910,5	**778196,5**	757861,5	761048,5	752436,0	759665,5	762081,0	751852,0	758000,5	754337,0	758381,0
Média		690158,8	**690901,7**	688506,7	689652,1	688538,6	689279,5	688746,3	688487,8	687060,3	687410,5	686186,0

A_{CH}: área da envoltória convexa

É possível observar na tabela que para os valores de α 0,9 e 0,91 o GRASP obteve melhores resultados, com uma média final levemente superior para $\alpha = 0{,}91$. Nos testes optou-se por não utilizar as instâncias de 10 pontos pelo fato dessas instâncias não discriminarem adequadamente os impactos na variação de α.

Para uma melhor visualização, a Figura 5.1 apresenta um gráfico com as áreas médias obtidas pelo GRASP em cada conjunto de instância. O gráfico mostra que $\alpha = 0{,}91$ mostrou-se o melhor ajuste para as instâncias de 25 e 100 pontos, além de estar bem próximo da melhor média obtida para as instância de 50 pontos. Com base nesses resultados, definiu-se o valor de $\alpha = 0{,}91$ para a realização dos experimentos finais.

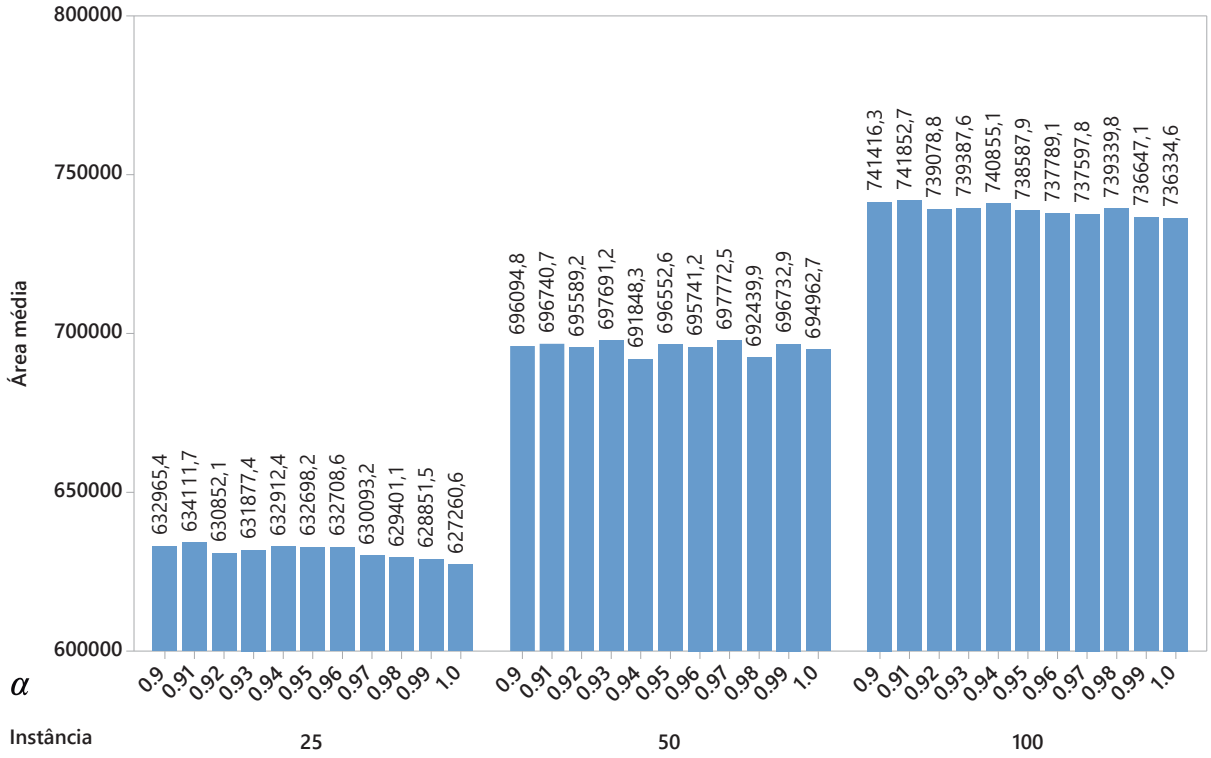


Figura 5.1: Área média obtida pelo GRASP para $\alpha \in [0, 9, 1, 0]$.

5.4 Parametrização da matheuristic

O esforço computacional necessário para a solução do Modelo 4.12 incluindo a restrição 4.24 depende do valor de k, isto é, quanto maior for o valor de k, maior será o espaço de busca e, consequentemente, o esforço computacional. Nesse sentido, foram realizados testes para $k = \{1, 2, 3, 4, 5\}$. Dada a solução inicial obtida pela heurística de triangulação, a *matheuristic* foi executada repetidamente até satisfazer um dos seguintes critérios de parada: atingir o tempo limite de uma hora ou encontrar o ótimo local. De modo formal, encontrada uma solução $\mathcal{S}_i$ pela *matheuristic*, esta é utilizada como a nova solução inicial, e o modelo é executado novamente. Quando $\mathcal{S}_{i+1} = \mathcal{S}_i$ o modelo atingiu o ótimo local. A Tabela 5.2 apresenta os resultados obtidos nos testes. Os valores em negrito destacam as melhores soluções encontradas no intervalo de valores de k testados. No caso das instâncias de 10 pontos o destaque também refere-se às soluções ótimas.

Tabela 5.2: Resultados obtidos pela *matheuristic* para k = {1, 2, 3, 4, 5}.

		Área					CPU (s)				
Instância	A_{CH}	**k = 1**	**k = 2**	**k = 3**	**k = 4**	**k = 5**	**k = 1**	**k = 2**	**k = 3**	**k = 4**	**k = 5**
input10_1	433981,0	386793,5	400430,5	400430,5	400430,5	**414579,5**	0,031	0,046	0,265	0,140	0,406
input10_2	342695,5	280908,0	280908,0	280908,0	280908,0	**288219,5**	0,031	0,046	0,234	0,390	0,515
input10_3	348374,0	333899,5	**341230,5**	**341230,5**	**341230,5**	**341230,5**	0,031	0,125	0,171	0,125	0,140
input10_4	490496,5	457559,0	457559,0	**465488,0**	**465488,0**	**465488,0**	0,031	0,046	0,093	0,078	0,218
input10_5	493022,0	457695,0	**470497,0**	**470497,0**	**470497,0**	**470497,0**	0,031	0,109	0,125	0,140	0,171
input10_6	318446,5	244987,0	261592,0	261592,0	261592,0	**265929,0**	0,031	0,109	0,140	0,312	0,515
input10_7	485913,0	395215,0	395215,0	395215,0	**406472,5**	**406472,5**	0,031	0,062	0,109	0,484	0,703
input10_8	407827,5	**401340,5**	**401340,5**	**401340,5**	**401340,5**	**401340,5**	0,031	0,031	0,046	0,062	0,046
input10_9	445677,0	398352,5	**398426,0**	**398426,0**	**398426,0**	**398426,0**	0,015	0,078	0,171	0,203	0,203
input10_10	246394,5	235043,0	**238795,0**	**238795,0**	**238795,0**	**238795,0**	0,031	0,046	0,093	0,250	0,406
input25_1	548433,0	442092,5	442092,5	489423,0	489423,0	**490489,0**	1,765	2,500	34,594	136,361	540,956
input25_2	821460,5	637588,0	643291,5	671768,0	**750508,0**	741733,5	2,375	4,859	23,937	1657,368	2583,884
input25_3	731502,0	623976,0	644022,5	644180,5	**678337,5**	678337,5	3,427	10,593	15,734	64,094	122,674
input25_4	669933,5	532283,0	542362,0	543402,5	561137,0	**583553,5**	1,765	2,187	8,015	45,453	740,953
input25_5	739425,5	602915,0	631334,5	631334,5	631334,5	**653035,0**	1,734	7,859	43,266	105,470	3596,307
input25_6	767510,0	607896,5	691526,5	695912,0	708681,5	**711222,5**	2,109	12,515	28,469	352,630	582,586
input25_7	759767,0	649768,5	671974,0	674989,5	**689114,0**	**689114,0**	3,953	6,406	9,250	46,828	163,783
input25_8	668515,0	551481,0	589675,0	591069,0	597399,0	**608166,0**	2,234	13,953	17,750	71,547	379,864
input25_9	748495,5	555015,0	623785,5	675769,5	675456,0	**681983,5**	1,828	13,843	73,047	529,929	1101,674
input25_10	639718,5	515885,0	518786,5	533483,5	564764,5	**566291,5**	1,890	5,250	10,218	100,87	420,865
input50_1	842095,0	679415,5	723543,5	730361,5	**737946,5**	692294,0	253,124	627,351	1441,963	3600,000	3600,000
input50_2	772040,5	621827,5	642983,5	**666932,0**	661421,0	642768,5	85,958	351,441	1691,868	3600,000	3600,000
input50_3	861561,5	688219,5	743168,5	**763786,5**	747049,0	751664,5	195,933	719,524	3037,429	3600,000	3600,000
input50_4	786575,5	621844,5	663236,5	684167,5	**685237,0**	649039,5	220,528	428,880	1542,999	3600,000	3600,000
input50_5	671751,5	538947,5	541646,0	588383,0	**593135,5**	581431,5	89,083	271,644	2109,885	3600,000	3600,000
input50_6	832342,0	737127,5	741693,5	**753422,5**	**753422,5**	747335,5	138,180	226,221	842,073	3598,904	3600,000
input50_7	841551,5	600525,5	668709,5	**698223,0**	697672,0	678814,5	83,096	741,087	3016,600	3600,000	3600,000
input50_8	804230,5	614900,5	676940,0	676584,0	**679089,5**	661429,0	154,861	703,243	1159,811	3597,118	3600,000
input50_9	761448,0	610479,0	621985,5	**670134,0**	661585,0	640689,5	86,063	253,721	2162,527	3600,000	3600,000
input50_10	857377,0	631429,0	727174,0	743209,5	**744473,0**	734746,0	128,617	501,850	1126,869	3600,000	3600,000
input100_1	912330,0										
input100_2	838407,0										
input100_3	893201,5										
input100_4	853128,5										
input100_5	885114,5			Out of memory					Out of memory		
input100_6	888872,5										
input100_7	899917,0										
input100_8	901445,5										
input100_9	843688,0										
input100_10	928504,5										

A_{CH}: Área da envoltória convexa

Pela tabela é possível observar que para as instâncias de 10 pontos a *matheuristic* encontra o ótimo na maioria dos casos para $k = 3$ e $k = 4$, e em todos os casos para $k = 5$. Nas instâncias com 25 pontos $k = 5$ também resultou nos melhores resultados para a maior parte das instâncias. Para as instâncias de 50 pontos, os melhores resultados foram obtidos para $k = 3$ e $k = 4$. É possível notar que o aumento no valor de k amplia o espaço de busca porém torna o modelo mais pesado, o que por sua vez resultou em soluções com menor qualidade para as instâncias com 50 pontos.

O gráfico apresentado na Figura 5.2 mostra uma comparação entre as áreas médias para todas instâncias obtidas pela *matheuristic*. Observa-se que para $k = 4$ a média total foi superior.

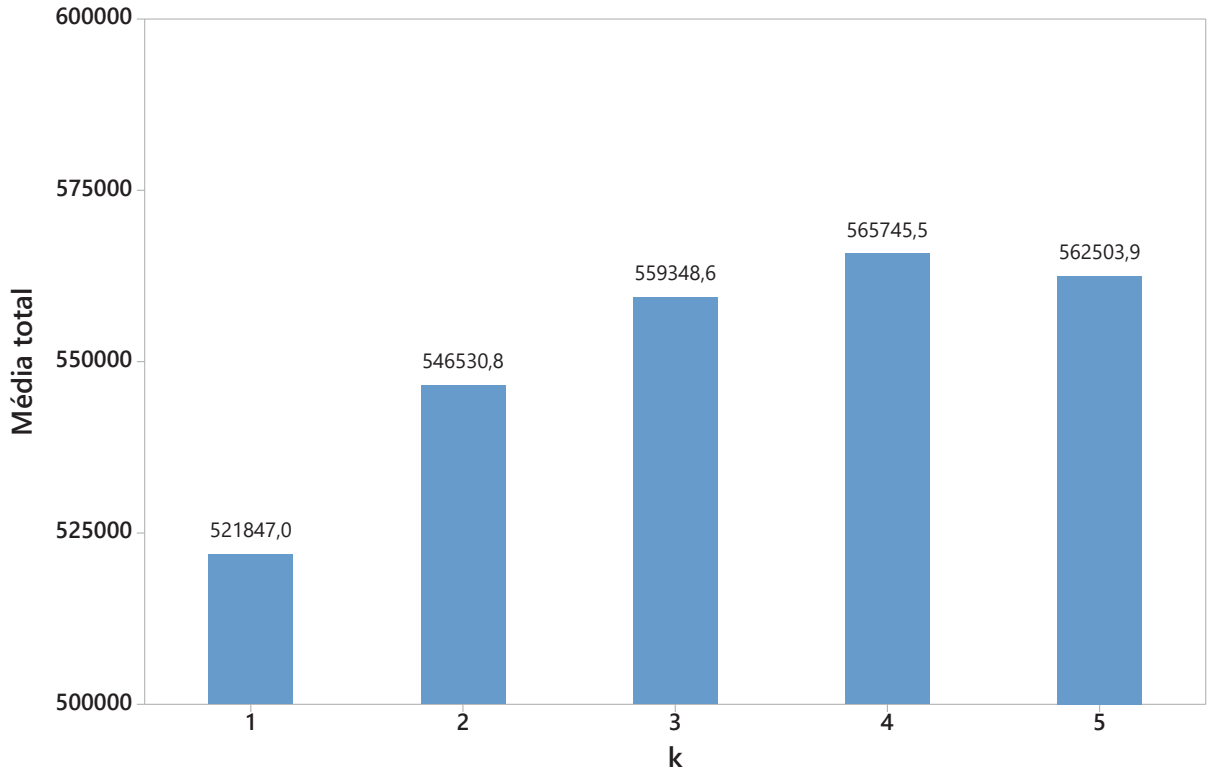


Figura 5.2: Áreas médias para todas instâncias obtidas pela *matheuristic* para $k = \{1, 2, 3, 4, 5\}$.

Testes similares foram realizados considerando a restrição 4.25 e utilizando como soluções iniciais as obtidas pelo algoritmo aproximado (Seção 4.1) e o algoritmo das envoltórias convexas (Seção 4.3.2). Contudo, observou-se que os resultados não foram competitivos aos apresentados nesta seção. Com base nessas informações, serão utilizados para comparação os resultados obtidos pela *matheuristic* aplicando a restrição 4.24 e $k = 4$.

5.5 Análise dos resultados

5.5.1 Modelos exatos

Na Tabela 5.3 estão sumarizados os resultados dos experimentos computacionais com os modelos matemáticos descritos nas seções 4.2.1 e 4.2.2. As colunas 3, 4, 5 e 6 exibem os resultados para o modelo baseado no cálculo de área de um polígono simples (Modelo 4.1), e as colunas 7, 8, 9 e 10 mostram os resultados do modelo baseado em triangulação (Modelo 4.12). Os valores em negrito destacam as soluções ótimas.

Tabela 5.3: Resultados obtidos pelos modelos exatos.

		Modelo 1				Modelo 2			
Instância	A_{CH}	L_P	L_D	**GAP (%)**	**CPU (s)**	L_P	L_D	**GAP (%)**	**CPU (s)**
input10_1	433981,00	**414579,50**	414579,50	0,00	0,115	**414579,50**	414579,50	0,00	0,125
input10_2	342695,50	**288219,50**	288219,50	0,00	0,058	**288219,50**	288219,50	0,00	0,265
input10_3	348374,00	**341230,50**	341230,50	0,00	0,034	**341230,50**	341230,50	0,00	0,046
input10_4	490496,50	**465488,00**	465488,00	0,00	0,016	**465488,00**	465488,00	0,00	0,062
input10_5	493022,00	**470497,00**	470497,00	0,00	0,061	**470497,00**	470497,00	0,00	0,078
input10_6	318446,50	**265929,00**	265929,00	0,00	0,078	**265929,00**	265929,00	0,00	0,156
input10_7	485913,00	**406472,50**	406472,50	0,00	0,062	**406472,50**	406472,50	0,00	0,156
input10_8	407827,50	**401340,50**	401340,50	0,00	0,021	**401340,50**	401340,50	0,00	0,046
input10_9	445677,00	**398426,00**	398426,00	0,00	0,047	**398426,00**	398426,00	0,00	0,093
input10_10	246394,50	**238795,00**	238795,00	0,00	0,078	**238795,00**	238795,00	0,00	0,156
input25_1	548433,00	476788,00	812348,50	70,37	3600,000	480624,50	538177,00	12,00	3600,000
input25_2	821460,50	687706,50	1246424,37	81,24	3600,000	743503,50	811004,10	9,00	3600,000
input25_3	731502,00	651016,00	1248515,25	91,77	3600,000	680270,00	727763,20	7,00	3600,000
input25_4	669933,50	526856,50	1083443,14	105,64	3600,000	590171,50	652797,60	11,00	3600,000
input25_5	739425,50	640466,00	1010784,44	57,82	3600,000	649732,00	716571,30	10,00	3600,000
input25_6	767510,00	679129,50	1103089,41	62,42	3600,000	711222,50	749342,30	5,00	3600,000
input25_7	759767,00	696951,50	1128256,15	61,88	3600,000	701971,00	748215,40	7,00	3600,000
input25_8	668515,00	565966,00	1045346,50	84,70	3600,000	608166,00	650999,70	7,00	3600,000
input25_9	748495,50	643627,50	1388115,80	115,67	3600,000	681174,00	742610,00	9,00	3600,000
input25_10	639718,50	539330,50	982916,65	82,24	3600,000	565254,50	620736,00	10,00	3600,000
input50_1	842095,00	—	—	—	3600,000	—	—	—	3600,000
input50_2	772040,50	—	—	—	3600,000	—	—	—	3600,000
input50_3	861561,50	—	—	—	3600,000	—	—	—	3600,000
input50_4	786575,50	—	—	—	3600,000	—	—	—	3600,000
input50_5	671751,50	—	—	—	3600,000	—	—	—	3600,000
input50_6	832342,00	—	—	—	3600,000	—	—	—	3600,000
input50_7	841551,50	—	—	—	3600,000	—	—	—	3600,000
input50_8	804230,50	—	—	—	3600,000	—	—	—	3600,000
input50_9	761448,00	—	—	—	3600,000	—	—	—	3600,000
input50_10	857377,00	—	—	—	3600,000	—	—	—	3600,000
input100_1	912330,00								
input100_2	838407,00								
input100_3	893201,50								
input100_4	853128,50								
input100_5	885114,50		Out of memory				Out of memory		
input100_6	888872,50								
input100_7	899917,00								
input100_8	901445,50								
input100_9	843688,00								
input100_10	928504,50								

A_{CH}: área da envoltória convexa
L_P: limitante primal obtido
L_D: limitante dual obtido
GAP $= 100 * \frac{(L_D - L_P)}{L_P}$: desvio de otimalidade obtido pelo modelo
CPU: tempo em segundos gasto

Como pode ser observado na tabela, ambos os modelos encontraram o ótimo para as instâncias de 10 pontos rapidamente. No quesito tempo de execução o modelo 1 se mostrou superior em todas as instâncias do conjunto de 10 pontos. Por outro lado, a qualidade dos limitantes obtidos pelo modelo 2 é muito superior ao modelo 1 nas instâncias com 25 pontos. O modelo 2 obteve desvios de otimalidade de no máximo 12%, enquanto que o modelo 1 obteve desvios acima de 50%, ultrapassando 100% em alguns casos. Nenhum dos modelos encontraram soluções factíveis para instâncias com 50 e 100 pontos no tempo limite de uma hora. Inclusive, no caso das instâncias de tamanho 100, o modelo ultrapassou o limite de memória RAM disponível na máquina.

5.5.2 Algoritmo aproximado e Heurísticas construtivas

Esta seção apresenta os resultados dos experimentos realizados com as metodologias de solução aproximada e heurística. A Tabela 5.4 apresenta a área obtida e os tempos de execução dos algoritmos MAXAP_Aproximado, MAXAP_TriangulosVazios e MAXAP_ConvexHulls (Algoritmos 2, 3 e 4, respectivamente). Os valores destacados em negrito simbolizam as melhores soluções obtidas.

Tabela 5.4: Resultados obtidos pelas metodologias aproximada e heurística.

		Área			**CPU (s)**		
Instância	A_{CH}	**Aproximado**	**Convex hulls**	**Triângulos**	**Aproximado**	**Convex hulls**	**Triângulos**
input10_1	433981,0	368327,0	286370,5	**386793,5**	0,000	0,000	0,000
input10_2	342695,5	196557,0	248210,0	**280908,0**	0,000	0,000	0,000
input10_3	348374,0	284373,0	315869,0	**328772,5**	0,000	0,000	0,000
input10_4	490496,5	448410,0	**457927,5**	457559,0	0,000	0,000	0,000
input10_5	493022,0	415376,5	415522,5	**432739,5**	0,000	0,000	0,000
input10_6	318446,5	229753,5	214739,0	**244987,0**	0,000	0,000	0,000
input10_7	485913,0	319993,0	353088,5	**395215,0**	0,000	0,000	0,000
input10_8	407827,5	242632,0	265692,5	**401340,5**	0,000	0,000	0,000
input10_9	445677,0	284348,5	267187,0	**398352,5**	0,000	0,000	0,000
input10_10	246394,5	225369,0	134498,0	**235043,0**	0,000	0,000	0,000
input25_1	548433,0	381291,0	313478,0	**442092,5**	0,000	0,000	0,004
input25_2	821460,5	456812,5	633886,0	**637588,0**	0,000	0,000	0,003
input25_3	731502,0	522522,5	461349,0	**621969,5**	0,000	0,000	0,003
input25_4	669933,5	416112,0	469549,0	**532283,0**	0,000	0,000	0,004
input25_5	739425,5	411559,0	483655,5	**602915,0**	0,000	0,000	0,004
input25_6	767510,0	564014,0	509208,0	**607896,5**	0,000	0,000	0,004
input25_7	759767,0	541152,0	366348,0	**620840,0**	0,000	0,000	0,003
input25_8	668515,0	487048,0	373622,5	**551481,0**	0,000	0,000	0,003
input25_9	748495,5	398391,5	**565556,0**	555015,0	0,000	0,000	0,005
input25_10	639718,5	423645,5	398066,0	**515885,0**	0,000	0,000	0,005
input50_1	842095,0	494204,0	567905,0	**648486,5**	0,000	0,000	0,036
input50_2	772040,5	420149,5	494394,0	**617124,0**	0,000	0,000	0,040
input50_3	861561,5	521540,0	544647,0	**674221,0**	0,000	0,000	0,042
input50_4	786575,5	551868,5	422082,0	**588079,0**	0,000	0,000	0,043
input50_5	671751,5	413532,5	393425,5	**537567,5**	0,000	0,000	0,040
input50_6	832342,0	433834,5	516389,0	**727537,0**	0,000	0,000	0,029
input50_7	841551,5	481641,0	571067,0	**600525,5**	0,000	0,000	0,024
input50_8	804230,5	450032,0	474641,0	**598271,0**	0,000	0,000	0,029
input50_9	761448,0	419313,5	471520,0	**609154,5**	0,000	0,000	0,044
input50_10	857377,0	480593,5	421893,0	**625913,5**	0,000	0,000	0,038
input100_1	912330,0	501442,0	572016,5	**699084,5**	0,000	0,001	0,348
input100_2	838407,0	430103,0	482111,5	**609746,0**	0,000	0,001	0,289
input100_3	893201,5	448903,5	521494,5	**678660,0**	0,000	0,001	0,282
input100_4	853128,5	477897,5	444426,5	**665150,0**	0,000	0,001	0,312
input100_5	885114,5	498244,5	554572,5	**718131,0**	0,000	0,001	0,312
input100_6	888872,5	479601,5	531735,5	**674639,5**	0,000	0,001	0,235
input100_7	899917,0	580697,0	538284,5	**720581,5**	0,000	0,001	0,266
input100_8	901445,5	502227,5	506502,5	**702288,5**	0,000	0,001	0,296
input100_9	843688,0	425951,0	501959,5	**668560,0**	0,000	0,001	0,295
input100_10	928504,5	588595,5	539713,5	**684878,0**	0,000	0,001	0,293

A_{CH}: área da envoltória convexa

É possível observar pela tabela que o MAXAP_TriangulosVazios (coluna 5) obteve as melhores soluções em 38 instâncias do total de 40. Por outro lado, por possuir a maior

complexidade resultou nos maiores tempos de execução. O MAXAP_ConvexHulls (coluna 4) alcançou os melhores resultados em duas instâncias, uma no conjunto de 10 pontos e outra no conjunto de 25 pontos. No entanto, em média, seus resultados são competitivos com os do algoritmo MAXAP_Aproximado (coluna 3). A comparação entre as metodologias pode ser melhor visualizada no gráfico da Figura 5.3, que exibe o percentual médio da área ocupada em relação à área da envoltória convexa para cada conjunto de instâncias.

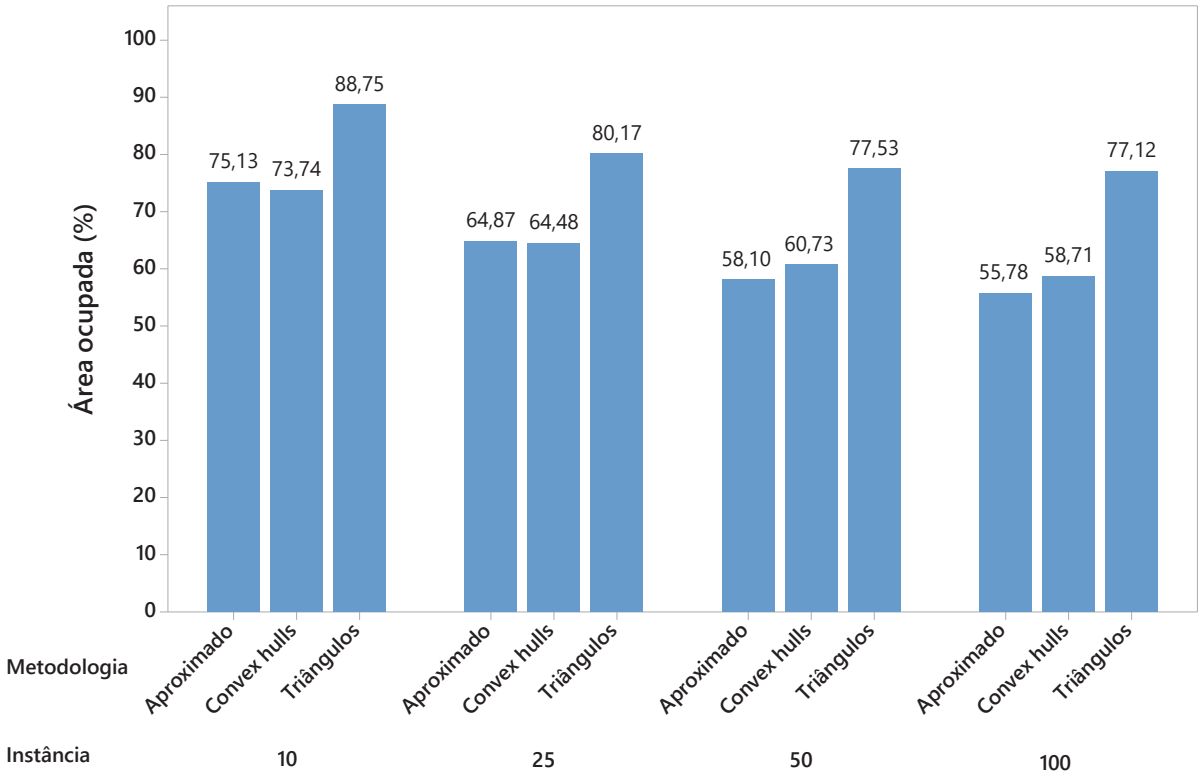


Figura 5.3: Percentual de área ocupada pelos conjuntos de instâncias.

O gráfico mostra a superioridade da heurística de triangulação frente aos algoritmos aproximado e de envoltórias convexas. Nas instâncias de tamanho 10, a abordagem MAXAP_TriangulosVazios obteve áreas que ocupam em média 88,75% da envoltória convexa, caindo pouco mais de 8% no conjunto com 25 pontos. Aumentando o tamanho da instância de 25 para 50 pontos, a mesma abordagem teve um decréscimo de 2,64%, e de apenas 0,41% aumentando para 100 pontos. É possível observar que a medida que o tamanho da instância aumenta, o MAXAP_Aproximado aproxima-se de seu pior desempenho teórico (fator $\frac{1}{2}$). Além disso, é interessante notar que o MAXAP_ConvexHulls também parece se aproximar da metade da área da envoltória convexa para instâncias maiores. Um possível trabalho futuro seria uma análise aprofundada sobre o funcionamento do MAXAP_ConvexHulls e descobrir se o mesmo também é um algoritmo aproximado, e qual seu fator aproximativo.

5.5.3 GRASP

A Tabela 5.5 apresenta os resultados obtidos pela metaheurística GRASP. Uma vez que as soluções ótimas para as instâncias de 10 pontos são conhecidas, para essas entradas

de dados o GRASP executou sobre dois critérios de parada: (i) até atingir a otimalidade; (ii) até atingir o tempo limite de uma hora. Nos demais conjuntos de instâncias o único critério de parada para a metaheurística é o tempo limite de uma hora. Os valores em negrito destacam soluções com garantia de otimalidade encontradas pelo GRASP.

Tabela 5.5: Resultados obtidos pela metaheurística GRASP.

Instância	A_{CH}	A_{GRASP}	**Pré-processamento (s)**	**Fase Construtiva (s)**	**# Iteracoes**	**CPU (s)**
input10_1	433981,0	**414579,5**	0,000	0,000	14	0,002
input10_2	342695,5	**288219,5**	0,000	0,000	21	0,000
input10_3	348374,0	**341230,5**	0,000	0,000	26	0,002
input10_4	490496,5	**465488,0**	0,000	0,000	11	0,003
input10_5	493022,0	**470497,0**	0,000	0,000	11	0,003
input10_6	318446,5	261592,0	0,000	0,000	16735444	3600,000
input10_7	485913,0	**406472,5**	0,000	0,000	24	0,002
input10_8	407827,5	**401340,5**	0,000	0,000	4	0,000
input10_9	445677,0	**398426,0**	0,000	0,000	7	0,000
input10_10	246394,5	**238795,0**	0,000	0,000	8	0,000
input25_1	548433,0	479465,0	0,001	0,004	853177	3600,000
input25_2	821460,5	742243,0	0,001	0,004	877966	3600,000
input25_3	731502,0	674916,0	0,001	0,003	958673	3600,000
input25_4	669933,5	581302,0	0,001	0,004	897548	3600,000
input25_5	739425,5	632963,0	0,001	0,004	797739	3600,000
input25_6	767510,0	697738,0	0,001	0,003	903966	3600,000
input25_7	759767,0	694065,0	0,001	0,003	927063	3600,000
input25_8	668515,0	608166,0	0,001	0,004	897318	3600,000
input25_9	748495,5	681142,5	0,001	0,004	793329	3600,000
input25_10	639718,5	559283,5	0,001	0,004	867789	3600,000
input50_1	842095,0	768719,5	0,014	0,035	102350	3600,000
input50_2	772040,5	671509,5	0,014	0,034	103926	3600,000
input50_3	861561,5	758085,5	0,012	0,035	102849	3600,000
input50_4	786575,5	698730,0	0,012	0,037	95236	3600,000
input50_5	671751,5	590099,0	0,013	0,034	104168	3600,000
input50_6	832342,0	768235,0	0,013	0,034	104980	3600,000
input50_7	841551,5	710442,0	0,011	0,032	109928	3600,000
input50_8	804230,5	676896,5	0,014	0,030	116211	3600,000
input50_9	761448,0	680150,5	0,012	0,033	106670	3600,000
input50_10	857377,0	760201,5	0,013	0,034	102960	3600,000
input100_1	912330,0	776144,5	0,130	0,251	14290	3600,000
input100_2	838407,0	714669,5	0,127	0,250	14362	3600,000
input100_3	893201,5	750353,5	0,120	0,265	13573	3600,000
input100_4	853128,5	733914,0	0,134	0,256	14039	3600,000
input100_5	885114,5	777778,5	0,134	0,260	13844	3600,000
input100_6	888872,5	751872,0	0,129	0,254	14156	3600,000
input100_7	899917,0	781489,0	0,139	0,248	14478	3600,000
input100_8	901445,5	761134,0	0,123	0,257	13957	3600,000
input100_9	843688,0	701015,0	0,147	0,249	14412	3600,000
input100_10	928504,5	773853,0	0,122	0,250	14388	3600,000

A_{CH}: área da envoltória convexa

A_{GRASP}: área obtida pelo GRASP

Pré-processamento (s): tempo médio em segundos gasto pelo pré-processamento (geração dos conjuntos de triângulos vazios e arestas)

Fase construtiva (s): tempo médio gasto em segundos pela fase construtiva do GRASP

 # Iterações: número de iterações rodadas

Pela tabela percebe-se que em 90% das instâncias de 10 pontos o GRASP encontrou a solução ótima em poucas iterações. Somente a instância *input10_6* executou durante o período de uma hora e não encontrou a solução ótima. Em instâncias com poucos pontos, o conjunto de triângulos vazios é pequeno, consequentemente a vizinhança a ser explorada também é pequena. Nestes casos, um alto valor de α pode prevenir uma exploração adequada do espaço de busca, implicando em uma convergência prematura.

A Figura 5.4 apresenta um gráfico de desempenho da metaheurística GRASP quanto à ocupação de área de suas soluções. Como na seção anterior, o gráfico exibe o percentual médio da área ocupada por cada conjunto de instâncias em relação às áreas das envoltórias convexas. Nas instâncias de 10 pontos as soluções obtidas pelo GRASP ocupam em média 91,87% das envoltórias convexas, enquanto que nas instâncias com 100 pontos houve um decréscimo de 6,83%, ocupando 85,04%.

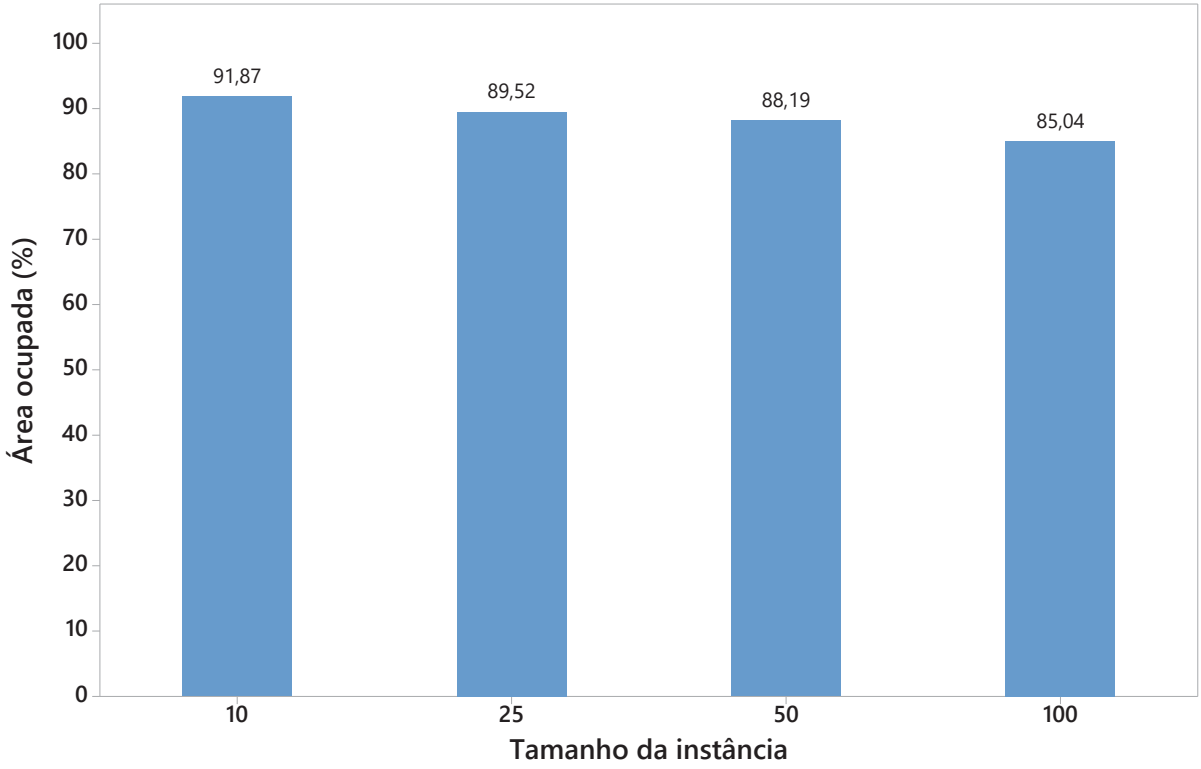


Figura 5.4: Percentual de área ocupada pelos conjuntos de instâncias.

5.5.4 Comparativo das metodologias

Finalmente, esta seção compara por meio de tabelas e gráficos os resultados obtidos por todas as metodologias de solução. As Tabelas 5.6 e 5.7 apresentam o comparativo entre todas a metodologias em termos de área encontrada e tempo de execução, respectivamente. Os valores destacados em negrito indicam soluções ótimas para instâncias de 10 pontos e as melhores soluções encontradas para as demais instâncias.

Na Tabela 5.6 observa-se que, com exceção dos modelos exatos (colunas 3 e 4), o GRASP e a *matheuristic* foram as metodologias que mais obtiveram soluções com garantia de otimalidade nas instâncias de tamanho 10, com 90% das instâncias para o GRASP e 70% para a *matheuristic*. Quanto às abordagens aproximada e heurísticas (colunas 5,

6 e 7), somente o MAXAP_TriangulosVazios (coluna 7) encontrou o ótimo na instância *input10_8*. Para as instâncias de tamanho 25, o modelo baseado em triangulação (coluna 4) se sobressaiu, encontrando as melhores soluções em 80% das instâncias. Para o mesmo conjunto, o GRASP e a *matheuristic* continuaram competitivos. A metaheurística mostra sua superioridade nas instâncias com 50 e 100 pontos. Nas instâncias de tamanho 50, o GRASP obteve as melhores soluções em 90% das instâncias, e a *matheuristic* em apenas 10%. Para o conjunto de instâncias com 100 pontos a *matheuristic* não conseguiu encontrar soluções e, comparado às outras metodologias que obtiveram resultados, o GRASP encontrou as melhores soluções para todas as instâncias do conjunto.

Tabela 5.6: Comparativo final entre as soluções obtidas por cada metodologia.

		Área						
Instância	A_{CH}	**Modelo 1**	**Modelo 2**	**Aproximado**	**Convex hulls**	**Triângulos**	**GRASP ($\alpha = 0,91$)**	**Matheuristic (k = 4)**
input10_1	433981,0	**414579,5**	**414579,5**	368327,0	286370,5	386793,5	**414579,5**	400430,5
input10_2	342695,5	**288219,5**	**288219,5**	196557,0	248210,0	280908,0	**288219,5**	280908,0
input10_3	348374,0	**341230,5**	**341230,5**	284373,0	315869,0	328772,5	**341230,5**	**341230,5**
input10_4	490496,5	**465488,0**	**465488,0**	448410,0	457927,5	457559,0	**465488,0**	**465488,0**
input10_5	493022,0	**470497,0**	**470497,0**	415376,5	415522,5	432739,5	**470497,0**	**470497,0**
input10_6	318446,5	**265929,0**	**265929,0**	229753,5	214739,0	244987,0	261592,0	261592,0
input10_7	485913,0	**406472,5**	**406472,5**	319993,0	353088,5	395215,0	**406472,5**	**406472,5**
input10_8	407827,5	**401340,5**	**401340,5**	242632,0	265692,5	**401340,5**	**401340,5**	**401340,5**
input10_9	445677,0	**398426,0**	**398426,0**	284348,5	267187,0	398352,5	**398426,0**	**398426,0**
input10_10	246394,5	**238795,0**	**238795,0**	225369,0	134498,0	235043,0	**238795,0**	**238795,0**
input25_1	548433,0	476788,0	480624,5	381291,0	313478,0	442092,5	479465,0	**489423,0**
input25_2	821460,5	687706,5	743503,5	456812,5	633886,0	637588,0	742243,0	**750508,0**
input25_3	731502,0	651016,0	**680270,0**	522522,5	461349,0	621969,5	674916,0	678337,5
input25_4	669933,5	526856,5	**590171,5**	416112,0	469549,0	532283,0	581302,0	561137,0
input25_5	739425,5	640466,0	**649732,0**	411559,0	483655,5	602915,0	632963,0	631334,5
input25_6	767510,0	679129,5	**711222,5**	564014,0	509208,0	607896,5	697738,0	708681,5
input25_7	759767,0	696951,5	**701971,0**	541152,0	366348,0	620840,0	694065,0	689114,0
input25_8	668515,0	565966,0	**608166,0**	487048,0	373622,5	551481,0	**608166,0**	597399,0
input25_9	748495,5	643627,5	**681174,0**	398391,5	565556,0	555015,0	681142,5	675456,0
input25_10	639718,5	539330,5	**565254,5**	423645,5	398066,0	515885,0	559283,5	564764,5
input50_1	842095,0	—	—	494204,0	567905,0	648486,5	**768719,5**	737946,5
input50_2	772040,5	—	—	420149,5	494394,0	617124,0	**671509,5**	661421,0
input50_3	861561,5	—	—	521540,0	544647,0	674221,0	**758085,5**	747049,0
input50_4	786575,5	—	—	551868,5	422082,0	588079,0	**698730,0**	685237,0
input50_5	671751,5	—	—	413532,5	393425,5	537567,5	590099,0	**593135,5**
input50_6	832342,0	—	—	433834,5	516389,0	727537,0	**768235,0**	753422,5
input50_7	841551,5	—	—	481641,0	571067,0	600525,5	**710442,0**	697672,0
input50_8	804230,5	—	—	450032,0	474641,0	598271,0	676896,5	**679089,5**
input50_9	761448,0	—	—	419313,5	471520,0	609154,5	**680150,5**	661585,0
input50_10	857377,0	—	—	480593,5	421893,0	625913,5	**760201,5**	744473,0
input100_1	912330,0			501442,0	572016,5	699084,5	**776144,5**	
input100_2	838407,0			430103,0	482111,5	609746,0	**714669,5**	
input100_3	893201,5			448903,5	521494,5	678660,0	**750353,5**	
input100_4	853128,5			477897,5	444426,5	665150,0	**733914,0**	
input100_5	885114,5	Out of memory		498244,5	554572,5	718131,0	**777778,5**	Out of memory
input100_6	888872,5			479601,5	531735,5	674639,5	**751872,0**	
input100_7	899917,0			580697,0	538284,5	720581,5	**781489,0**	
input100_8	901445,5			502227,5	506502,5	702288,5	**761134,0**	
input100_9	843688,0			425951,0	501959,5	668560,0	**701015,0**	
input100_10	928504,5			588595,5	539713,5	684878,0	**773853,0**	

A_{CH}: Área da envoltória convexa

Na Tabela 5.7 por outro lado, percebe-se o *tradeoff* entre qualidade e eficiência das

metodologias. Os modelos exatos executaram durante uma hora nos conjuntos de instâncias com 25 e 50 pontos, e ocasionaram em estouro de memória nas instâncias de tamanho 100. A *matheuristic* teve o mesmo comportamento, porém conseguiu encontrar o ótimo local nas instâncias com 25 pontos e em duas instâncias de 50 pontos antes de atingir o tempo limite de 1 hora. O GRASP também rodou por uma hora em uma instância de tamanho 10 e em todas as demais com 25, 50 e 100 pontos.

Tabela 5.7: Comparativo final entre os tempos de execução de cada metodologia.

	CPU (s)						
Instância	**Modelo 1**	**Modelo 2**	**Aproximado**	**Convex hulls**	**Triângulos**	**GRASP (α = 0,91)**	**Matheuristic (k = 4)**
input10_1	0,115	0,125	0,000	0,000	0,000	0,002	0,140
input10_2	0,058	0,265	0,000	0,000	0,000	0,000	0,390
input10_3	0,034	0,046	0,000	0,000	0,000	0,002	0,125
input10_4	0,016	0,062	0,000	0,000	0,000	0,003	0,078
input10_5	0,061	0,078	0,000	0,000	0,000	0,003	0,140
input10_6	0,078	0,156	0,000	0,000	0,000	3600,000	0,312
input10_7	0,062	0,156	0,000	0,000	0,000	0,002	0,484
input10_8	0,021	0,046	0,000	0,000	0,000	0,000	0,062
input10_9	0,047	0,093	0,000	0,000	0,000	0,000	0,203
input10_10	0,078	0,156	0,000	0,000	0,000	0,000	0,250
input25_1	3600,000	3600,000	0,000	0,000	0,004	3600,000	136,361
input25_2	3600,000	3600,000	0,000	0,000	0,003	3600,000	1657,368
input25_3	3600,000	3600,000	0,000	0,000	0,003	3600,000	64,094
input25_4	3600,000	3600,000	0,000	0,000	0,004	3600,000	45,453
input25_5	3600,000	3600,000	0,000	0,000	0,004	3600,000	105,470
input25_6	3600,000	3600,000	0,000	0,000	0,004	3600,000	352,630
input25_7	3600,000	3600,000	0,000	0,000	0,003	3600,000	46,828
input25_8	3600,000	3600,000	0,000	0,000	0,003	3600,000	71,547
input25_9	3600,000	3600,000	0,000	0,000	0,005	3600,000	529,929
input25_10	3600,000	3600,000	0,000	0,000	0,005	3600,000	100,870
input50_1	3600,000	3600,000	0,000	0,000	0,036	3600,000	3600,000
input50_2	3600,000	3600,000	0,000	0,000	0,040	3600,000	3600,000
input50_3	3600,000	3600,000	0,000	0,000	0,042	3600,000	3600,000
input50_4	3600,000	3600,000	0,000	0,000	0,043	3600,000	3600,000
input50_5	3600,000	3600,000	0,000	0,000	0,040	3600,000	3600,000
input50_6	3600,000	3600,000	0,000	0,000	0,029	3600,000	3598,904
input50_7	3600,000	3600,000	0,000	0,000	0,024	3600,000	3600,000
input50_8	3600,000	3600,000	0,000	0,000	0,029	3600,000	3597,118
input50_9	3600,000	3600,000	0,000	0,000	0,044	3600,000	3600,000
input50_10	3600,000	3600,000	0,000	0,000	0,038	3600,000	3600,000
input100_1			0,000	0,001	0,348	3600,000	
input100_2			0,000	0,001	0,289	3600,000	
input100_3			0,000	0,001	0,282	3600,000	
input100_4			0,000	0,001	0,312	3600,000	
input100_5	Out of memory		0,000	0,001	0,312	3600,000	Out of memory
input100_6			0,000	0,001	0,235	3600,000	
input100_7			0,000	0,001	0,266	3600,000	
input100_8			0,000	0,001	0,296	3600,000	
input100_9			0,000	0,001	0,295	3600,000	
input100_10			0,000	0,001	0,293	3600,000	

O que pode-se concluir analisando os dados da tabela acima com os da Tabela 5.6

é: (i) caso não exista um fator mínimo de qualidade mas deseja-se obter uma solução rapidamente, as melhores opções são os algoritmos MAXAP_Aproximado e MAXAP_ConvexHulls; (ii) para instâncias não muito grandes (até 100 pontos, por exemplo), diante da necessidade de encontrar uma solução boa com restrições apertadas de tempo, a melhor opção pode ser a heurística MAXAP_TriangulosVazios; (iii) quando espera-se obter uma solução boa e o fator tempo não é crítico, a metaheurística GRASP é a melhor opção.

A Figura 5.5 apresenta outro comparativo entre as metodologias, mostrando um gráfico com o percentual médio da área ocupada por cada conjunto de instâncias em relação às áreas das envoltórias convexas. O gráfico une os resultados exibidos nas Figuras 5.3 e 5.4 com os obtidos pelos modelos exatos e pela *matheuristic*.

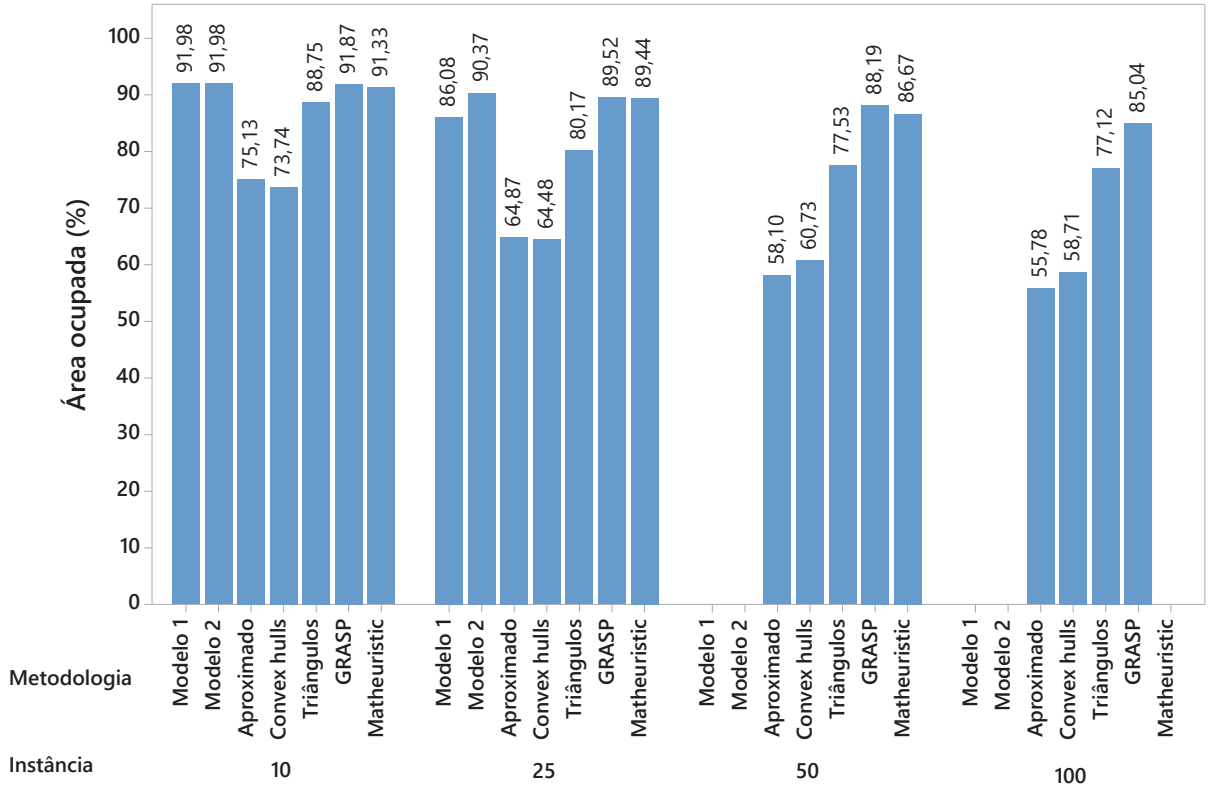


Figura 5.5: Percentual de área ocupada pelas metodologias.

Pela figura é possível ver que no conjunto de instâncias com 10 pontos, mesmo os modelos tendo encontrado soluções ótimas, as metodologias de triangulação, GRASP e *matheuristic* obtiveram resultados muito próximos. Já nas instâncias de tamanho 25, entre os modelos exatos, o modelo baseado em triangulação (Modelo 2) obteve o melhor resultado em média, com o GRASP e a *matheuristic* continuando próximos. Nas instâncias de 50 pontos, os modelos exatos não conseguiram encontrar uma solução no tempo estipulado, mas pode-se observar novamente a superioridade das metodologias GRASP e *matheuristic*, com uma leve vantagem para a metaheurística GRASP. Finalmente, no conjunto de instâncias de tamanho 100, tanto os modelos quanto a *matheuristic* não obtiveram resultados, com o GRASP mais uma vez superando as outras metodologias.

Por fim, uma outra perspectiva de comparação é ilustrada nas Figuras 5.6 e 5.7. Ambas

apresentam um gráfico que mostra o percentual de instâncias para as quais as metodologias obtiveram soluções cujo valor de área possui um desvio x em relação a melhor solução obtida, onde $x \in [0, 100\%]$. O cálculo do desvio $\mathcal{D}_i^m$ obtido pela metodologia m para a instância i pode ser realizado da seguinte forma:

$$\mathcal{D}_i^m = \frac{S_i^* - S_i^m}{S_i^*} \tag{5.1}$$

onde S_i^* é a melhor solução obtida para a instância i e S_i^m é a solução obtida pela metodologia m para a instância i.

A Figura 5.6 apresenta este gráfico comparando as metodologias que conseguiram encontrar soluções para todas instâncias de tamanho 10, 25 e 50.

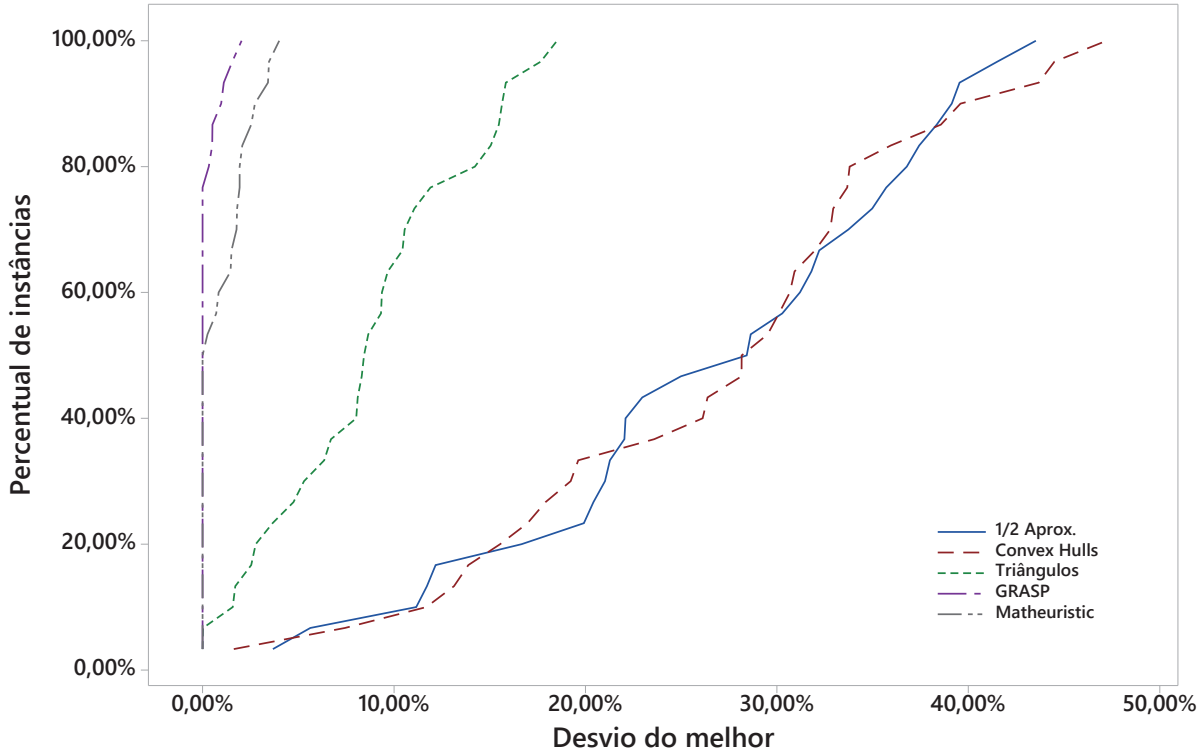


Figura 5.6: Desempenho comparativo entre as metodologias para instâncias de 10, 25 e 50 pontos.

Como é possível perceber no gráfico, o GRASP obteve o maior número de melhores soluções (desvio 0%), seguido pelas metodologias *matheuristic* e a heurística MAXAP_TriangulosVazios. Os algoritmos MAXAP_Aproximado e MAXAP_ConvexHulls obtiveram os maiores desvios, porém foram bastante competitivos entre si.

O último gráfico, ilustrado na Figura 5.7, mostra a mesma comparação, porém restrito às metodologias que encontraram soluções para todas as instâncias testadas. O gráfico deixa claro a hegemonia da metaheurística GRASP, que obteve as melhores soluções em todas as instâncias, seguido pelo MAXAP_TriangulosVazios e por último os competitivos algoritmos MAXAP_Aproximado e MAXAP_ConvexHulls.

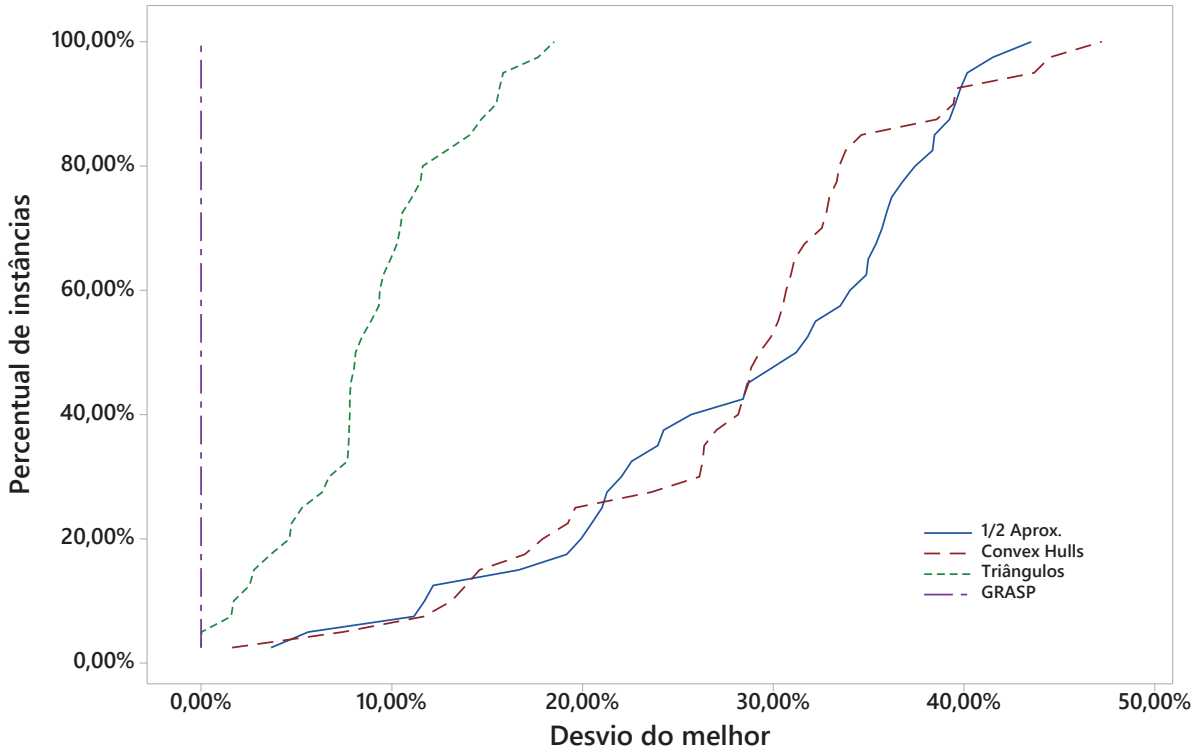


Figura 5.7: Desempenho comparativo entre as metodologias para instâncias de 10, 25, 50 e 100 pontos.

Capítulo 6

Conclusões e trabalhos futuros

O MAXAP é um problema de otimização combinatória NP-difícil que foi proposto por Fekete [12] em 1992. O problema consiste em encontrar um polígono de área máxima gerado por um conjunto de pontos no plano, possuindo aplicações no contexto de reconhecimento de padrões, reconstrução de imagens, clusterização e robótica.

Este trabalho propõe formulações matemáticas e metodologias de solução heurística e exata para o MAXAP. As instâncias utilizadas na avaliação experimental foram geradas com coordenadas aleatórias em posição geral, no intervalo [0, 1000]. Foram propostos dois modelos matemáticos de programação linear inteira, duas heurísticas construtivas, uma metaheurística GRASP e uma *matheuristic*. Experimentos computacionais compararam as metodologias propostas entre si e com o algoritmo $\frac{1}{2}$-aproximado proposto por Fekete [12].

O primeiro modelo PLI proposto (4.1) baseia-se na fórmula para cálculo de área de um polígono simples, e o segundo modelo proposto (4.12) fundamenta-se em triangulações de polígonos. Os resultados obtidos pela solução dos modelos mostram a dificuldade em resolver de maneira exata o MAXAP, encontrando soluções ótimas no período determinado de uma hora somente para instâncias de 10 pontos, em um conjunto de instâncias de até 100 pontos. Para as instâncias com 25 pontos, a solução do segundo modelo (4.12) obteve melhores resultados, encontrando soluções com desvios de otimalidade de 8,7% em média, valor muito abaixo dos 81,37% obtidos pelo primeiro modelo (4.1). Nas instâncias com 50 e 100 pontos os modelos tiveram desempenhos similares, não obtendo nenhum limitante primal ou dual; inclusive, para as instâncias com 100 pontos, a execução dos dois modelos resultou no esgotamento da memória principal do computador onde os experimentos foram conduzidos.

Este trabalho propõe duas heurísticas construtivas gulosas: a primeira fundamentada em triangulação de polígonos (Algoritmo 3); a segunda baseia-se na concatenação de envoltórias convexas (Algoritmo 4). O estudo computacional mostrou que a heurística de triangulação obteve um desempenho superior à heurística de concatenação de envoltórias convexas. Por outro lado, a heurística de envoltórias convexas obteve um desempenho competitivo com um algoritmo $\frac{1}{2}$-aproximado da literatura (Algoritmo 2), com uma leve vantagem para a heurística de envoltórias convexas para as maiores instâncias.

A metaheurística GRASP proposta para o MAXAP contém uma fase construtiva baseada na heurística de triângulação de polígonos. Testes empíricos foram realizados

para o ajuste do parâmetro α da metaheurística. A característica aleatorizada do GRASP permitiu diversificar a exploração do espaço de soluções, obtendo os melhores resultados quando comparados aos das heurísticas construtivas e do algoritmo aproximado. Nas instâncias de 10 pontos obteve soluções ótimas para 90% das instâncias; no conjunto de instâncias de 25 pontos obteve soluções com desvios pequenos aos obtidos pelo modelo PLI baseado em triangulação de polígonos 4.12.

Também é proposto neste trabalho uma *matheuristic* (Seção 4.4) que consiste na hibridização do modelo PLI 4.12 e da heurística construtiva (Algoritmo 3), ambos baseados em triangulação de polígonos. Nesta abordagem o modelo foi adaptado de modo que sua solução seja diferente da solução obtida pela heurística construtiva em no máximo k triângulos. Os experimentos computacionais mostraram que para $k = 4$ a *matheuristic* obteve em média resultados melhores (Seção 5.4). Apesar de não ter conseguido encontrar soluções para as instâncias de 100 pontos, a análise comparativa mostrou que a *matheuristic* obteve resultados competitivos aos obtidos pelo GRASP para instâncias de até 50 pontos.

A contribuição principal deste trabalho consiste no avanço do estado-da-arte no que se refere às metodologias de solução, heurísticas e exatas, para o MAXAP. As metodologias de solução propostas neste trabalho têm o potencial de serem aplicadas no aperfeiçoamento de soluções envolvendo reconhecimento de padrões, reconstrução de imagens, clusterização e robótica. Além disso, o conjunto de instâncias gerado neste trabalho[1] torna-se um *benchmark* para avaliar e comparar outras metodologias para o MAXAP.

6.1 Trabalhos futuros

A partir das formulações PLI apresentadas neste trabalho, as primeiras propostas para o MAXAP, foi possível obter soluções ótimas somente para o conjunto de instâncias com 10 pontos. Sugere-se como trabalho futuro realizar um estudo poliédrico dos dois modelos propostos, propondo desigualdades válidas fortes, eventualmente desigualdades definidoras de facetas. Esse estudo tem potencial de refinar os modelos propostos, ou inclusive fazer surgir novos modelos, reduzindo os desvios de otimalidade e aumentando o tamanho das instâncias para as quais podem ser obtidas soluções ótimas.

Outro possível tema de pesquisa está em aperfeiçoamentos das metodologias heurísticas. Por exemplo, o estudo de um método de busca local que possa ser adicionado à metaheurística GRASP sem impactar seu desempenho computacional a ponto de comprometer uma exploração adequada do espaço de busca.

Após a implementação das metodologias e a execução dos experimentos, foi realizada uma atualização da revisão bibliográfica, onde encontrou-se o trabalho mais recente sobre poligonização de área ótima, de Peethambaran, Parakkat e Muthuganapathy [26]. Esse trabalho apresenta uma heurística aleatorizada proposta para o MINAP ou MAXAP (Algoritmo 1). Visando a comparação com esse novo trabalho, procurou-se executar o GRASP adotando o mesmo *dataset* que os autores utilizaram para o MAXAP. No

[1]Disponível em: `http://www.ic.unicamp.br/~fusberti/problems/maxarea/`. Acesso em: 19 mar. 2018.

entanto, os resultados obtidos pelos autores não estavam coerentes, uma vez que apresentaram áreas maiores que a área da envoltória convexa (um limitante superior trivial). Além disso, as instâncias do *dataset*[2] não estão em posição geral, resultando em instabilidades numéricas não previstas no GRASP. Diante disso, decidiu-se não comparar os resultados obtidos pelo GRASP com aqueles apresentados pelos autores, ficando como trabalho futuro a implementação e análise comparativa do Algoritmo 1 com as demais metodologias propostas neste trabalho.

[2]Disponível em `https://people.sc.fsu.edu/~jburkardt/datasets/spaeth/spaeth.html`. Acesso em: 25 nov. 2017.

Referências Bibliográficas

[1] David L Applegate, Robert E Bixby, Vasek Chvátal, and William J Cook. *The Traveling Salesman Problem: A Computational Study (Princeton Series in Applied Mathematics)*. Princeton University Press, 2007.

[2] Thomas Auer and Martin Held. Heuristics for the generation of random polygons. In *Proceedings of the 8th Canadian Conference on Computational Geometry (CCCG'96)*, pages 38–43. CCCG, 1996.

[3] Mokhtar S Bazaraa, John J Jarvis, and Hanif D Sherali. *Linear Programming and Network Flows*. John Wiley & Sons, 4 edition, 2010.

[4] Aaron Becker, Sándor P Fekete, Alexander Kröller, Seoung K Lee, James McLurkin, and Christiane Schmidt. Triangulating unknown environments using robot swarms. In *Proceedings of the twenty-ninth annual symposium on Computational geometry*, pages 345–346. ACM, 2013.

[5] Marco Caserta and Stefan Voß. *Metaheuristics: Intelligent Problem Solving*, pages 1–38. Springer US, Boston, MA, 2009.

[6] Bernard Chazelle. On the convex layers of a planar set. *IEEE Transactions on Information Theory*, 31(4):509–517, 1985.

[7] Bernard Chazelle and David P Dobkin. Intersection of convex objects in two and three dimensions. *Journal of the ACM (JACM)*, 34(1):1–27, 1987.

[8] Lucas P R Corrêa. Teorema da galeria de arte e triangularização de polígonos e pontos no plano, 2008. Monografia, USP - Universidade de São Paulo, São Paulo, Brasil.

[9] George B Dantzig. *Linear Programming and Extensions*. Princeton University Press, 1963.

[10] Linda Deneen and Gary Shute. Polygonizations of point sets in the plane. *Discrete & Computational Geometry*, 3(1):77–87, 1988.

[11] Sándor P Fekete, Stephan Friedrichs, Michael Hemmer, Melanie Papenberg, Arne Schmidt, and Julian Troegel. Area-and boundary-optimal polygonalization of planar point sets. In *31st European Workshop on Computational Geometry*, pages 133–136, 2015.

[12] Sándor P Fekete. *Geometry and Travelling Salesman Problem*. PhD thesis, University of Waterloo, 1992.

[13] Sándor P Fekete. On simple polygonizations with optimal area. *Discrete & Computational Geometry*, 23(1):73–110, 2000.

[14] Sándor P Fekete, Tom Kamphans, Alexander Kröller, Joseph S B Mitchell, and Christiane Schmidt. Exploring and triangulating a region by a swarm of robots. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 206–217. Springer, 2011.

[15] Sándor P Fekete and William R Pulleyblank. Area optimization of simple polygons. In *Proceedings of the Ninth Annual Symposium on Computational Geometry*, SCG '93, pages 173–182, New York, NY, USA, 1993. ACM.

[16] Sándor P Fekete, Sophia Rex, and Christiane Schmidt. Online exploration and triangulation in orthogonal polygonal regions. In *WALCOM: Algorithms and Computation*, pages 29–40. Springer, 2013.

[17] Thomas A Feo and Mauricio G C Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6(2):109–133, 1995.

[18] A García and Javier Tejel. A lower bound for the number of polygonizations of n points in the plane. *Ars Combinatoria*, 49:3–19, 1998.

[19] Alfredo Garcıa, Marc Noy, and Javier Tejel. Lower bounds on the number of crossing-free subgraphs of kn. *Computational Geometry*, 16(4):211–221, 2000.

[20] Michael R. Garey, David S. Johnson, Franco P. Preparata, and Robert E. Tarjan. Triangulating a simple polygon. *Information Processing Letters*, 7(4):175 – 179, 1978.

[21] Steven Halim, Felix Halim, Steven S Skiena, and Miguel A Revilla. *Competitive Programming 3*. Lulu Independent Publish, 2013.

[22] Maarten Löffler and Wolfgang Mulzer. Unions of onions: Preprocessing imprecise points for fast onion decomposition. *CoRR*, abs/1302.5328, 2013.

[23] Henk Meijer and David Rappaport. Upper and lower bounds for the number of monotone crossing free hamiltonian cycles from a set of points. *Ars Combinatoria*, 30:203–208, 1990.

[24] Flávio K Miyazawa and Cid C De Souza. Introdução a otimização combinatória. In *Jornadas de Atualização em Informática - Congresso da Sociedade Brasileira de Computação - JAI-SBC*. SBC, 2015.

[25] Christos H Papadimitriou and Kenneth Steiglitz. *Combinatorial optimization: algorithms and complexity*. Courier Corporation, 1998.

[26] Jiju Peethambaran, Amal Dev Parakkat, and Ramanathan Muthuganapathy. An empirical study on randomized optimal area polygonization of planar point sets. *Journal of Experimental Algorithmics (JEA)*, 21(1):1–10, 2016.

[27] Breno Piva and Cid C De Souza. The minimum stabbing triangulation problem: Ip models and computational evaluation. In *Second International Symposium Combinatorial Optimization (ISCO 2012)*, volume 7422 of *Lecture Notes in Computer Science*, pages 36–47. Springer, 2012.

[28] Franco P Preparata and Michael I Shamos. *Computational geometry: an introduction*. Springer Science & Business Media, 2012.

[29] Jakob Puchinger and Günther R Raidl. Combining metaheuristics and exact algorithms in combinatorial optimization: A survey and classification. In *International Work-Conference on the Interplay Between Natural and Artificial Computation*, pages 41–53. Springer, 2005.

[30] Mauricio G C Resende. Greedy randomized adaptive search procedures (grasp). *AT&T Labs Research Technical Report*, 98(1):1–11, 1998.

[31] Mauricio G C Resende and Celso C Ribeiro. Greedy randomized adaptive search procedures: Advances, hybridizations, and applications. In *Handbook of metaheuristics*, pages 283–319. Springer, 2010.

[32] Kenneth H Rosen. *Handbook of Discrete and Combinatorial Mathematics*. Discrete Mathematics and Its Applications. Taylor & Francis, 1999.

[33] Micha Sharir, Adam Sheffer, and Emo Welzl. Counting plane graphs: perfect matchings, spanning cycles, and kasteleyn's technique. In *Proceedings of the twenty-eighth annual symposium on Computational geometry*, pages 189–198. ACM, 2012.

[34] Dan Sunday. Area of triangles and polygons. `http://geomalgorithms.com/a01-_area.html`. Acesso em 11 mar. 2018.

[35] Maria Teresa Taranilla, Edilma Olinda Gagliardi, and Gregorio Hernández Peñalver. Approaching minimum area polygonization. 2011.

[36] David P Williamson and David B Shmoys. *The design of approximation algorithms*. Cambridge University Press, 2011.

[37] Laurence A Wolsey. *Integer programming*. Wiley-Interscience, New York, NY, USA, 1998.