

Faculdade de Engenharia da Universidade do Porto



Determinação de Percursos de Corte em Grafos Dinâmicos

Tomé Freitas da Costa

Tese submetida no âmbito do
Mestrado Integrado em Engenharia Electrotécnica e de Computadores
Major de Automação

Orientador: António Miguel da Fonseca Fernandes Gomes (Prof. Dr.)

Junho de 2008

A Dissertação intitulada

“Determinação de percursos de corte em grafos dinâmicos”

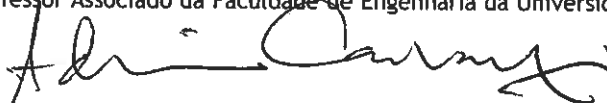
foi aprovada em provas realizadas em 24/Julho/2008

o júri

Presidente

Professor Doutor Adriano da Silva Carvalho

Professor Associado da Faculdade de Engenharia da Universidade do Porto



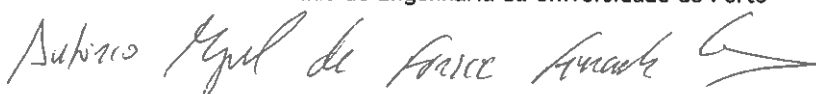
Professora Doutora Maria Teresa do Valle Moura da Costa

Equiparada a Professora Adjunta do Instituto Superior de Engenharia do Porto



Professor Doutor António Miguel da Fonseca Fernandes Gomes

Professor Auxiliar da Faculdade de Engenharia da Universidade do Porto



O autor declara que a presente dissertação (ou relatório de projecto) é da sua exclusiva autoria e foi escrita sem qualquer apoio externo não explicitamente autorizado. Os resultados, ideias, parágrafos, ou outros extractos tomados de ou inspirados em trabalhos de outros autores, e demais referências bibliográficas usadas, são correctamente citados.

Autor - Tomé Freitas da Costa



Faculdade de Engenharia da Universidade do Porto

Resumo

Dado o crescente interesse industrial, particularmente das indústrias têxtil, de calçado ou de metalomecânica, em otimizar os seus sistemas de produção, surge a necessidade de se adaptarem modelos matemáticos conhecidos aos métodos de resolução dos seus problemas. Mais especificamente, fala-se de Problemas de Determinação de Percursos de Corte (PDPCs), onde é necessário efectuar um conjunto de movimentos com uma ferramenta de corte sobre uma placa de material *super-duro*, de modo a obter-se um conjunto predeterminado de peças. Esta dissertação trata do estudo dos referidos modelos matemáticos e a sua adaptação aos PDPCs, de modo a poder desenvolver-se uma aplicação informática que encontre uma solução *boa* para o problema.

Do ponto de vista teórico, o PDPC é um Problema de Rotas em Arco (PRA), que se enquadra no mundo da Optimização Combinatória, onde é necessário planear o percurso de uma ferramenta de corte de forma a cortar um conjunto de peças minimizando a distância do percurso, reduzindo consequentemente o tempo de corte. Do ponto de vista prático, é um problema que normalmente se encontra associado a um conjunto específico de restrições industriais que lhe conferem variados graus de complexidade e originalidade, pelo que o desenvolvimento deste trabalho foi orientado sobre um caso já conhecido.

O trabalho aqui desenvolvido assenta na recodificação de uma aplicação existente baseada num caso real. Procura-se uma implementação directa com a respectiva correcção de alguns detalhes de mau funcionamento existentes na aplicação original e sugere-se um novo critério de decisão para o desenvolvimento de uma heurística alternativa àquela que foi trabalhada. Destaca-se a análise e descoberta de dois erros do algoritmo original assim como a correcção de um deles como o principal resultado obtido desta dissertação.

No fim efectuaram-se diversas experiências computacionais, com dois conjuntos bem distintos de instâncias, com o principal objectivo de validar o bom funcionamento da aplicação desenvolvida.

Abstract

Given the growing industrial interest, particularly textile, shoe or metallurgical industries, in optimise their own production systems, comes the need to adapt known mathematical models to the resolution methods of the problems they have. Specifically, one refers to Cutting Path Determination Problems (CPDPs), where there is the need to execute a set of movements with a cutting tool over a *super-hard* plate, in order to get a predetermined set of pieces. This dissertation is about the study of those mathematical models and their adaptation to the CPDPs so that would be possible to develop a piece of software that could find a *good* solution for the problem.

From a theoretical point of view, the CPDP is an Arc Routing Problem (ARP), which belongs to the Combinatorial Optimization world and where one needs to plan the path of a cutting tool in order to cut a set of pieces, minimizing the path length and therefore reducing the cutting time. From a practical point of view, it is a problem that usually has a very specific set of industrial constraints, which allow different degrees of complexity and originality. All the orientation of this work followed an already known case.

The work of this dissertation is based on the recoding of a piece of software used in a real-life situation. One seeks to develop a direct implementation along with correcting some malfunctioning details of the original software. Also, one new decision criterion is suggested for the development of an alternative heuristic. It should be noticed that the discovering of two mistakes within the original algorithm and the fixing of one of them was the the main achievement of this dissertation.

In the end, computational experiments were made, with two different sets of instances, with the main goal of validating the functioning of the software one developed.

Agradecimentos

Gostaria de manifestar publicamente a minha imensa gratidão pela ajuda e o apoio, de modo directo ou indirecto, que foi fundamental para a execução deste trabalho.

Em primeiro lugar ao Prof. Miguel Gomes. Além de toda a responsável orientação ao longo do semestre, o meu obrigado pela incrível disponibilidade e as inúmeras *horas extra* que fez.

À minha família, agradeço a incansável paciência nos meus dias de mau-humor e nos outros todos em que não existi. Aos pais e à avó, obrigado pelo apoio, pelo carinho e pela compreensão que depositaram em mim. Foi preponderante para o sucesso deste trabalho. À minha irmã, porque foi preciso descarregar em alguém, agradeço o sorriso e a admiração.

Ao Zé, porque além de um enorme amigo foi como um irmão. Porque fizemos juntos todo este caminho e porque mesmo no fim ele estava lá à minha espera, pronto para me ajudar.

A todos os amigos, por não só acreditarem em mim como também por fazerem questão de o dizerem constantemente. Foram uns anos inesquecíveis e todos estiveram presentes. À Daniela, ao Judeu e à Li, uma palavra especial pela força dos últimos dias. Uns fisicamente, outros não, agradeço a companhia e as palavras amigas na difícil recta final. Obrigado.

Tomé

Para o Gô

Conteúdo

1	Introdução	1
1.1	Âmbito da Dissertação	1
1.2	Descrição e Enquadramento do Problema	2
1.3	Trabalho Realizado	3
1.4	Organização da Dissertação	4
2	O Problema do Carteiro Rural	7
2.1	Problemas de Rotas em Arcos	8
2.1.1	O Problema do Carteiro Chinês	10
2.1.2	O Problema do Caixeiro Viajante	11
2.2	O Problema do Carteiro Rural	12
2.2.1	Aplicações	12
2.2.2	Variantes	13
2.2.2.1	O Problema do Carteiro Rural Não Dirigido	13
2.2.2.2	O Problema do Carteiro Rural Dirigido	14
2.2.2.3	O Problema do Carteiro Rural Misto	14
2.2.2.4	O Problema do Carteiro Rural Ventoso e outros PCRs	15
2.2.3	O Problema do Carteiro Rural Dinâmico	15
2.3	Determinação de Percursos de Corte	16
2.3.1	Abordagem “Dinâmica”	17
2.3.1.1	Critério-VMA	24
2.3.2	Abordagem “Force-Support Tree”	27
2.4	Comentários Finais	35
3	Ferramentas Auxiliares	37
3.1	Introdução	37
3.2	Linguagens de Marcação	37
3.2.1	<i>HyperText Markup Language</i> (HTML)	38
3.2.2	<i>eXtensible Markup Language</i> (XML)	38
3.2.2.1	<i>Scalable Vectorial Graphics</i> (SVG)	41
3.3	Linguagens de Programação	41
3.3.1	C++ e a <i>Standard Template Library</i>	42
3.3.2	PHP	44
3.4	Comentários Finais	44
4	Algoritmo de Determinação de Percursos de Corte	45
4.1	Introdução	45
4.2	O ficheiro de entrada	46

4.3	A Heurística VMA	47
4.3.1	Revisões de Implementação	50
4.3.2	Questões de Eficiência	56
4.4	Problemas encontrados	57
4.5	Heurística Construtiva “Peça Mais Próxima”	64
4.6	Comentários Finais	66
5	Testes Computacionais	69
5.1	Instâncias Utilizadas	69
5.1.1	Instâncias “CCORTE”	70
5.1.2	Instâncias “ESICUP”	70
5.2	Experiências Computacionais	71
6	Conclusões	75
A	Percursos de corte obtidos nas experiências computacionais	77
	Referências	100

Lista de Figuras

2.1	Contextualização dos conjuntos de problemas abordados.	8
2.2	As sete pontes de Königsberg.	9
2.3	Grafo do problema das sete pontes de Königsberg.	10
2.4	Transformação do grafo para o PCR Não Dirigido (adaptado de [9]).	14
2.5	Enquadramento industrial de um PDPC (adaptada de [22]).	17
2.6	Envolvente rectangular de um polígono.	20
2.7	Diferentes tipos de visibilidade entre dois vértices v_i e v_j	21
2.8	Fluxograma do algoritmo de “backtracking” por pesquisa em árvore (adaptado de [22]).	23
2.9	Representação em pseudocódigo de alto nível da heurística construtiva VMA.	28
2.10	Projecção criada pelo fio de cobre electrificado (adaptada de [17]).	30
2.11	Ângulo aberto a representar a porção de placa, em torno da peça, que se encontra <i>vazia</i> (adaptada de [17]).	31
2.12	Diâmetros mínimos locais (adaptada de [17]).	32
2.13	Divisão da área a sombreado, pelo vértice v , em três partes diferentes (adaptada de [17]).	33
3.1	Estrutura de um ficheiro HTML.	39
3.2	Exemplo de parte de um ficheiro XML usado no desenvolvimento deste trabalho (adaptada de [21]).	40
3.3	Representação e estrutura de um ficheiro em formato <i>.svg</i>	42
4.1	Estrutura em árvore do ficheiro XML usado como input do PDPC.	46
4.2	Fluxograma da heurística construtiva VMA (adaptado de [22]).	49
4.3	Diagrama de herança das classes definidas.	52
4.4	Representação de alto nível da matriz criada pela interacção das duas classes.	54
4.5	Arquitectura da <i>matriz das visibilidades</i> (adaptado de [22]).	55
4.6	Arquitectura da estrutura de memória (adaptado de [22]).	56
4.7	Problema de visibilidade causado pela posicionamento específico de P_1 e P_2	57
4.8	Representação geométrica das duas peças em análise.	58
4.9	Representação geométrica da situação que provoca o erro em análise, por corte da aresta $(v_{conflito}^1, v_{conflito}^2)$	61
A.1	Instância <i>p01</i>	78
A.2	Instância <i>p02</i>	79
A.3	Instância <i>p03</i>	80
A.4	Instância <i>p04</i>	81
A.5	Instância <i>p05</i>	82
A.6	Instância <i>p06</i>	83

A.7	Instância <i>p07</i> .	84
A.8	Instância <i>p08</i> .	85
A.9	Instância <i>p09</i> .	86
A.10	Instância <i>p10</i> .	87
A.11	Instância <i>ALBANO</i> .	88
A.12	Instância <i>BLAZ</i> .	89
A.13	Instância <i>DAGLI</i> .	90
A.14	Instância <i>DIGHE1</i> .	91
A.15	Instância <i>DIGHE2</i> .	92
A.16	Instância <i>FU</i> .	93
A.17	Instância <i>JAKOBS1</i> .	94
A.18	Instância <i>JAKOBS2</i> .	95
A.19	Instância <i>MAO</i> .	96
A.20	Instância <i>MARQUES</i> .	97

Lista de Tabelas

5.1	Dados relativos às instâncias “CCORTE” utilizadas nas experiências computacionais.	70
5.2	Dados relativos às instâncias “ESICUP” utilizadas nas experiências computacionais.	71
5.3	Resultados obtidos para as instâncias “CCORTE”.	72
5.4	Resultados obtidos para as instâncias “ESICUP”.	73

Lista de Abreviaturas

AD	Algoritmo de Dijkstra
ESICUP	<i>EURO Special Interest Group on Cutting and Packing</i>
HTML	<i>HyperText Markup Language</i>
IO	Investigação Operacional
MIEEC	Mestrado Integrado em Engenharia Electrotécnica e de Computadores
OO	Orientado a Objectos
PCC	Problema do Carteiro Chinês
PCR	Problema do Carteiro Rural
PCRD	Problema do Carteiro Rural Dinâmico
PDPC	Problema de Determinação de Percursos de Corte
PHP	<i>PHP: Hypertext Processor</i>
PMP	Peça Mais Próxima
PRA	Problema de Rotas em Arcos
PRG	Problema de Rotas Genérico
STL	<i>Standard Template Library</i>
SVG	<i>Scalable Vectorial Graphics</i>
TSP	<i>Traveling Salesman Problem</i>
VMA	Vértice Mais Acima
XHTML	<i>eXtensible HyperText Markup Language</i>
XML	<i>eXtensible Markup Language</i>

Capítulo 1

Introdução

1.1 Âmbito da Dissertação

No mundo da matemática e das suas aplicações computacionais, o ramo que se ocupa do estudo modelizado de estruturas representadas por vértices (ou nós) e arestas (ou arcos), é formalmente conhecido por Teoria dos Grafos ([12] e [4]).

É um ramo que engloba uma enorme diversidade de problemas e um sem-fim de aplicações práticas, pelo que estabelece um forte relacionamento com o mundo da Investigação Operacional (IO). Este é um mundo que reflecte as suas inúmeras influências nos problemas do quotidiano comum, de tal modo que não fica indiferente a ninguém. São problemas da mais variada natureza e relacionam-se, por exemplo, com a necessidade de percorrer estradas, ruas ou até efectuar a sua devida manutenção de forma *inteligente*. Ainda, os mais variados serviços, como por exemplo a distribuição do correio ou jornais, a recolha de lixo, a leitura de contadores de água e luz, a supervisão de parquímetros ou a inspecção de cablagens e condutas de gás, entre *muitos* outros, pertencem a esses mesmos problemas que apresentam vantagens se resolvidos de forma *inteligente*. Fala-se de elevados custos anuais associados à maior parte destes serviços, fala-se também de elevados tempos de execução pelo que, qualquer pequena melhoria da solução desses problemas significa uma considerável poupança de recursos a todos os níveis.

Considerando um contexto industrial de enorme abrangência e complexos sistemas de produção que incluem, por exemplo, a indústria têxtil, a indústria de calçado e a indústria metalomecânica, entende-se que diariamente surjam inúmeros problemas onde as referidas pequenas melhorias nas suas respectivas soluções se reflectiriam em enormes benefícios práticos. Deste modo, e considerando o mesmo âmbito industrial, sabe-se que um dos problemas que surge com frequência, ainda que naturalmente adaptado às restrições de cada contexto específico, é o problema da determinação de um caminho (ou de um percurso) que uma qualquer ferramenta de corte irá percorrer de modo a que, no menor intervalo de tempo possível, seja capaz de cortar um conjunto de peças ou figuras de uma determinada superfície.

Ainda que tratando-se de um problema *industrialmente* comum, que visto de uma perspectiva de mais alto nível deixa transparecer relativa simplicidade, a complexidade das situações em que este surge vai necessariamente influenciar o conjunto de restrições que lhe vão ser impostas e consequentemente o seu grau de complexidade. O processo tecnológico de corte, onde se inclui a ferramenta utilizada para o corte e ainda determinadas características sobre a placa a cortar (como o seu material ou o seu posicionamento), é, deste modo, o factor mais determinante na elaboração de um determinado problema de determinação de um percurso de corte.

O desenvolvimento de um determinado algoritmo de resolução para este tipo de problemas assume naturalmente um compromisso entre a qualidade da solução e o tempo que esta demorou a ser obtida. Ou seja, ainda que seja possível o desenvolvimento de algoritmos de resolução exacta capazes de obter uma solução óptima, há obviamente o interesse em obter soluções apenas *boas* mas em tempo útil. Surgem assim os métodos heurísticos para a resolução do problema que, não garantindo a resolução do problema até à optimalidade, são capazes de encontrar, eficientemente, uma solução satisfatória.

1.2 Descrição e Enquadramento do Problema

O objectivo desta dissertação é o estudo e a implementação de algoritmos capazes de determinar percursos de corte em grafos dinâmicos. O problema que está na base desta proposta de dissertação surge em indústrias que necessitam de recorrer a processos tecnológicos de corte onde o percurso de corte tem de ser contínuo, isto é, após ter-se iniciado o processo de corte, não é possível interrompê-lo e reiniciá-lo num outro ponto. Um exemplo de um processo de corte com estas características é o que se baseia em electroerosão por fio, onde o material a cortar está suspenso na máquina de corte e um fio de cobre atravessado por uma corrente eléctrica vai cortando as peças, que caem à medida que vão sendo completamente cortadas. Este processo de corte é habitualmente utilizado em situações onde o material a cortar é muito resistente e é, obviamente, um processo muito demorado.

O Problema da Determinação de Percursos de Corte (PDPC) pode ser entendido como a segunda de duas partes sequenciais de um problema maior, relacionado com o corte de peças de uma placa. Este problema “maior” integra, então, dois problemas distintos mas complementares e com o mesmo objectivo final de minimização de custos: posicionamento de um conjunto predefinido de peças de formas irregulares numa dada placa (problema que passa pela minimização do desperdício entre as peças ou pela maximização do valor das peças colocadas) e determinação do percurso de corte das mesmas (problema que passa pela minimização do tempo e/ou custo do corte a efectuar).

O PDPC consiste não só em determinar um percurso propriamente dito, mas também em garantir que se cortem efectivamente todas as figuras (regulares ou irregulares) de uma superfície de corte. Num contexto industrial, os PDPCs surgem intimamente ligados

à produção de sistemas de ferramentas de corte de alta precisão para trabalhar madeira e seus derivados, cortiça, plásticos, alumínio e outros metais. Da produção de algumas dessas ferramentas, faz parte integrante o corte das cabeças de corte, a partir de uma patela (ou placa) de diamante. O corte das referidas figuras de uma superfície pode ser executado manual ou automaticamente, existindo um conjunto vasto de ferramentas de corte, entre as quais se destacam a faca mecânica, o feixe de raios laser, o jacto de água a alta pressão e o fio de cobre, que irão naturalmente influenciar o posicionamento da placa

Trata-se de um problema de *Optimização Combinatória* onde é necessário planejar o percurso de uma máquina de corte de forma a cortar um conjunto de peças com formas irregulares minimizando o custo do corte, pretendendo-se minimizar a distância do percurso, de forma a reduzir o tempo de corte. Este problema pode ser modelizado como uma variante dinâmica do Problema do Carteiro Rural na medida em que se entende o posicionamento das peças na placa como um conjunto de vértices e arestas representado por um grafo que sofre alterações sempre que uma peça cai, e onde é necessário impôr restrições adicionais ao percurso de corte, nomeadamente que se cortem completamente todas as peças antes de caírem e, dada a forma como o material está suspenso na máquina de corte, não ocorram seccionamentos deste a meio do processo de corte. Assim, à medida que cada peça cai, quando é totalmente cortada, o grafo sobre o qual se realiza a pesquisa do percurso de corte é dinâmico, alterando-se durante o próprio processo de corte e dependendo da ordem pela qual os cortes são processados.

1.3 Trabalho Realizado

O trabalho apresentado nesta dissertação é consequência de uma investigação na área dos Problemas de Rotas em Arcos (PRAs), de modo a que se pudesse obter enquadramento necessário à resolução de um problema industrial de optimização combinatória, nomeadamente a determinação de percursos de corte em grafos dinâmicos.

Numa fase inicial foi efectuado um estudo sobre as principais referências bibliográficas existentes. Primeiro, uma pesquisa de mais alto nível sobre os Problemas de Rotas em Arcos (PRAs) em geral, com especial atenção para o Problema do Carteiro Chinês (PCC) e para o Problema do Carteiro Rural (PCR) e seus principais conceitos e variantes. Com base neste último, compreenderam-se as modelizações efectuadas do mesmo para contextos de problemas específicos. Assim, entendeu-se o Problema da Determinação de Percursos de Corte como um caso particular de uma variante dinâmica do PCR.

Relativamente ao problema específico em mãos, foram estudados e analisados mais detalhadamente os dois únicos casos conhecidos ([23] e [17]) que, apesar de tratarem do mesmo problema, são distintos nas suas estratégias de resolução. Analisaram-se os detalhes que as divergências de restrições impunham e compararam-se métodos de resolução quando as restrições eram comuns.

Em paralelo foi desenvolvida uma pesquisa na área das ferramentas que viriam a ser usadas na resolução do problema e implementação da solução, que passava por compreender dois campos complementares: ao nível de desenvolvimento da aplicação final foram aprofundados conhecimentos relativos à linguagem de programação C++, tirando partido das potencialidades da *Standard Template Library*; ao nível da apresentação gráfica e respectivo interface para com um utilizador foram estudadas linguagens de programação *web* como o HTML e o PHP. Ainda a este nível, estudaram-se os formatos gráficos que melhor pudessem servir todos estes propósitos e, dada a forte componente geométrica do problema desta dissertação, optou-se pelo *Scalable Vector Graphics* (SVG), como ferramenta poderosa na área do desenho vectorial.

A fase seguinte do trabalho compreendeu dois processos temporalmente distintos. Primeiro, análise exaustiva das estruturas de dados e da lógica da aplicação em Java, que iria servir de base ao trabalho a desenvolver. Refira-se que esta é a aplicação que esteve na origem do trabalho descrito em [22, 23]. Sem se ter efectuado uma análise das linguagens de programação que seriam mais adequadas para o trabalho, considerou-se, em jeito de imposição, que o algoritmo recodificado teria de ser apresentado em C++. Assim, efectuou-se a conversão de todo o código para a sintaxe e lógica do C++ e em seguida, o exaustivo *debug* desta nova versão desenvolvida. Em ambos os casos, o processo de compreensão de funcionamento lógico e da implementação originalmente usada foi demorado e lento devido não só à enorme complexidade do algoritmo como também à forma, por vezes excessiva, como toda a informação contida nas estruturas de dados evolui. Deste modo, resultou um algoritmo capaz de executar as mesmas funcionalidades do original, numa linguagem de programação diferente e com correcções de duas origens diferentes: as correcções associadas ao modo de funcionamento básico e intrínseco da linguagem usada e as correcções derivadas da descoberta de pequenos *bugs* de código e *bugs* associados a situações que não tinham sido anteriormente previstas e que originavam erros de funcionamento.

Por fim, desenvolveu-se uma nova abordagem heurística para o problema, baseada num novo critério de decisão *guloso*, tendo em vista o melhoramento do funcionamento da aplicação para algumas situações específicas. Em paralelo foram efectuadas algumas melhorias ao nível da implementação, às quais se seguiram diversos testes de desempenho na versão C++ do algoritmo e alguns testes comparativos entre esta e a aplicação original em Java.

1.4 Organização da Dissertação

Esta dissertação encontra-se organizada em seis capítulos. O presente capítulo constitui uma breve introdução ao tema da dissertação, na qual se apresenta o seu âmbito, enquadramento e uma descrição sucinta do problema sobre o qual esta incide. Ainda, refere-se de modo genérico o trabalho realizado ao longo do desenvolvimento desta dissertação.

O capítulo 2 é dedicado à apresentação do Problema do Carteiro Rural. Começa-se com um enquadramento deste problema no mundo dos Problemas de Rotas em Arcos e apresentam-se as principais variantes do mesmo. Seguindo-se uma lógica de particularidade, introduz-se o Problema do Carteiro Rural Dinâmico como principal referência para o problema tratado nesta dissertação e analisam-se duas situações específicas de Problemas de Determinação de Percursos de Corte, de modo a melhor compreender a dificuldade do problema ([23] e [17]). Estudaram-se detalhadamente esses dois casos e retiveram-se as principais conclusões como base para o trabalho a realizar.

No capítulo 3 descrevem-se conceitos e ferramentas básicas que desempenharam um papel fundamental neste trabalho. O capítulo encontra-se dividido em linguagens de marcação, onde se apresenta o XML como ferramenta de portabilidade poderosa e em linguagens de programação, onde as potencialidades da STL são explicitadas. Referem-se as ferramentas estudadas para o desenvolvimento de uma página *web* que acompanhou o desenvolvimento do trabalho efectuado nesta dissertação.

O capítulo 4 é dedicado à implementação do algoritmo para resolver o Problema de Determinação de Percursos de Corte. É abordada a re-codificação efectuada, tendo em vista revisões ao nível da estrutura lógica de funcionamento da aplicação e questões de eficiência. Destacam-se os principais problemas encontrados neste processo e, nalguns casos, apresentam-se soluções para os resolver. Por último é proposta uma nova heurística construtiva para o PDPC, baseada num novo critério de decisão, influenciado pelas referências estudadas no capítulo 2. Explica-se a lógica por detrás da heurística originalmente implementada e efectua-se uma análise comparativa com a nova heurística proposta.

No capítulo 5 apresentam-se diversos resultados computacionais, efectuados em dois conjuntos distintos de instâncias. Explicam-se as origens dessas instâncias e tecem-se diversas considerações sobre os resultados obtidos.

Por último, no capítulo 6, resumem-se as principais conclusões da presente dissertação, salientando os tópicos mais relevantes a reter. Termina-se com perspectivas de desenvolvimentos futuros na continuidade deste trabalho.

Capítulo 2

O Problema do Carteiro Rural

No mundo da matemática e das suas aplicações computacionais, o ramo que se ocupa do estudo modelizado de estruturas simplesmente representadas por vértices (ou nós) e arestas (ou arcos), é formalmente conhecido por Teoria dos Grafos ([12] e [4]).

Do conjunto de problemas conhecidos que envolvem esta área da matemática, destacam-se os Problemas de Rotas em Arcos (PRAs), que são um caso mais restrito dos Problemas de Rotas Genéricos (PRGs). Assim, seja $G = (V, A)$ um grafo conexo¹ e sem lacetes², $V = \{v_1, \dots, v_n\}$ o conjunto de vértices (ou nós) e $A = \{(v_i, v_j) : v_i, v_j \in V \text{ e } i \neq j\}$ o conjunto de arestas (ou arcos) cada uma associada a um custo, distância ou comprimento não negativo c_{ij} (com $c_{ij} = \infty$ se o arco não se encontra definido). Define-se os PRGs como o tipo de problemas que procura determinar o circuito de custo mínimo que contenha um subconjunto $Q \subseteq V$ de vértices obrigatórios e um subconjunto $R \subseteq A$ de arestas obrigatórias. Em termos de notação utilizada, uma aresta *dirigida* (ou seja, com um custo associado c_{ij} diferente de c_{ji}) é definida como *arco*.

Partindo desta definição genérica que engloba um enorme conjunto de problemas, vão abordar-se neste capítulo contextos cada vez mais restritos até se atingir a especificidade do problema da determinação de percursos de corte, tratado detalhadamente nesta dissertação como uma variante do Problema do Carteiro Rural (PCR) (um dos principais PRAs), seguindo uma lógica de contextualização tal como podemos ver representada na figura 2.1. Na figura é ainda possível observar o Problema do Carteiro Chinês (PCC) como outro dos principais PRAs, que será referido mais à frente.

Na secção 2.1 inicia-se a viagem pelo mundo dos PRAs, com referência ao Problema do Carteiro Chinês (PCC).

Na secção 2.2 deste capítulo detalha-se o Problema do Carteiro Rural (PCR). Inicia-se com uma breve descrição do problema, as suas principais aplicações na vida real e as suas principais variantes. Por último irá dar-se maior atenção ao Problema do Carteiro Rural

¹Existe um caminho entre quaisquer dois vértices do grafo.

²Arcos que partem e chegam ao mesmo vértice.

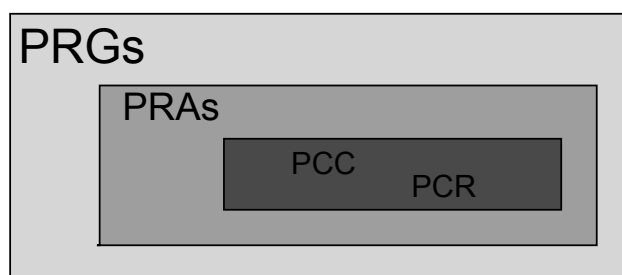


Figura 2.1: Contextualização dos conjuntos de problemas abordados.

Dinâmico (PCRD), por ser a variante do problema que motivou o desenvolvimento desta dissertação.

A secção 2.3 é dedicada à determinação de percursos de corte, uma das conhecidas aplicações industriais dos PCRs. Após breve introdução, refere-se a abordagem dinâmica como principal orientação a seguir nesta dissertação e a abordagem estática do problema como ponto de comparação.

O capítulo termina com alguns comentários finais, onde se salienta a importância das variantes do PCR no desenvolvimento desta dissertação.

2.1 Problemas de Rotas em Arcos

Por definição, dado um subconjunto predefinido de arcos de um grafo, o principal objectivo dos PRAs consiste em determinar um caminho de custo mínimo que atravessasse esse subconjunto, com ou sem restrições adicionais.

São problemas que se encontram nas mais diversas situações do quotidiano e com as quais existe um contacto *camuflado* pela natural diluição das mesmas no dia-a-dia de cada um. Deste modo sabe-se, por exemplo, que cada vez que se pega num carro para efectuar uma deslocação de um sítio para outro, invariavelmente é efectuado um trajecto, cujo comprimento pode ser minimizado e que pode envolver ou não um determinado conjunto de restrições (como parar num posto de abastecimento ou efectuar um desvio deliberado para recolher alguém). Generalizando, os PRAs encontram-se relacionados com a necessidade de percorrer estradas, ruas ou qualquer outro meio que ligue dois locais distintos ou relacionadas com a sua respectiva manutenção periódica. Ainda, os mais variados serviços, como por exemplo a distribuição do correio ou jornais, a recolha de lixo, a leitura de contadores de água e luz, a supervisão de parquímetros ou a inspecção de cablagens e condutas de gás, entre outros, completam o leque de problemas englobados nos PRAs. Dados os elevados custos anuais associados à maior parte destes serviços, qualquer pequena melhoria da solução do problema (que passará sempre por uma redução de algum custo a ele associado) significa uma considerável poupança de recursos económicos e redução de desperdícios por parte das empresas privadas ou dos organismos públicos. São problemas reais que após a transformação em modelos matemáticos adquirem uma dimensão tal cuja

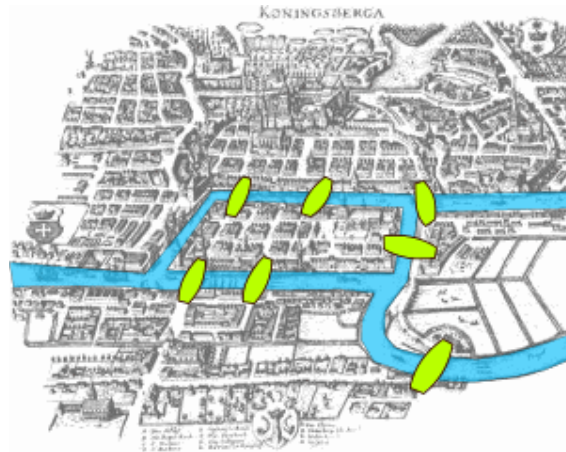


Figura 2.2: As sete pontes de Königsberg.

abrangência irá ser inteligentemente aplicada na resolução de outros problemas da mesma natureza.

A análise dos problemas que vão ser abordados neste capítulo seguirá uma introdução básica: um problema de compreensão empírica que usa habilmente uma ferramenta poderosa para a sua resolução e que no fim apresenta uma solução rapidamente perceptível. Assim, qualquer análise de um problema relacionado com a teoria dos grafos começa obrigatoriamente pela primeira referência escrita sobre o assunto, também conhecida como o problema das sete pontes de Königsberg, cidade da antiga Prússia. A particularidade desta cidade era possuir parte do seu território em duas ilhas no rio Pregel ligadas entre si e a ambas as margens do rio por sete pontes (figura 2.2). Conta a história que os habitantes locais gostavam de passear pela cidade ao Domingo com a intenção de conseguirem efectuar um trajecto de modo a atravessar todas as pontes uma vez apenas. A resposta foi encontrada em 1736 pelo famoso matemático Leonard Euler, que conseguiu provar em [10] quais as condições necessárias e suficientes para a existência de tal trajecto, assim como demonstrou não ser possível efectuá-lo sobre as referidas pontes.

Com base na figura 2.2, desenvolveu-se o grafo que serviu de estudo para Euler, em [10], de modo a entender-se o *grafo*, como uma representação gráfica de extrema importância que se encontra na base de todos os PRAs. Assim, na figura 2.3, temos quatro pontos (vértices ou nós), equivalentes às quatro porções de território que o rio define, e sete ligações entre estes mesmos pontos (arestas ou arcos), equivalentes às sete pontes sobre o rio.

Dois importantes PRAs, resultantes da definição dos PRGs, e que vão ser abordados nas secções seguinte são:

1. O Problema do Carteiro Rural, onde $Q = 0$ e R qualquer;
2. O Problema do Carteiro Chinês, onde $Q = 0$ e $R = A$.

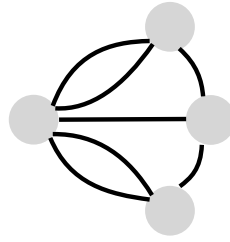


Figura 2.3: Grafo do problema das sete pontes de Königsberg.

os quais foram exaustivamente abordados por Eiselt et al. em [8] e [9]. Deste modo teremos, em ambas as situações, um problema em que é necessário percorrer um conjunto de ligações (arestas), com a diferença a residir no facto de num caso ser obrigatório percorrê-las todas (PCC) e no outro não (PCR).

2.1.1 O Problema do Carteiro Chinês

O Problema do Carteiro Chinês (PCC) é um famoso problema de IO que será aqui abordado numa perspectiva de introdução ao PCR. Entende-se que é um problema intrinsecamente mais fácil de perceber, pela situação básica que representa, pelo que a transição para a compreensão do PCR será naturalmente mais simples uma vez entendido o PCC. A designação para o PCC é atribuída ao matemático chinês Guan, do Shangtun Normal College, que trabalhou algum tempo numa estação de correios durante a revolução cultural chinesa (Korte) e formulou pela primeira vez este problema da seguinte forma ([15]):

“Um carteiro tem de cobrir o sector que lhe está afecto antes de regressar à estação de correios. O problema consiste em determinar o percurso mais curto para o carteiro.”

Define-se um grafo conexo como *Euleriano* se existir um caminho fechado em G que contenha exactamente uma vez cada um dos arcos e pelo menos uma vez cada um dos vértices. Esta propriedade revela-se de particular importância [12] pois as condições necessárias e suficientes para que um grafo conexo, G , seja Euleriano constituem um resultado fundamental:

1. Se G é *Não Dirigido*, todos os vértices devem ter um número par de arestas incidentes, ou seja, grau³ par. Entende-se por *Não Dirigido* um grafo cujos arcos são *arestas*.⁴ Em termos práticos, o custo c_{ij} associado à aresta (i, j) é igual ao custo c_{ji} , associado à aresta (j, i) .

³O conceito de grau de um vértice encontra-se associado ao número de arestas (ou arcos) incidentes nesse mesmo vértice.

⁴A diferença de nomenclatura entre *arcos* e *arestas* não será sempre especificada (por ser indistinta na maior parte da bibliografia existente), excepto nos casos em que tal se torna realmente necessário.

2. Se G é *Dirigido*, o número de arcos que entra num vértice tem de ser igual ao número de vértices que sai. Entende-se por *Dirigido* um grafo cujo o custo c_{ij} é diferente do custo c_{ji} .
3. Se G é *Misto*, todos os vértices devem ter um número par de arcos incidentes dirigidos e não dirigidos; além disso, para todo o $S \subseteq V$, a diferença entre o número de arcos dirigidos de S para $V \setminus S$ e o número de arcos dirigidos de $V \setminus S$ para S deve ser menor ou igual ao número de arcos não dirigidos que ligam S a $V \setminus S$, designada por “Condição de Equilíbrio” [12]. Designa-se por grafo *Misto* um grafo onde existem quer arestas (com custos c_{ij} iguais a c_{ji}), quer arcos (com custos c_{ij} diferentes de c_{ji}).

No entanto, facilmente se compreende que o PCC é um problema que foge um pouco aos contextos reais (pela restrição de obrigatoriedade de percorrer *todas* as arestas de um determinado grafo) o que lhe confere reduzidas aplicações práticas. Assim, será introduzido o PCR, enquanto extensão conceptual do PCC, como um problema que se aproxima mais da realidade e, naturalmente, com maior utilidade na abordagem à resolução dos problemas do quotidiano.

2.1.2 O Problema do Caixeiro Viajante

O Problema do Caixeiro Viajante é um problema matemático que tem vindo a ser profundamente estudado desde o início do século XIX e que resumidamente é explicado como:

Dado um número finito de cidades e um custo associado à travessia entre cada uma delas, o objectivo passa por determinar um caminho de custo mínimo que visite cada uma das cidades uma só vez e termine no ponto de partida (caminho *Hamiltoniano*).

Assim, assumindo custos simétricos (o custo da cidade i para a cidade j é o mesmo da cidade j para a cidade i), a determinação de um caminho entende-se como a ordem pela qual as cidades são visitadas pelo caixeiro.

Ainda que não pertencendo directamente ao conjunto dos PRAs, o Problema do Caixeiro Viajante (TSP)⁵ surge intimamente ligado aos PRAs, nomeadamente ao PCR. Tal como abordado em [20], o PCR é *NP-hard*⁶ e pode ser visto como uma generalização do TSP, na medida em que cada instância de um TSP pode ser convertida numa instância do PCR.

Assim, é possível efectuar a transformação de um determinado PRA em TSP. Sem entrar em grande detalhe matemático, a transformação do grafo efectua-se em pequenos

⁵Do inglês Traveling Salesman Problem e que por questões de conveniência será a sigla adoptada para todas as referências ao Problema do Caixeiro Viajante.

⁶Na teoria da complexidade computacional, *NP-hard* representa a classe de problemas de decisão que são intrinsecamente *mais difíceis* de resolver do que aqueles que conseguem ser resolvidos em tempo polinomial.

passos, passando obviamente por transformar o grafo G num grafo dirigido e em seguida converter o problema para um problema equivalente de rotas em vértices, onde a cada vértice se encontra associada apenas uma aresta. A vantagem de tal transformação reside na possibilidade de se aproveitarem os muitos trabalhos publicados e os resultados já conhecidos sobre o TSP, um dos problemas mais estudados na área da Optimização Combinatória.

2.2 O Problema do Carteiro Rural

O facto do PCR poder ser interpretado como um caso mais geral do PCC, pois apenas se exige que um subconjunto de arestas seja obrigatoriamente percorrido, transporta-o para uma dimensão mais aproximada dos problemas do mundo real uma vez que o número de situações práticas em que seja realmente necessário percorrer todas as arestas do grafo é reduzido. Assim, entende-se matematicamente que uma solução do PCC poderá servir como solução para o PCR, e nunca o contrário.

Assim, o PCR é uma generalização do PCC, tal como foi referido em [6], na medida que, dado o conjunto A de arestas existentes no grafo, o subconjunto R de arestas obrigatórias coincide com A .

Tal como o nome indica, este problema foi inicialmente associado ao problema da distribuição de correio em áreas rurais, onde esta normalmente se faz com um só carteiro. Tomando a definição de grafo atrás referida e dado um subconjunto R de ruas a ser *obrigatoriamente* servidas pelo carteiro na sua distribuição, a particularidade deste problema relativamente ao PCC consiste, em termos metafóricos, no facto de existir um *outro* subconjunto R' de ligações entre as povoações que não necessitam dos serviços do carteiro mas podem ser naturalmente utilizadas por ele, pelo que o carteiro *tem* apenas de percorrer as que determinarem o caminho mais curto para fazer as suas entregas. Assim, o conjunto A , de arestas existentes no grafo, consiste na reunião dos subconjuntos R e R' .

Deste modo, torna-se evidente que para $R = A$, o PCR torna-se um simples PCC [6].

2.2.1 Aplicações

As aplicações do PCR são inúmeras, tal como se referiu na introdução deste capítulo. No entanto, grande parte das situações aí referidas não podem ser modeladas como PCRs puros, uma vez que possuem características específicas e distintas entre si. O que acontece nestas situações passa por adaptar, com maior ou menor complexidade, o modelo teórico às restrições impostas.

Nesta subsecção pretende-se apenas enumerar algumas das aplicações mais comuns, associadas a problemas do quotidiano. Dado o contacto mais ou menos directo que a maior parte das pessoas tem (ou teve) com estas situações, a ideia passa por oferecer uma primeira incursão, menos densa, no mundo dos PCRs, tendo em vista o paralelismo destas mesmas situações tão naturalmente conhecidas por todos. Assim:

- Varredores de Ruas;
- Limpa-Neves;
- Recolha de Lixo;
- Distribuição do Correio;
- Transporte em Autocarros Escolares;
- Leitura de Contadores;

são algumas das principais aplicações do PCR, que foram estudadas e devidamente adaptadas e constituem uma importante referência na bibliografia do problema. Ficam aqui referidas apenas a título de contextualização do PCR, assim como um primeiro contacto mais superficial da enorme abrangência do problema. Refere-se [22] para maior profundidade neste assunto, onde poderão ser encontradas referências específicas sobre cada um dos tópicos acima apontados.

2.2.2 Variantes

Como qualquer problema matemático, o PCR pode ser adaptado a diferentes contextos reais, mediante as alterações introduzidas no seu conjunto de restrições. Estas alterações de problemas reais são por vezes tão específicas que o paralelismo metafórico com um *carteiro* perde o sentido de comparação imediato. Assim, interessa reter a base do modelo teórico de resolução do problema, que irá sofrer mais ou menos alterações. Nas subsecções seguintes serão abordadas as variantes *dirigida*, *não dirigida*, *mista* e *ventosa* do PCR, sendo a variante *dinâmica* detalhada numa secção seguinte, pela sua significativa importância neste trabalho de dissertação.

2.2.2.1 O Problema do Carteiro Rural Não Dirigido

Nesta variante do PCR, todos os arcos são não dirigidos, o que equivale a dizer que são *arestas*. Como referido em [8], quando o grafo $\overline{G} = (V, R)$ é conexo, o PCR Não Dirigido pode ser entendido, dada a sua simplificação, como se se tratasse de um PCC Não Dirigido e resolvido através da determinação dos *caminhos mínimos* (em G) entre vértices de grau ímpar.

O PCR é habitualmente resolvido sobre um grafo modificado, $G' = (V', A')$, assim definido ([22]): $V' = \{v_i \in V : (v_i, v_j) \in R \text{ para algum } v_j \in V\}$ e A' é obtido a partir de A da seguinte forma:

1. Acrescenta-se a A uma aresta (v_i, v_j) para cada $v_i, v_j \in V'$, com custo c_{ij} equivalente ao caminho mínimo (em G) entre v_i e v_j (na figura 2.4 é possível observar esta situação para a aresta de custo 7, na parte inferior do grafo);

2. Retiram-se todas as arestas $(i, j) \in A \setminus R$ para cada $c_{ij} = c_{ik} + c_{kj}$, para algum k , bem como uma de duas arestas paralelas que tenham o mesmo custo (figura 2.4).

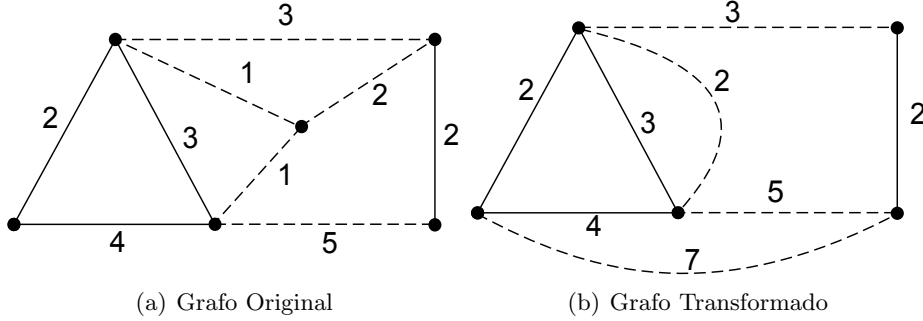


Figura 2.4: Transformação do grafo para o PCR Não Dirigido (adaptado de [9]).

Na figura 2.4 apresenta-se um exemplo da transformação acima descrita sobre um grafo G , onde se chama a atenção para as arestas a cheio como sendo o conjunto R de arestas, e os números a representarem o respectivo custo associado. No grafo transformado G' , o conjunto R induz p componentes conexas⁷ $G_1, \dots, G_p$ com os respectivos conjuntos de vértices $V_1, \dots, V_p$ que formam uma partição de V' ([22]).

2.2.2.2 O Problema do Carteiro Rural Dirigido

O PCR Dirigido define-se sobre um grafo $G = (V, A)$ onde A representa um conjunto de arcos dirigidos. O conceito de arco dirigido surge associado à travessia unilateral de um vértice i para um vértice j , que pode ser metaforicamente entendido como uma rua de sentido único. Ou seja, ainda que exista uma aresta que ligue i a j , a travessia entre esses dois vértices só poderá ser efectuada de i para j ou vice-versa. O problema pode ser reduzido a (no sentido de poder vir a ser interpretado como) um PCC sempre que $\overline{G} = (V, R)$ seja conexo e é também resolvido sobre um grafo modificado $G' = (V', A')$, construído da mesma forma que o grafo modificado do PCR Não Dirigido, com a seguinte alteração: os arcos (v_i, v_j) a introduzir em G' são agora dirigidos e têm comprimento c_{ij} equivalente ao comprimento do caminho mínimo entre v_i e v_j .

2.2.2.3 O Problema do Carteiro Rural Misto

O PCR Misto deve a sua designação por ser um problema que se encontra associado a um grafo *misto*, definido como $G = (V, E_1 \cup E_2)$, onde E_1 é um conjunto de arcos dirigidos e E_2 um conjunto de arestas não dirigidas. O problema é normalmente identificado com o Problema do Empilhador, onde os arcos de A são considerados como movimentos que o empilhador tem de efectuar, uma única vez, numa determinada direcção ([22]). Esta

⁷Componente conexa de G , ou simplesmente *componente*, designa todo o subgrafo conexo maximal de G .

variante do PCR é assim designada pela abrangência do grafo, que contém quer arestas (no sentido de arcos não dirigidos), quer arcos (dirigidos). Assim, pode concluir-se que as três variantes já abordadas se definem todas sobre o grafo $G = (V, E_1 \cup E_2)$. Se E_1 é vazio, trata-se a variante *Não Dirigida*, se E_2 é vazio trata-se a variante *Dirigida*, e define-se, por último, a variante *Mista* se E_1 e E_2 são ambos não-vazios.

2.2.2.4 O Problema do Carteiro Rural Ventoso e outros PCRs

No PCR Não Dirigido os custos associados às arestas (ou arcos) são equivalentes quer se efectue travessia de i para j , quer no sentido contrário. Assim, o PCR Ventoso surge com o objectivo de satisfazer, de modo mais realista, alguns dos problemas do quotidiano comum, onde essa equivalência de custos não se verifica. Metaforicamente, a ideia do PCR Ventoso sugere o trabalho que um carteiro *rural* teria de efectuar num dia ventoso, isto é, percorrendo ligações, ora com o vento a favor, ora com o vento contra.

Seja $G = (V, A)$, um grafo conexo e não dirigido com dois custos não negativos, c_{ij} e c_{ji} , associados a cada aresta $(i, j) \in A$, que representam o custo da travessia de i para j e de j para i , respectivamente, e R o conjunto de arestas *obrigatórias*, tal que $R \subseteq A$. O PCR Misto procura encontrar o caminho de custo mínimo que percorra, pelo menos uma vez, cada uma das arestas de R . É fácil observar que, pela definição de *custo*, quer o PCR Dirigido, o PCR Não Dirigido e o PCR Misto são casos especiais do PCR Ventoso ([2]).

Por último, é óbvio concluir que, dependendo das restrições impostas, o PCR pode fugir das abordagens de qualquer uma destas *grandes* variantes do problema, atrás referidas. Assim, e sem entrar em grandes detalhes, referenciam-se ainda o Problema do Carteiro Rural Ventoso com serviço em Zigzag proposto por Irnich em [18] e mais tarde em [19], o Problema do Carteiro Rural Coleccionador de Prémios por Aráoz, Fernández e Meza em [11], inicialmente definido por Aráoz, Fernández e Zoltan em [1], as subvariantes com penalizações e/ou proibições de caminho, quer no PCR Dirigido em [3], quer no PCR Misto em [5], o Problema do Carteiro Rural Periódico considerado por Ghiani et al em [13] e ainda, por último, o Problema do Carteiro Rural Dinâmico (PCRD), unicamente abordado por Moreira et al em [23] e que irá ser mais detalhado em seguida.

2.2.3 O Problema do Carteiro Rural Dinâmico

Dada a vastíssima literatura sobre o assunto, o PCR tem vindo a ser aplicado aos mais diversos contextos do mundo da engenharia (e não só), sofrendo constantes adaptações estruturais, mediante as condições que o envolvem. Deste modo, o PCRD, assim apelidado por em [23], surgiu no interesse de desenvolver uma aplicação informática para a determinação de percursos de corte numa indústria metalo-mecânica.

Tomando [23] como primeira referência sobre o assunto, entende-se o PCRD como uma variante do PCR onde o subconjunto A de arestas (ou arcos) inicial é diferente do subconjunto A' de arestas (ou arcos) final. Referindo ainda o já usado paralelismo

metafórico, entende-se o PCRD como um problema em que as ligações entre as diversas povoações a visitar vão surgindo à medida que o carteiro cumpre determinados requisitos. Por determinados requisitos entende-se a visita de um conjunto previamente definido de povoações (fazendo o paralelismo para o Problema da Determinação de Percursos de Corte, visitar todas as arestas de uma *povoação* equivale a dizer que se cortaram todas as arestas de uma *peça*). Numa perspectiva mais geral do problema, pode-se considerar que $A = A'$ com A a integrar um subconjunto de arestas de custo infinito.

O conceito de “dinâmico” surge, então, do facto da estrutura do grafo em questão ir integrando novas arestas com o desenvolver do percurso. Uma vez que se trata de uma variante que surgiu no âmbito de um problema concreto, torna-se evidente que será mais imediato perceber o PCRD, quando abordado, mais à frente, o problema da determinação de percursos de corte.

2.3 Determinação de Percursos de Corte

O mundo da Investigação Operacional (IO) reveste-se do objectivo último de obter soluções óptimas ou próximas da mesma para todos os seus problemas. No entanto, e porque a IO desloca a dimensão matemática para uma dimensão real, no campo da engenharia, sabe-se que existe um conjunto enorme de restrições que dificultam, ou em alguns casos impossibilitam, que se resolva um determinado problema até à optimalidade.

Na determinação de percursos de corte, cada problema apresenta as suas restrições específicas, o que constitui um problema diferente para cada situação, que foge aos padrões de *standardização* que se procuram atingir nas indústrias de hoje em dia na tentativa de minimização do uso dos seus mais importantes recursos: tempo e dinheiro. Por outro lado, um problema novo incita ao desenvolvimento de novas técnicas e abordagens heurísticas para satisfação das condições impostas e irá certamente contribuir de forma positiva para o conjunto de soluções conhecidas dos problemas de IO.

Seguindo a linha de raciocínio que tem vindo a ser usada e que partiu da generalização dos problemas, chega-se finalmente ao problema específico abordado neste trabalho de dissertação. O Problema da Determinação de Percursos de Corte (PDPC) pode ser entendido como a segunda de duas partes sequenciais de um problema maior, relacionado com o corte de peças de uma placa. Este problema “maior” integra, então, dois problemas distintos mas complementares e com o mesmo objectivo de minimização de custos: posicionamento de um conjunto predefinido de peças de formas irregulares numa dada placa e determinação do percurso de corte das mesmas, tal como se pode ver na figura 2.5. Relativamente ao primeiro, que, duma perspectiva meramente superficial, consiste em posicionar itens pequenos dentro de itens maiores ([24]), sabe-se que é um problema da classe de problemas *NP-hard* e que tem vindo a ser cada vez mais estudado e aprofundado, com recentes abordagens em [14] e [7]. No caso do PDPC, a ideia é, encontrada

a solução final do problema de posicionamento, encontrar um caminho mínimo a ser percorrido pela ferramenta de corte (fio de cobre electrificado, feixes de raios laser, maçarico, jacto de água, etc....) de modo a cortar todas as peças que se encontram dispostas na placa, satisfazendo as restrições impostas.

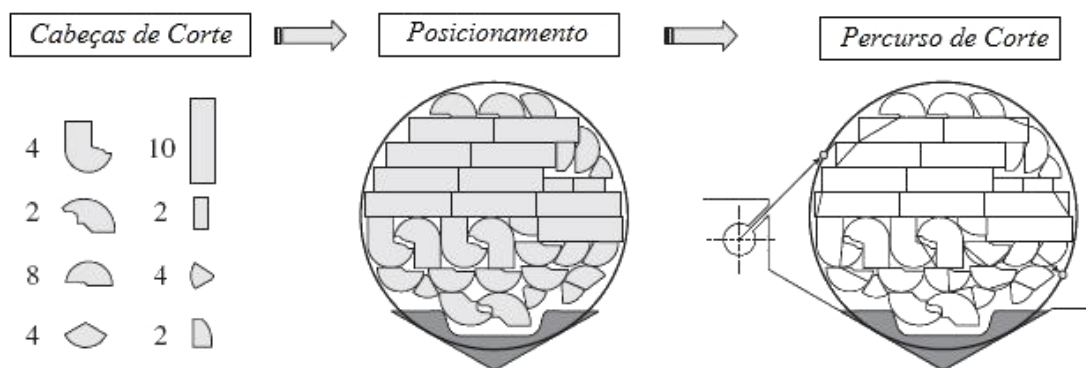


Figura 2.5: Enquadramento industrial de um PDPC (adaptada de [22]).

Os PDPCs encontram-se divididos em dois grandes conjuntos de problemas, relacionados com o processo tecnológico de corte utilizado que irá influenciar o modo de colocação da placa a cortar. Assim, definem-se os problemas de origem *estática* quando a placa onde se vai efectuar o corte se encontra apoiada (não há queda das peças), e os problemas de origem *dinâmica* quando a referida placa se encontra suspensa (as peças vão caindo à medida que se efectuam os respectivos cortes, o que provoca alterações na geometria da placa). A diferença base entre estes dois conjuntos de problemas reside nas alterações que a queda de uma peça provoca no grafo associado (i.e. criam-se novas visibilidades entre peças).

Assim, nas duas subsecções seguintes serão consideradas duas situações industriais do PDPC, com diferentes restrições, relativas a dois casos de estudo. Salienta-se o facto de ambas as situações analisadas serem o mesmo problema *dinâmico*, mas com diferentes abordagens. A principal diferença entre elas é precisamente o facto de uma tirar partido do referido dinamismo do problema na sua abordagem heurística, ao passo que a outra simplesmente não o faz, pelo que se dividiu, respectivamente, as subsecções seguintes em *abordagem “dinâmica”* e *abordagem “Force-Support Tree”* para evidenciar tal aspecto. Será efectuada uma descrição de cada caso, seguida de uma análise crítica focando as principais vantagens de implementação usadas e que foram aproveitadas no desenvolvimento deste trabalho.

2.3.1 Abordagem “Dinâmica”

Esta subsecção deste trabalho de dissertação refere-se especificamente a [23] e é sobre o trabalho aí desenvolvido que se estudou e se efectuaram as seguintes conclusões.

Neste primeiro caso do PDPC, existe um conjunto muito peculiar de restrições que, embora confirmam uma originalidade pouco comum ao problema, acrescem-no, obviamente, de maior complexidade. As restrições industriais do problema impõem uma patela circular suspensa, constituída por uma fina camada de diamante sobreposta a uma placa de tungsténio, presa por duas pinças na base, sendo a ferramenta de corte utilizada o fio de cobre electrificado, que, iniciando o processo de corte, não deixa a placa até que todas as peças colocadas tenham sido cortadas.

Deste modo torna-se imediato inferir três aspectos fundamentais do problema:

- Sendo a ferramenta de corte um fio de cobre electrificado, qualquer movimento de um ponto para outro na placa (exceptuando as situações de movimento em vazio⁸) corresponde a um corte efectivo irreversível;
- Estando a placa suspensa e presa por duas pinças na sua base, qualquer corte que atinja um ponto já cortado (exceptuando as situações em que ocorreu retrocedimento do fio de cobre) irá provocar a queda, para um contentor específico, de um pedaço de placa;
- A queda sequencial de pedaços da placa confere constantes alterações à componente geométrica do problema.

Para que se possa entender melhor a dimensão do problema em causa, convém entender as consequências subjacentes aos aspectos acima referidos.

As restrições inerentes à ferramenta de corte implicam que cada movimento seu terá de ser validado para que não ocorra a situação do corte do interior de uma peça. Assim, associado ao movimento do fio de cobre só poderá ocorrer: (1) o corte efectivo da aresta de uma peça; (2) o corte efectivo de uma porção de placa que não corresponde a nenhuma aresta, ou (3) o movimento em vazio. Por uma questão lógica, os movimentos do tipo 2 e 3 serão considerados *cortes em vazio* e encontram-se associados ao conceito de *pontes*.⁹ Convém salientar o facto de que o retrocedimento do fio de cobre por um caminho já percorrido é apenas um caso particular dos movimentos em vazio e que a solução final para o PDPC é a soma dos cortes do tipo 1 com os cortes em vazio. Assim, e uma vez que os cortes do tipo 1 são obrigatórios, o problema passa por minimizar o comprimento dos cortes equivalentes à segunda parcela (pelo simples facto de que existem variadíssimas maneiras de efectuar as transições correspondentes aos cortes em vazio).

Relativamente à restrição que coloca a placa suspensa e presa por duas pinças na base, e aliada à restrição acima citada, há que considerar a validação dos ciclos formados pela ferramenta de corte. Por ciclo entende-se que o percurso de corte irá encontrar um vértice com uma aresta incidente já cortada (equivalente a dizer que o fio de cobre já passou por esse mesmo vértice). Quando tal situação ocorre, sabe-se, pelas condições impostas

⁸Movimento associado à deslocação do fio de cobre em áreas da placa já cortadas.

⁹Segmentos de recta entre dois vértices de uma peça que não correspondem à aresta de nenhuma peça.

e descritas, que uma porção de placa irá cair, pelo que se torna crucial verificar se essa mesma porção de placa é constituída por algo que não seja uma só peça ou uma porção de desperdício.¹⁰ Assim, todas as combinações que incluam uma peça e desperdício, duas ou mais peças sem desperdício e, obviamente, duas ou mais peças com desperdício levam à formação de ciclos inadmissíveis e obrigam a uma alteração no rumo que o percurso iria tomar. Estas alterações de percurso encontram-se associadas a uma estrutura de memória que será mais à frente explicada, juntamente com o conceito de *admissibilidade do ciclo*.

Ainda sobre esta restrição convém salientar que, pelo facto da placa se encontrar presa pela base, torna-se necessário efectuar um percurso de corte que siga uma trajectória geral *de cima para baixo*, de modo a que a placa não fique sem suporte. Deste modo, por uma questão de conveniência, irá considerar-se que o sentido positivo do eixo das ordenadas é de cima para baixo e o sentido positivo do eixo das abcissas, da esquerda para a direita.

Sobre as constantes alterações à forma da placa, provocadas pela queda de peças ou de porções de desperdício, convém salientar um aspecto importante: a cada alteração da forma geométrica da placa estará sempre associada uma alteração das *visibilidades* entre dois vértices, o que irá provocar alterações no subconjunto A , de arestas (ou arcos) existentes no grafo com um custo c_{ij} associado. Assim, facilmente se conclui que o grafo G terá um carácter dinâmico, consequência directa destas alterações de geometria da placa. Antes de se aprofundar um pouco mais o assunto, convém clarificar alguns conceitos básicos associados à geometria do problema.

- *Vizinhança de Polígonos*: por vizinhança de um polígono P_i entende-se um qualquer conjunto de polígonos $V \in P$ tal que P_j apresenta pelo menos um contacto, sem sobreposição, com o polígono P_i . Deste modo, apenas as situações de *vértice comum* (em que existe contacto de um vértice do polígono P_i com um vértice do polígono P_j) e de contacto de um vértice de P_i com uma aresta de P_j (e vice-versa) colocam P_i vizinho P_j e, consequentemente, P_j vizinho de P_i . Este conceito de vizinhança é calculado com base num conjunto de funções geométricas, das quais se salienta o cálculo da *envolvente rectangular*, ortogonal aos eixos de coordenadas, de um polígono. Usando as coordenadas extremas de um polígono é possível obter os dois vértices necessários à construção da sua envolvente, nomeadamente o vértice *SuperiorEsquerdo*, que terá os menores valores de abcissa e ordenada de todo o conjunto de pontos (vértices) do polígono e o vértice *InferiorDireito*, que terá os maiores valores de abcissa e ordenada de todo o conjuntos de pontos do polígono, tal como se pode ver na figura 2.6. É óbvio notar que, com apenas os dois referidos vértices, é possível construir o referido rectângulo, uma vez que este tem de ter os seus lados paralelos aos eixos ortonormados considerados.

A determinação da envolvente rectangular de um polígono e o teste de contacto ou sobreposição de duas envolventes, são processos extremamente simples mas de enorme

¹⁰Porção de placa que não contém peça.

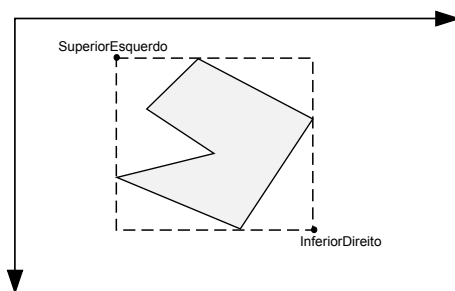


Figura 2.6: Envoltente rectangular de um polígono.

utilidade. Assim, como se torna evidente, o teste de vizinhança entre dois polígonos só será efectuado nos casos em que se verifica o referido contacto ou sobreposição das envoltentes rectangulares.

- *Arestas*: Sabendo-se *à priori* que o PDPC segue a solução encontrada para um problema de posicionamento, é imediato concluir que irá existir uma multiplicidade de contactos entre as arestas dos vários polígonos, o que irá originar um conjunto de *cortes parciais*. Numa tentativa de resolver o problema, de elevada complexidade, relacionado com o facto de se saber, em cada instante, quais as arestas já cortadas e a localização do(s) corte(s) já efectuado(s) sobre elas, optou-se por efectuar um *seccionamento* das arestas em que se verifica contacto ou sobreposição de outras. Assim, torna-se suficiente saber, em cada instante, quais as arestas que já foram cortadas para saber o estado de cada polígono, tendo em consideração que cada aresta poderá pertencer apenas a dois polígonos simultaneamente, dada a bidimensionalidade do problema. Considerando ainda o caso em que duas arestas (já seccionadas) se encontram perfeitamente sobrepostas (isto é, os seus vértices final e inicial coincidem), o problema reduz-se a estas arestas *duplas*, ou, nos restantes casos, a arestas *simples*, que tomarão sempre qualquer um dos seguintes estados binários: *cortada* ou *não cortada*.
- *Visibilidade*: Derivando directamente das restrições atrás mencionadas, o conceito de *visibilidade* entre dois vértices, ou, num sentido mais lato, entre dois polígonos é fundamental no PDPC. Tal como foi referido, qualquer movimento efectuado pela ferramenta de corte (com a excepção dos *cortes em vazio*) efectua um corte irreversível. Assim, torna-se necessário validar cada movimento para garantir a integridade física dos polígonos colocados na placa, ou seja, o segmento de recta $\overline{v_i v_j}$, equivalente ao corte orientado do vértice v_i para o vértice v_j (que podem pertencer ou não ao mesmo polígono), não pode violar o interior de nenhum polígono, sob pena de inutilização do mesmo. Ainda, cada movimento de v_i para v_j tem de ser *minimal*, no sentido de que não poderão existir quaisquer vértices no segmento $\overline{v_i v_j}$, o que é

garantido pelo *seccionamento* de arestas acima descrito. O conceito de movimento *minimal* equivale, portanto, a cortar *uma aresta de cada vez*.

Assim, considera-se que v_j é *visível* a partir de v_i , se um possível corte do segmento $\overline{v_i v_j}$ não viola o interior de qualquer outro polígono e se esse corte se encontra associado a apenas uma aresta. Generalizando, dir-se-á que um polígono P_k é *visível* a partir de um outro polígono P_w , se algum dos vértices de P_k for *visível* a partir de algum dos vértices de P_w . Deste modo, assume-se que a existir visibilidade entre dois vértices, existirá visibilidade entre os polígonos que os contêm.

Antes de se detalhar os aspectos inerentes ao conceito de visibilidade, convém explicar que os polígonos em questão são constituídos por um conjunto ordenado de pontos (vértices), assim como os segmentos de recta que os unem (arestas), igualmente ordenado. De ora em diante, e tomando o sentido horário do relógio para definir a ordem dos vértices de um determinado polígono, ter-se-á definido o lado direito das arestas como o *interior* do polígono e, naturalmente, o lado esquerdo das arestas como o *exterior* do polígono.

O motivo pelo qual dois vértices v_i e v_j se podem considerar *invisíveis* tem uma de duas origens. A primeira, verifica-se quando $\overline{v_i v_j}$ intersecta um obstáculo, nomeadamente uma aresta ou um vértice do mesmo ou outro polígono. A segunda, de natureza mais complexa, ocorre quando $\overline{v_i v_j}$ sobrepõe o interior de um polígono, sem intersectar nenhum obstáculo. Estas situações são definidas como *Invisibilidade Directa* e *Invisibilidade Indirecta*, respectivamente, e encontram-se ilustradas em 2.7. Tal como estudado em [22], refere-se que no caso de *invisibilidade indirecta*, o polígono cujo interior é violado terá de conter simultaneamente v_i e v_j , pois caso contrário, a existir invisibilidade, teria sido verificada no teste da *invisibilidade directa*. No entanto, esta situação é demasiado redutora e não prevê o caso específico que será apresentado e explicado mais à frente neste trabalho, na secção 4.4.

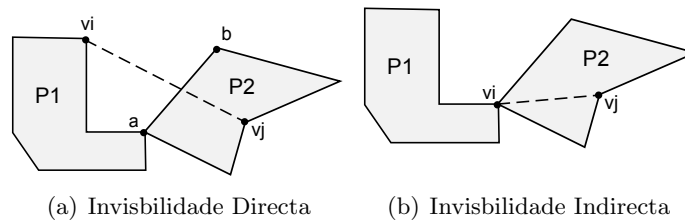


Figura 2.7: Diferentes tipos de visibilidade entre dois vértices v_i e v_j .

Para terminar esta abordagem relativa ao conceito de visibilidade, apresenta-se a noção de *área de influência* de uma peça P_i , que consiste no conjunto de peças colocadas na placa cujas relações de visibilidade são influenciadas pelo posicionamento de P_i . Significa isto que a remoção de P_i da placa só irá afectar as visibilidades das peças que se encontravam na sua área de influência. Assim pode-se concluir que a

área de influência de uma peça P_i corresponde ao seu conjunto de peças visíveis. Este conceito surge na perspectiva de tornar o algoritmo computacionalmente menos pesado, uma vez que se minimiza o número de cálculos a efectuar, relacionados com a actualização das visibilidades sempre que ocorre queda de uma peça.

- *Memória*: Neste problema específico, o uso de uma *memória* encontra-se associado à formação de ciclos inadmissíveis, quando se testa a validade de um vértice a introduzir no percurso. Portanto, quando um vértice gera um ciclo inadmissível (definido como *vértice ciclo*) não é incluído no percurso e este é redireccionado através de um mecanismo designado, neste contexto, de “*backtracking*” por *pesquisa em árvore*, que será explicado em seguida. Esta memória regista todos os *vértices ciclo* e respectivas tentativas de redireccionamento para que se possa evitar a reincidência em percursos que já se sabe *não levarem a lado nenhum*. Esta memória funciona como a *inteligência* da heurística abordada, na medida em que é uma estrutura crucial para o bom funcionamento da aplicação sem no entanto ser impeditiva de elevada eficiência da mesma. Para isso contribui a sua forma de implementação que prevê um *reset* absoluto à memória sempre que uma das seguintes situações ocorre: queda de peça ou formação de ciclo admissível gerado pelo corte de uma aresta. Assim, quando qualquer uma das situações atrás descritas ocorre, a memória é apagada, evitando-se assim a evolução desmesurada de uma estrutura tão influente na determinação, com sucesso, de um percurso de corte. Convém, no entanto, referir que o apagar da memória (associado a qualquer uma das situações referidas) não significa que determinados redireccionamentos de percurso outrora inadmissíveis se tenham tornado admissíveis. O apagar deliberado de erros passados terá então o *preço a pagar* de recalculer esses percursos e reescrevê-los na memória, sendo este processo de *aprendizagem/desaprendizagem* da memória a garantia de funcionamento do sistema.

É de salientar que o percurso realizado durante a última e a próxima queda de peça / formação de ciclo admissível gerado pelo corte de uma aresta, não implica necessariamente a formação de alguma memória, assim como convém referir que na estrutura de memória não figuram caminhos repetidos. Pode sim acontecer que existam n caminhos diferentes que levem ao mesmo *vértice ciclo*.

Na figura 2.8 é possível observar e entender melhor o funcionamento do mecanismo acima descrito através de um fluxograma de alto nível.

Para não se conferir um desnecessário grau de complexidade à explicação desta estrutura, irão apenas referir-se quais as três situações distintas em que o processo de retrocedimento de percurso pára. Assim, caso o movimento entre o vértice actual e o vértice candidato a vértice seguinte seja identificado como não inadmissível relativamente à formação de ciclos, só serão considerados como candidatos a vértice seguinte os vértices atingíveis por arestas não cortadas. Se tal situação não ocorrer,

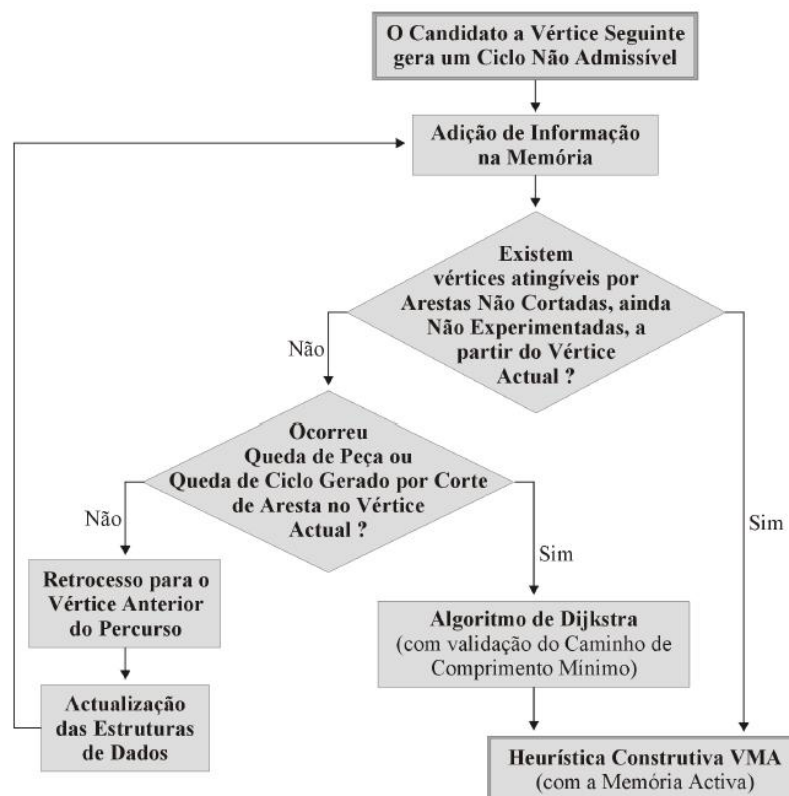


Figura 2.8: Fluxograma do algoritmo de “backtracking” por pesquisa em árvore (adaptado de [22]).

irá proceder-se a um retrocesso ao longo do percurso de corte até que ocorra uma das seguintes possibilidades: *encontrou-se alternativa*, quando a partir do vértice actual existem ainda vértices atingíveis por arestas não cortadas ainda não experimentadas; *encontrou-se queda de peça*, quando o novo vértice actual, ao ser inserido no percurso de corte numa iteração anterior, originou o corte completo de uma peça; ou *encontrou-se queda de porção da patela gerada pelo corte de uma aresta*, quando o novo vértice actual, ao ser inserido no percurso de corte numa iteração anterior, originou um ciclo admissível (equivalente a uma porção da placa não correspondente a uma peça) cujo último corte foi uma aresta.

- *Ciclos de Corte Não Admissíveis*: O conceito de ciclo admissível foi já abordado anteriormente, quando se explicou o conceito de *memória*. Ainda que relativamente auto-explicativo nas situações em que foi descrito, salientam-se aqui alguns aspectos relevantes intrínsecos ao seu processo de funcionamento, nomeadamente as duas fases do processo de validação da admissibilidade de um ciclo. Primeiro, ocorre a validação das arestas não cortadas incidentes a pelo menos um dos vértices que constituem o ciclo e, em seguida, ocorre a validação das restantes arestas não cortadas (e, naturalmente, ainda não validadas). Relembrando as situações em que um ciclo se diz *inadmissível*, referidas no início desta secção, fica assim definido que um ciclo diz-se *admissível* se no seu interior não existirem peças que ainda não tenham sido totalmente cortadas (não existem arestas não cortadas no interior do ciclo), ou seja, se, testadas todas as arestas não cortadas (incidentes e não incidentes a vértices do ciclo), não se detectar nenhuma inadmissibilidade, o ciclo é validado como *ciclo admissível*.

2.3.1.1 Critério-VMA

Em [22] são estudados dois casos distintos, cada um associado a uma heurística de resolução diferente, nomeadamente o caso em que a placa de diamante e tungsténio se encontra totalmente apoiada e o caso em que esta se encontra suspensa e presa por duas pinças, com o critério de decisão “Vértice Mais Próximo” (VMP) e “Vértice Mais Acima” (VMA), respectivamente. Ainda que o primeiro caso (VMP) seja uma clara simplificação do segundo, uma vez que certas restrições deixam de ser consideradas, e que, deste modo, possa constituir uma natural primeira abordagem ao problema em questão, o trabalho desenvolvido nesta dissertação desenvolveu-se exclusivamente para o caso da placa suspensa e todas as restrições inerentes (VMA), acima referidas.

Deste modo, foi analisada a heurística “Vértice Mais Acima” (VMA) que, dadas as restrições referidas, corta uma aresta da cada vez, seguindo um critério de escolha do *vértice seguinte*, dado um *vértice actual*, como uma conjugação de dois factores:

1. Cortar, prioritariamente, as peças mais afastadas do suporte da placa;
2. Minimizar o corte em vazio.

Relativamente ao primeiro aspecto, e por não constituir uma restrição *forte*, existe uma certa flexibilidade na maneira como é abordado, desde que respeitando uma trajectória *de cima para baixo*, tal como já referido. Deste modo, sempre que a partir do *vértice actual* existam candidatos a *vértice seguinte*, a escolha irá sempre recair no vértice com menor¹¹ valor de ordenada. Em caso de empate, ou seja, no caso de existir mais do que um candidato com o mesmo valor de ordenada, escolher-se-á aquele com menor valor de abcissa, equivalente a definir uma trajectória global da *esquerda para a direita* na placa.

Ligado ao segundo aspecto encontra-se obviamente a situação em que não existem candidatos a *vértice seguinte*, atingíveis por arestas não cortadas. Tal significa que o movimento efectuado terá de ser necessariamente em vazio (ou através de arestas já cortadas ou através de pontes). Assim, gozando da flexibilidade do primeiro aspecto referido, a escolha do *vértice seguinte*, dadas as condições, recairá sobre o candidato¹² *mais próximo* do *vértice actual*. No entanto, caso exista um, ou mais, candidatos à mesma distância, escolhe-se o vértice que tiver menor valor de ordenada (*mais acima*). Em caso de novo empate escolher-se-á o vértice com menor abcissa (*mais à esquerda*).

Naturalmente que, em ambas as situações referidas, a escolha do *vértice seguinte* está sujeita a validação relativamente à formação de ciclos inadmissíveis, com a excepção de ser atingível por uma aresta já cortada (como já referido).

Assim, em jeito de resumo, o critério-VMA pode ser entendido como a escolha do vértice mais acima (se atingível por aresta não cortada), ou do vértice mais próximo do vértice actual (se atingível por ponte ou aresta já cortada), com pelo menos uma aresta incidente não cortada, que não se encontre num caminho experimentado sem sucesso (se a memória estiver activa) e que não forme um ciclo não admissível.

Considerando os seguinte métodos e estruturas de dados, apresenta-se em 2.9 a heurística construtiva VMA em pseudocódigo de alto nível:

- *v_actual* - vértice actual;
- *v_anterior* - vértice anterior ao vértice actual;
- *candidato_v_seguinte* - vértice candidato a vértice seguinte que, neste contexto, pode subentender um caminho de comprimento mínimo até ao vértice seguinte, propriamente dito;
- *v_seguinte* - vértice candidato a vértice seguinte, após este ter sido validado relativamente à formação de ciclos;
- *peças* - vector que contém todas as coordenadas de posicionamento de todas as peças colocadas na placa;
- *matriz* - matriz multidimensional de visibilidades

¹¹Neste caso específico em que os *yy* é crescente de cima para baixo

¹²São considerados candidatos todos os vértices visíveis com arestas adjacentes por cortar.

- *padrão_de_corte* - descrição poligonal das peças posicionadas na placa;
- *patelaInOut* - coordenadas dos vértices de entrada e de saída da placa;
- *memória* - estrutura de memória da heurística, na qual se encontram registados os caminhos experimentados sem sucesso;
- *percurso_de_corte* - vector de vértices que constitui uma solução para o PDPC;
- v_j - vértice genérico;
- P_i, P_k - peças genéricas;
- **constrói_estruturas_de_dados**(*padrão_de_corte*) - método responsável pela construção de todas as estruturas de dados necessárias ao funcionamento da heurística, das quais se destacam a *matriz das visibilidades* e a *lista de arestas*;
- **inicializa_percurso**(*patelaInOut, peças*) - método que inicializa o percurso de corte, com base nas coordenadas do ponto de entrada da placa e das coordenadas dos vértices das peças posicionadas;
- **determina_peça_actual**(*v_actual, peças*) - método que determina a peça que contém o vértice actual;
- **classifica_vértice**(*v_actual, v_j*) - método que classifica o tipo de ligação entre o vértice v_j e o vértice actual. Retorna “aresta não cortada”, “aresta cortada” ou “ponte” (no caso de serem visíveis), e regista em memória as melhores ligações que é possível estabelecer a partir do vértice actual;
- **critério_VMA**() - método que determina qual o candidato a vértice seguinte (ou o caminho de comprimento mínimo até ele) com base nas melhores ligações registadas no método **classifica_vértice**, segundo o critério VMA, atrás explicado;
- **corte_vazio_consecutivo**(*v_actual, candidato_v_seguinte*) - método que impede movimentos desnecessários por pontes ou arestas já cortadas;
- **valida_candidato**(*v_actual, candidato_v_seguinte*) - método que avalia os efeitos resultantes da introdução do candidato a vértice seguinte (ou o caminho de comprimento mínimo até ele) no percurso de corte, relativamente à formação de ciclos;
- **redirecciona_percurso**(*v_actual*) - método no qual se aplica o processo de redireccionamento do percurso de corte, através do mecanismo de *backtracking* por pesquisa em árvore, atrás descrito. É também neste método que se constrói a estrutura *memória* (se aplicável);
- **introduz_no_percurso**(*v_seguinte*) - método que introduz o vértice seguinte (ou o caminho de comprimento mínimo até ele) no percurso de corte;

- **retira_do_percurso**(v_actual) - método que retira do percurso de corte o vértice actual (actualizando quer a dimensão do percurso, quer o novo vértice actual);
- **actualiza_corte**(v_actual , $v_seguinte$) - método que actualiza o estado físico das peças decorrente das alterações provocadas pelo corte efectuado. Se alguma peça for totalmente cortada, ou se se formar um ciclo admissível gerado pelo corte de uma aresta, este método é também responsável pela actualização das visibilidades e destruição da estrutura de *memória*;
- **actualiza_retrocesso**(v_actual , $v_anterior$) - método que actualiza o estado físico das peças decorrentes das alterações provocadas pelo retrocesso do percurso de corte;
- **determina_caminho_mínimo**(v_actual , *memória*) - método que determina o caminho de comprimento mínimo (válido relativamente à formação de ciclos) até ao vértice seguinte, com base no algoritmo de Dijkstra. As restrições da estrutura de *memória* (se activa) são preponderantes;
- **todas_as_peças_cortadas**() - método que determina se todas as peças já foram totalmente cortadas da placa;
- **finaliza_percurso**(v_actual) - método que finaliza o percurso de corte após todas as peças estarem completamente cortadas da patela e que será brevemente explicado mais à frente.

Esta abordagem apresenta bons resultados computacionais e constitui a base de partida para o trabalho a ser desenvolvido. Apesar de se analisar outra perspectiva em seguida, a heurística apresentada é, naturalmente, a mais adaptada às condições específicas deste problema.

2.3.2 Abordagem “Force-Support Tree”

Esta subsecção apresenta uma estratégia diferente na resolução do mesmo PDPC. Tal como na abordagem *dinâmica*, também nesta abordagem ([17]) existe um conjunto muito peculiar de restrições associadas, que derivam das condições físicas impostas pela realidade industrial. Uma vez que ambas as abordagens figuram no mesmo contexto, é possível perceber que algumas restrições de uma abordagem pudessem servir a outra, e vice-versa, pelo paralelismo dos problemas por vezes existente. No entanto, ambas foram analisadas tal como se encontram originalmente descritas, ficando para uma fase posterior a possível aplicação de restrições e métodos de resolução de uma na implementação da outra.

Esta abordagem procura fugir às abordagens de programação matemática mais comuns e dar lugar a uma perspectiva de resolução mais geométrica, onde as restrições não foram representadas por (in)equações mas sim tratadas directamente como diagramas e respectivos grafos associados. Assim, procura representar-se as restrições de modo directo

algoritmo heurística VMA()

```

    constrói_estruturas_de_dados(padrão_de_corte);
    v_actual = inicializa_percurso(patelaInOut, peças);
     $P_k$  = determina_peça_actual(v_actual, peças);
    repete
        para todo  $P_i \in peças$ 
            se  $matriz(P_k, P_i) == 1$  então
                para todo  $v_j \in P_i$ 
                    classifica_vértice(v_actual,  $v_j$ );
                candidato_v_seguinte = critério_VMA();
            se  $corte\_vazio\_consecutivo(v\_actual, candidato\_v\_seguinte) == true$ 
                então
                    retira_do_percurso(v_actual);
                    actualiza_retrocesso(v_actual, v_anterior);
                    candidato_v_seguinte = determina_caminho_mínimo(v_actual,
memória);
                se  $valida\_candidato(v\_actual, candidato\_v\_seguinte) == true$  então
                    v_seguinte = candidato_v_seguinte;
                    introduz_no_percurso(v_seguinte);
                    actualiza_corte(v_actual, v_seguinte);
                senão
                    redirecciona_percurso(v_actual);
    enquanto  $todas\_as\_peças\_cortadas() == false$ ;
    finaliza_percurso(v_actual);
    retorna percurso_de_corte;

```

Figura 2.9: Representação em pseudocódigo de alto nível da heurística construtiva VMA.

e simples sem incorrer em alargamentos ou reduções do espaço de procura do problema, como frequentemente acontece em processos de relaxação contínua ou de aplicação de heurísticas, respectivamente.

Nesta situação descrita em [17], as peças a cortar são *cabeças de corte*¹³ e são colocadas numa placa circular de diamante policristalino suspensa e presa por uma máquina e a ferramenta de corte é um fio de cobre electrificado. Assim torna-se necessário esclarecer alguns aspectos relativos a este problema:

- As peças a cortar formam uma região conexa simples, delimitada por uma sequência de arestas, que podem ser segmentos de recta ou arcos circulares. Caso seja um arco circular, a concavidade será sempre voltada para *dentro*, ou seja, o lado que contém o centro do arco pertence ao interior da peça. Dada a geometria das peças em questão, o número vértices (ou arestas) de cada peça encontra-se limitado por uma constante;
- A ferramenta de corte é um fio de cobre electrificado, de diâmetro não nulo. Consequentemente, o percurso de corte não será uma linha ideal, mas terá uma largura não nula;
- A placa de diamante policristalino encontra-se suspensa e presa por uma máquina enquanto decorre o processo de corte, pelo que haverá uma área da placa que não poderá conter peças;
- As *cabeças de corte* que irão ser cortadas da placa têm de encaixar perfeitamente, em uma ou mais arestas de *contacto*, com o corpo da ferramenta de corte.

Dadas as similaridades de algumas premissas dos dois PDPC, é possível que alguns dos aspectos relacionados com as restrições dos problemas venham a ser naturalmente revisitados, no entanto com diferentes propostas de resolução.

Assim, para melhor se entender a dimensão do problema e ser possível organizar estruturadamente as condições iniciais do problema, de modo a compreender intuitivamente uma abordagem para a sua resolução, serão aqui expostas as cinco restrições abordadas neste PDPC.

1. Uma vez que a ferramenta de cobre possui uma forma geométrica circular de diâmetro não nulo, é imediato concluir que se torna necessário alargar a forma de cada peça antes destas serem colocadas na placa. Ainda que um alargamento equivalente ao raio do fio de cobre seja suficiente para garantir a integridade física original das peças, a verdade é que a forma de uma peça já cortada não corresponde à forma final esperada, devido à sua *projecção*. Considere-se que o fio inicia o corte num ponto qualquer p_1 e que segue todas as arestas da peça. A peça em questão irá ficar

¹³Se se considerar uma ferramenta de corte comum, esta é constituída por um corpo (de material relativamente duro) e uma cabeça de corte (de material super-duro) que será integrada no corpo.

totalmente cortada quando o fio atingir p_2 , tal como se pode ver na figura 2.10. A área restante que fica por cortar na peça, resultante do diâmetro não nulo do fio de cobre, é definida como *projecção*. Assim, torna-se necessário distinguir as arestas de *contacto*, acima referidas, como arestas de *projecção proibida*, pela sua necessidade de encaixe *exacto* com o corpo da ferramenta e pelo processo de limagem a que todas as outras arestas (definidas como arestas de *projecção permitida*) estarão sujeitas após a montagem da cabeça com o corpo da ferramenta de corte e que eliminará a *projecção* em causa.

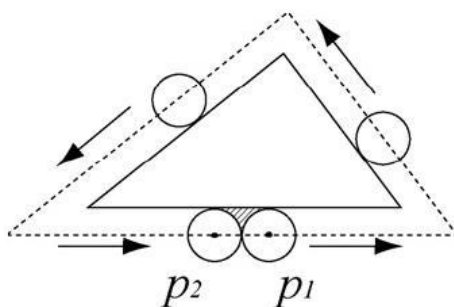


Figura 2.10: Projeção criada pelo fio de cobre electrificado (adaptada de [17]).

Assim, o facto do percurso de corte ter de acabar obrigatoriamente de cortar cada peça numa aresta de *projecção permitida*, relativa à própria peça, constitui a **primeira restrição**.

2. A **segunda restrição** deste problema relaciona-se com mais uma imposição física do contexto industrial e consiste em restringir o corte de cada peça à sua forma final. Isto deve-se ao reduzido tamanho das peças que torna virtualmente impossíveis quaisquer cortes adicionais num pós-processamento, pelo que cada peça deve ser cortada da placa já na sua forma final, sem conter um porção de *desperdício* ou outras peças agarradas.
3. Seja $(a_1, a_2, \dots, a_k)$ uma sequência circular anti-horária de arestas de uma peça. Uma aresta a diz-se *aberta* se o seu lado de fora está *vazio* e diz-se *fechada* quando ainda existe porção de placa (com ou sem peça) nesse mesmo lado. Pela **segunda restrição** pode então concluir-se que, antes da queda de uma peça, existirá uma sequência de arestas que pode ser dividida em duas sequências: *aberta* e *fechada*. Sejam a_i e a_j a primeira e última aresta, respectivamente, da sequência que foi definida como *fechada*. Se se projectarem ao longo do vértice terminal as arestas (inicial e final) da sequência de arestas *abertas*, tal como na figura 2.11 (adaptada de [17]), a_i e a_j formarão um ângulo θ , definido como *ângulo aberto*, que será positivo (2.11(b)) se as projecções não se cruzarem e negativo (2.11(a)) caso contrário.

Dada a espessura da placa de diamante policristalino, pode acontecer que uma peça seja totalmente cortada mas que não caia efectivamente da placa, ficando equilibrada

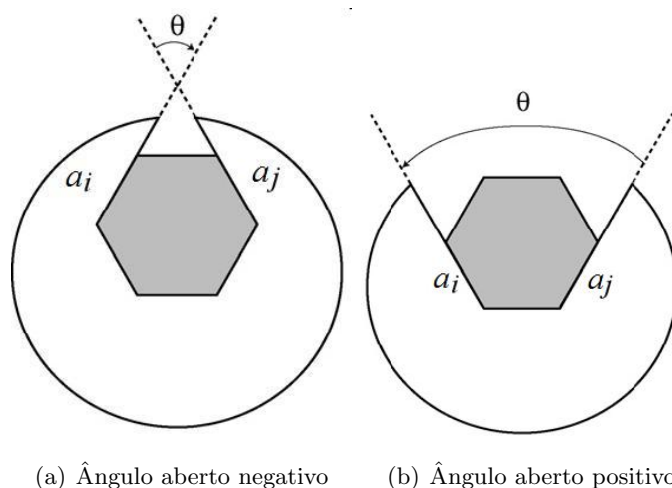


Figura 2.11: Ângulo aberto a representar a porção de placa, em torno da peça, que se encontra *vazia* (adaptada de [17]).

pela restante porção de placa que a circunda. Esta situação não deve acontecer pelo simples facto de que a ferramenta de cobre poderá mais tarde sobrepor aquela área (que se supunha *vazia*), danificando a peça em questão. Para que tal não se verifique, torna-se necessário garantir que, aquando do corte de uma peça, esta se encontra numa zona com espaço suficiente que permita a sua queda. Assim, a **terceira restrição** impõe que o *ângulo aberto* seja positivo quando tal situação vai ocorrer.

4. Para evitar situações de vibração, que podem danificar o fio de cobre ou a placa e ainda para assegurar que as porções restantes de placa por cortar serão resistentes o suficiente à passagem do fio (que, por motivos óbvios, é colocado em movimento à sua velocidade máxima permitida, aumentando deste modo a força exercida na placa quando em corte), introduz-se o conceito de *diâmetro mínimo local*.

Sejam p e q dois pontos distintos que se movem continuamente nas bordas do material restante e $d(p, q)$ a distância euclidiana entre eles. Esta distância $d(p, q)$ é definida como um *diâmetro mínimo local* se não toma valores inferiores quando p e q se movem na sua vizinhança. Assuma-se que as linhas a cheio correspondem às bordas do material e que as linhas a traço interrompido correspondem aos *diâmetros mínimos locais* na figura 2.12.

A **quarta restrição** obriga todos os *diâmetros mínimos locais* do material que falta cortar, exceptuando-se os casos em que estes são criados pelo ponto actual do fio de cobre, a serem maiores que uma distância predefinida d_0 .

5. Tal como é de esperar, dada a geometria circular em questão, o posicionamento das peças na placa irá certamente originar inúmeras áreas de desperdício em torno das peças e ao longo de toda a placa. Assim, para satisfazer a **terceira restrição** e criar o pretendido espaço *livre* à volta das peças colocadas, torna-se necessário dividir as

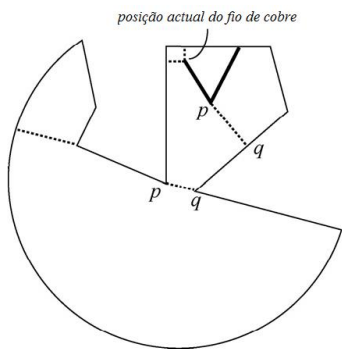


Figura 2.12: Diâmetros mínimos locais (adaptada de [17]).

regiões de desperdício e cortar também algumas destas sub-regiões. Deste modo, a **quinta restrição** impõe que quando se vai processar o corte de uma sub-região de desperdício, o seu *ângulo aberto* seja positivo.

Tomando como entrada do problema o conjunto inicial constituído por um posicionamento qualquer de polígonos numa área circular e um conjunto de arestas já etiquetadas de *projecção permitida* ou *projecção proibida*, o PDPC nesta situação passa por encontrar um percurso que satisfaça as cinco restrições. A abordagem utilizada consistiu no desenvolvimento de três algoritmos sequenciais até se obter o percurso final, onde a saída do primeiro é a entrada do segundo e a saída do segundo é a entrada do terceiro, sendo a saída do terceiro, naturalmente, o percurso final resultante. Assim, serão analisados os três algoritmos assim como será exposta a estrutura da sua implementação, em pseudocódigo de alto nível.

Tal como já explicado, as regiões de desperdício circundam as peças colocadas na placa e originam, na maior parte das vezes, ângulos abertos negativos, pelo que se torna necessário subdividir essas regiões e cortar algumas fora de modo a tornar os ângulos abertos das peças positivos. Assim desenvolveu-se um algoritmo para realizar a *partição de uma região em regiões quase convexas*, que toma como valores de entrada, o conjunto inicial acima referido.

Seja $\epsilon = 0.1$ uma constante de pequeno valor e seja X uma região bidimensional delimitada por uma sequência de segmentos de recta e arcos circulares. X é considerada uma região *quase convexa* se todos os ângulos internos têm valor não superior a $\pi + \epsilon$. Um vértice cujo ângulo interno é superior a $\pi + \epsilon$ é definido de vértice *fortemente reflexo*.

Seja v um vértice *fortemente reflexo* da região R , s o ângulo interno de v e t a transformação $t = s - (\pi + \epsilon)$. Tal como podemos ver em 2.13 a área escurecida é dividida em v em três áreas diferentes com ângulos t , $\pi + \epsilon - t$ e t , respectivamente, sendo a área central designada como área *manuseável*. Se se efectuar uma partição de R , através de um segmento de recta que atravessasse essa área *manuseável* e que ligue v a um ponto da borda R , irão obter-se duas sub-regiões R_1 e R_2 cada uma associada a um ângulo interno s'_1 e s'_2 , respectivamente, não *fortemente reflexos*. Assim, conclui-se que a partição de R

em duas sub-regiões irá sempre provocar a diminuição, em pelo menos um, do número de vértices *fortemente reflexos*.

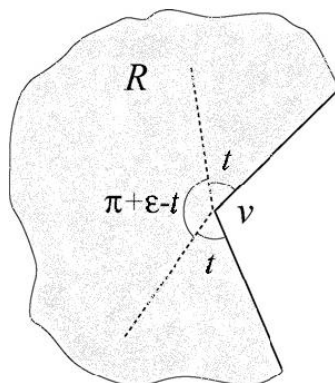


Figura 2.13: Divisão da área a sombreado, pelo vértice v , em três partes diferentes (adaptada de [17]).

O algoritmo (1), em pseudocódigo de alto nível, segue a seguinte orientação:

1. Inicializar S com uma determinada região, tal que S só contenha essa mesma região;
2. Se todas as regiões em S são *quase convexas*, retornar S e terminar. Se não, escolher e eliminar uma região R não *quase convexa* de S ;
3. Escolher um vértice *fortemente reflexo* v de R ;
4. Encontrar o ponto da borda de R que se encontra na área *manuseável* e com menor distância a v ;
5. Dividir a região R pelo segmento de recta formado por v e p em duas sub-regiões e colocá-las em S . Voltar a (2).

Este algoritmo divide as regiões de desperdício em sub-regiões *quase convexas* e atribui uma etiqueta *projecção permitida* a ambos os lados de todas as arestas que surgem da partição de regiões. Dado que na placa teremos agora apenas peças e sub-regiões de desperdício, por uma questão de comodidade, será tudo igualmente considerado sob o conceito de *partes*.

Se considerarmos n como o número de peças, então $O(n)$ é o número de partes existentes, uma vez que o número de vértices e arestas de cada peça é limitado por uma constante, tal como referido no primeiro aspecto relativo a este problema, no início desta subsecção.

Pondo de parte a ideia básica e exponencialmente crescente com n de enumerar todas as permutações das partes e encontrar, se possível, um caminho de comprimento mínimo associado a cada permutação para se poder escolher o percurso de menor distância, surge a ideia de implementar uma simples heurística baseada no vizinho mais próximo. No

entanto, dado o elevado grau de complexidade das restrições e a necessidade de garantir o sucesso em tempo útil, procurou desenvolver-se um mecanismo heurístico que reduzisse a área de procura aos percursos viáveis. Surge assim o conceito de *árvore de suporte sustentado*. Para uma dada solução do algoritmo anterior (ou seja, uma partição da placa em partes), considera-se uma árvore com um conjunto de nós N e um conjunto de arcos A que satisfazem as seguintes condições:

- O conjunto de nós N é o conjunto de todas as partes colocadas na placa;
- A raiz corresponde à maior subregião de desperdício;
- Se uma parte p_1 é *pai* de uma parte p_2 , então p_1 e p_2 partilham um aresta cujo comprimento é maior que o limite d_0 (quarta restrição) e que possui a etiqueta de *projecção permitida*, relativa a p_2 .

Dadas as condições supracitadas, o algoritmo (2) que irá gerar uma *árvore de suporte sustentado* encontra-se assim implementado, em pseudocódigo de alto nível:

1. Encontrar maior região de desperdício e torná-la *raiz*;
2. Colocar a raiz no espaço E_1 e todas as restantes partes no espaço E_2 ;
3. Escolher e eliminar uma parte p de E_2 , que partilhe uma aresta com uma parte p' de E_1 tal que essa aresta contenha a etiqueta *projecção permitida* relativamente a p e cujo comprimento seja superior a d_0 . Se não existe tal parte em E_2 , terminar processo com a mensagem “A Árvore de Suporte Sustentado Não Existe”;
4. Aumentar a árvore colocando p como *filho* de p' e deslocando p para E_1 ;
5. Se E_2 estiver vazio, retornar árvore e parar. Se não, voltar a (3).

Convém referir que existe liberdade na escolha da parte p , no passo (3) do algoritmo e que se tal escolha for aleatória, o algoritmo corre em $O(n)$. No entanto se se alterar o critério de escolha de modo a que a parte escolhida p seja a que possuir maior aresta, o tempo de execução aumenta para $O(n \log n)$ pois o espaço E_2 será implementado por um *heap*. De qualquer forma, a existir uma solução, este algoritmo é sempre capaz de a encontrar.

Assumindo agora que uma árvore é gerada neste último algoritmo e considerando que uma parte é uma *folha* da árvore gerada e que é cortada segundo uma ordem tal que a aresta correspondente ao arco da árvore é a última a ser cortada, têm-se automaticamente garantidas as restrições 1, 2 e 4, do conjunto de restrições acima explicitados. Assim, e seguindo uma heurística para a determinação de um percurso de corte que satisfaça as cinco restrições, foi implementado o seguinte algoritmo (3) em pseudocódigo de alto nível:

1. Partição das regiões de desperdício em sub-regiões *quase convexas*, descrito em (1);

2. Gerar uma árvore de suporte sustentado através de (2). Se não for possível gerar *árvore*, parar procedimento com mensagem de erro: “A Árvore de Suporte Sustentado Não Pode Ser Gerada”;
3. Inicializar a ferramenta de corte com coordenadas pertencentes à borda da placa e inicializar o percurso de corte vazio;
4. Escolher a *folha* da *árvore* que se encontre mais próxima da posição do fio de cobre e que satisfaça a restrição do *ângulo aberto*. Se não existir tal *folha*, parar procedimento com mensagem de erro: “Percurso de Corte Não Encontrado”;
5. Adicionar ao percurso de corte o caminho de comprimento mínimo da posição do fio de cobre à *folha* encontrada e adicionar o caminho correspondente ao corte total da peça associada a essa *folha*. Atualizar posição do fio de cobre;
6. Eliminar *folha* da *árvore*;
7. Se a *árvore* consistir apenas da sua *raiz*, retornar percurso de corte e parar. Se não, voltar a (4).

No final será possível obter um percurso de corte para o problema dado, excepto se ocorrer uma das seguintes situações: a árvore de suporte sustentado não pode ser gerada, o que exige uma alteração ao nível da entrada dos dados (que passa por obter uma solução alternativa do problema de posicionamento) ou todas as *folhas* têm *ângulos abertos* negativos, o que poderá ser resolvido criando um *árvore* alternativa. No entanto, existe sempre a opção de não se parar o algoritmo caso esta situação ocorra, uma vez que se verifica que, na maior parte dos casos, as peças caem da placa mesmo com um *ângulo aberto* negativo.

Assim, esta abordagem permitiu o desenvolvimento de um conceito de árvore que, pelo seu método de construção, resolve três das cinco restrições impostas, sobrando apenas a questão do *ângulo aberto* (quer para as peças, quer para as porções de desperdício) para resolver no algoritmo final. De qualquer modo, retêm-se algumas das premissas usadas na heurística, nomeadamente a ideia de cortar uma parte de uma só vez, ou seja, iniciando-se o corte de uma parte, só se irá considerar uma parte seguinte quando a actual se encontrar completamente cortada da placa.

2.4 Comentários Finais

Neste capítulo foi estudado o PCR e algumas das suas variantes, com especial destaque para o PCR Dinâmico, enquanto principais referências de modelos teóricos do trabalho a desenvolver. Abordaram-se também, sem grande profundidade e apenas para completar o quadro, outros problemas de rotas em arco. Por último, estudaram-se os PDPC destacando em detalhe duas perspectivas de resolução distintas. Num primeiro caso, uma abordagem “dinâmica” que irá ser a base fundamental de todo o trabalho desenvolvido

nesta dissertação e num segundo caso, uma abordagem diferente (em árvore), com uma componente geométrica fortemente marcada, que serviu não só de ponto de comparação de modo muito geral para o primeiro caso, como também para aprofundar a investigação efectuada no contexto industrial dos PDPC, enriquecendo o *background* adquirido sobre o assunto.

Capítulo 3

Ferramentas Auxiliares

3.1 Introdução

Todas as situações actuais relacionadas com a informática em geral, encontram-se hoje servidas de inúmeras soluções facilmente adquiridas à distância de um *click* (ou de um livro nalguns casos mais raros), fruto do enorme *boost* que este domínio sofreu nas últimas décadas. A elevadíssima concorrência aliada ao crescente objectivo de se obter tudo cada vez *melhor* e cada vez *mais rápido*, num mundo totalmente ligado por cabos, colocou à disposição de cada um uma panóplia bastante diversificada de diferentes maneiras de resolver um determinado problema. Assim, neste âmbito, foram escolhidas e abordadas algumas dessas ferramentas que se julgaram mais adequadas ao problema em mãos.

Neste capítulo descrevem-se as principais ferramentas utilizadas no desenvolvimento deste trabalho, em duas secções distintas: na secção 3.2 abordam-se as linguagens de marcação e como se inserem no âmbito deste trabalho, e na secção 3.3 refere-se a principal linguagem de programação utilizada assim como as suas potencialidades para com a aplicação a desenvolver.

3.2 Linguagens de Marcação

A linguagem de marcação é uma linguagem do mundo informático que consiste na ideia de associar marcas ou etiquetas (*tags*¹) a um determinado documento. Estes documentos são normalmente documentos de texto que ficam mais facilmente acessíveis quando marcados por estas etiquetas, o que associado a uma qualquer aplicação informática capaz de as identificar e reconhecer, torna o documento amplamente mais normalizado e, consequentemente, mais *tratável* em aspectos como a sua manutenção ou a sua portabilidade, entre outros.

¹Apresenta-se a tradução para o inglês por ser o conceito mais difundido entre comunidade, e que, por uma questão de normalização, virá a ser utilizado em futuras referências.

3.2.1 *HyperText Markup Language* (HTML)

Não sendo propriamente um tipo de linguagem de marcação detalhadamente abordado neste trabalho de dissertação, a *Linguagem de Marcação para HiperTexto*² é, naturalmente, uma referência profundamente influente neste domínio. Assim torna-se *quase* obrigatório referir o HTML quando se abordam as linguagens de marcação, quanto mais não seja pela sua íntima relação com as páginas *web*. Foi então neste contexto que se aprofundaram os conhecimentos de HTML, enquanto ferramenta básica e necessária, tendo em vista o desenvolvimento de uma página *web* onde se pudesse publicar o trabalhado efectuado com o devido detalhe e apresentação gráfica.

Sem entrar em grande detalhe, um documento HTML é então um documento de texto que respeita um conjunto de *tags* predefinidas que têm de ser respeitadas para este poder ser interpretado pelos *browsers* tradicionais, tais como o Internet Explorer, o Mozilla Firefox, o Opera, entre outros. Após a inicialização do documento com a *tag* `<html>`, este segue uma estrutura um pouco rígida (por oposição à estrutura *livre* do XML, como se verá em seguida) de cabeçalho e corpo. O cabeçalho é uma secção declarativa delimitada pela *tag* `<head>`, onde se inclui a informação sobre o documento actual, como o título, palavras chave que podem ser utilizadas por motores de busca ou ainda o estilo definido, através das *tags* `<title>`, `<meta>` e `<style>`, respectivamente. O corpo, onde aparece o conteúdo do documento através das mais diversas *tags*, é delimitado pela *tag* `<body>` ou `<frameset>`. Na figura 3.1 podemos ver o aspecto básico de um documento HTML.

3.2.2 *eXtensible Markup Language* (XML)

Como o próprio nome indica, a linguagem de marcação extensível³ apresenta uma incrível mais-valia relativamente ao HTML e a outras linguagens de marcação, pela sua extensibilidade ao nível das marcações. Quer isto dizer que, ao contrário do HTML, não existem *tags* pré-definidas e é possível qualquer utilizador definir um conjunto ilimitado de *tags* que mais se adequem à informação que pretende armazenar no ficheiro XML, ainda que, não sendo qualquer *browser* comum que, deste modo, conseguirá interpretar o referido ficheiro, mas sim um processador XML (constituído por *parser*⁴ e um gestor de entidades). Assim, a maior potencialidade deste tipo de ficheiros passa por permitir a troca de informações entre ambientes incompatíveis.

Um documento XML é considerado *bem-formado* se respeitar todas as regras de sintaxe impostas pela norma e encontra-se dividido em *elementos* (ou nós), onde por sua vez se destacam os *atributos*, que funcionam como as propriedades de um determinada marcação

²Por uma questão de conveniência será utilizada em futuras referências a sigla HTML, do equivalente inglês *HyperText Markup Language*.

³Também aqui, por uma questão de conveniência, será utilizada a sigla XML, do equivalente inglês *eXtensible Markup Language*.

⁴Um *parser* é um analisador da sintaxe de um conjunto de dados de entrada, que segue determinadas regras gramaticais e gera um estrutura de dados, normalmente em árvore, facilmente processável.

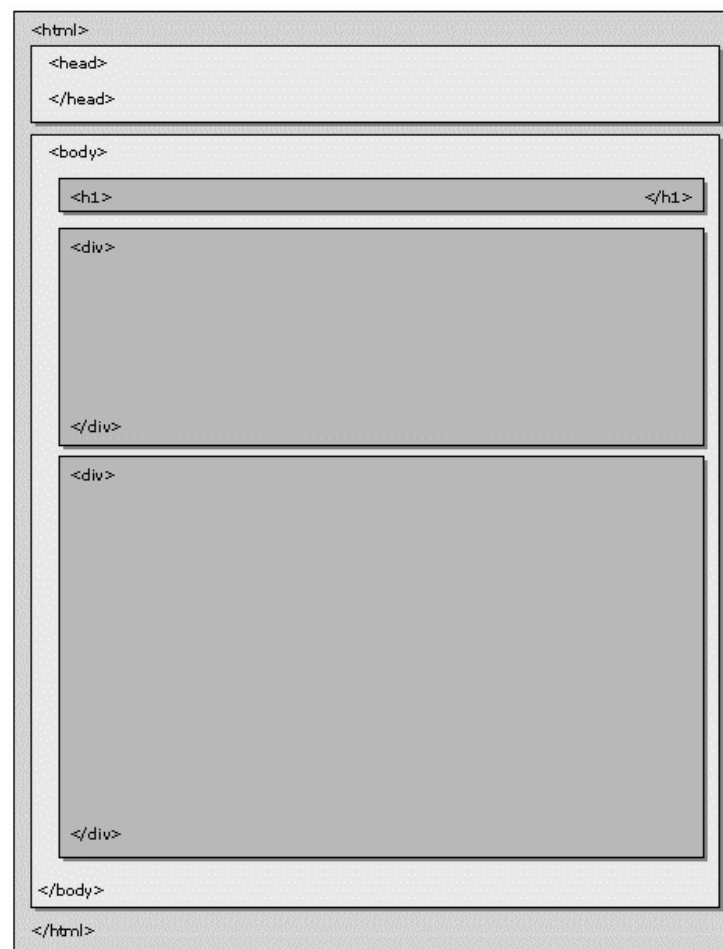


Figura 3.1: Estrutura de um ficheiro HTML.

e os *conteúdos*. Assim, o aspecto genérico de um elemento qualquer de um ficheiro XML toma a seguinte forma:

- `<tag atributo_1="valor" atributo_2="valor">conteúdo</tag>`

Uma das principais regras para a *boa-formação* de um documento XML passa por este possuir apenas um elemento raiz (o elemento mais acima se se considerar uma estrutura em árvore).

O estudo mais detalhado do XML, quer as propriedades intrínsecas deste tipo de ficheiros assim como o seu manuseamento prendeu-se com o facto de se saber *à priori* que a informação relacionada com o problema que se tinha em mãos se encontra toda armazenada em ficheiros deste tipo. Mais especificamente, tratando-se de um PDPC, sabe-se que o seu conjunto de dados de entrada passa pela solução encontrada para um determinado problema de posicionamento. Assim, o ficheiro de saída deste problema de posicionamento contém um conjunto de elementos (criteriosamente definidos num *schema*⁵) relacionados com o contexto geométrico do problema, que constituirá a base das estruturas de dados que irão ser criadas numa primeira fase de resolução do PDPC. Parte de um desses referidos ficheiros pode ser vista em seguida na figura 3.2, com especial chamada de atenção para a referida liberdade de escolha das *tags*, associadas ao problema em questão:

```
<polygons>
  <polygon id="poly1" nVertices="3">
    <lines>
      <segment n="1" x0="0.0" y0="0.0" x1="1.0" y1="1.0" />
      <segment n="2" x0="1.0" y0="1.0" x1="2.0" y1="0.0" />
      <segment n="3" x0="2.0" y0="0.0" x1="0.0" y1="0.0" />
    </lines>
    <xMin>0.0</xMin>
    <xMax>2.0</xMax>
    <yMin>0.0</yMin>
    <yMax>1.0</yMax>
    <perimeter>4.8284</perimeter>
    <area>1</area>
  </polygon>
</polygons>
```

Figura 3.2: Exemplo de parte de um ficheiro XML usado no desenvolvimento deste trabalho (adaptada de [21]).

Por último, sabe-se, pelas razões acima referidas, que existe um sem fim de potencialidade na fusão do XML com o HTML. Tal já acontece sob a forma do XHTML (*eXtensible HyperText Markup Language*) e tende a insurgir-se cada vez mais no mundo *web*, como substituto evoluído do HTML.

⁵Ficheiro que contém determinadas imposições ao nível da criação de uma estrutura para um ficheiro XML.

3.2.2.1 Scalable Vectorial Graphics (SVG)

Como caso específico do XML, o *Scalable Vectorial Graphics* (SVG) surge da associação da linguagem de marcação *livre* com o domínio do desenho vectorial. Antes de se detalhar sobre o SVG, convém perceber que o desenho vectorial surge em complementariedade ao *mapa de bits* e apresenta inúmeras vantagens quando associado à computação gráfica.

Basicamente, o desenho vectorial é uma imagem criada por vectores. É normalmente composta por pontos, linhas rectas, linhas curvas (associadas às curvas de *Bézier*), figuras geométricas simples ou compostas e ainda outros elementos que possam ser descritos por uma equação matemática, o que permite a sua manipulação (normalmente associada ao aumento ou diminuição da imagem ou o tratamento específico de porções isoladas da mesma) sem perda de resolução. Assim, torna-se óbvio constatar que, dada a natureza geométrica do problema, este terá vantagens claras no uso de um suporte gráfico vectorial.

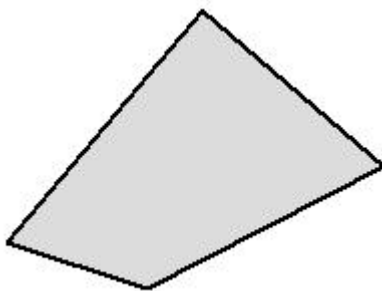
Tal como já referido, o SVG é uma especificação das potencialidades do XML e a ideia é gerar um ficheiro de texto normalizado que produza uma saída gráfica quando devidamente interpretado. Actualmente existem várias aplicações que o fazem, mas a inclusão de *plugins* nos *browsers* tradicionais está a tornar este formato cada vez mais difundido e preferido entre os formatos gráficos existentes, dada também a vantagem do tamanho do ficheiro ser independente da dimensão da imagem. Assim, respeitando um conjunto de *tags* reservadas (como por exemplo `<rect>` para definir rectângulos ou `<circle>` para definir círculos), é possível criar qualquer imagem com o formato *.svg*, que poderá ser visualizada numa determinada aplicação, especificamente o *Mozilla Firefox*.

Na figura 3.3 podemos observar os dois lados de um ficheiro *.svg*: a visualização gráfica (3.3(a)) e a estrutura de texto por detrás (3.3(b)).

Associado a este formato gráfico, encontra-se a possibilidade de, tal como no HTML, embutir *script* entre duas *tags*: `<script>` e `</script>`. Esta particularidade permite adicionar a uma determinada imagem um vastíssimo conjunto de novas situações, que passam, normalmente, pela interacção com um utilizador. Este pode *agir* com a imagem através de eventos pré-definidos (como por exemplo através de um *click* de rato) que irão fazer correr um conjunto de linhas de código de programação. Assim, de modo muito simplificado, foi possível conferir algum *dinamismo* às imagens utilizadas no desenvolver deste trabalho, o que se revelou incrivelmente útil na detecção de erros provocados pelo corte interior de uma peça que não é possível detectar analisando apenas o percurso final determinado.

3.3 Linguagens de Programação

A programação Orientada a Objectos (OO) é um método de implementação segundo o qual os programas e aplicações são organizados sob a forma de *objectos* que interagem uns com os outros. Cada um desses objectos é uma instância duma determinada classe, que se encontra integrada numa hierarquia de classes que define as respectivas relações de *herança*



(a) Imagem em .svg

```
<svg width="100%" height="100%" version="1.1"
xmlns="http://www.w3.org/2000/svg">

<polygon points="220,100 300,210 170,250 123,234"
style="fill:white; stroke:black; stroke-width:1"/>

</svg>
```

(b) Código correspondente

Figura 3.3: Representação e estrutura de um ficheiro em formato .svg.

de umas para as outras. A principal característica destas linguagens de programação é precisamente o uso de *objectos*, que podem ser entendidos como uma abstracção de software com um conjunto de propriedades e procedimentos associados, para representar qualquer coisa real ou virtual.

3.3.1 C++ e a *Standard Template Library*

Dado o vastíssimo leque de opções relacionadas com as linguagens de programação em geral e para não se divergir muito dos tópicos estudados, começa-se por acrescentar que a linguagem usada neste trabalho era uma das restrições, ainda que *soft*, impostas *à priori* para a resolução do problema. Uma vez que já abordada em disciplinas anteriores durante o período de estudos relacionado com o actual MIEEC, são conhecidos os aspectos básicos de funcionamento (com algum detalhe) e respectiva sintaxe da linguagem em questão, pelo que se torna evidente que, e dado o curto espaço de tempo para abordar um problema desta magnitude, seria uma opção coerente com os objectivos do trabalho. Ainda, sabia-se que a aplicação informática existente sobre a qual o trabalho iria ser efectuado foi implementada usando uma linguagem de sintaxe semelhante assim como igualmente *orientada a objectos*. No entanto, uma vez que não se trata de um trabalho desenvolvido especificamente sobre esta ferramenta de trabalho, não se irão detalhar os conteúdos gerais do C++ mas sim qual foi o seu papel específico neste contexto, de modo a não tornar esta secção indevidamente densa.

Sendo uma linguagem poderosa, mas relativamente complexa pelas suas influências do C, o C++ apresenta uma enorme mais valia na forma de uma biblioteca normalizada de componentes reutilizáveis, que é a grande vantagem da *orientação a objectos* ([16]), a *Standard Template Library* (STL).

A STL⁶ foi desenvolvida com o intuito de facilitar a vida aos programadores organizando um conjunto de funcionalidades existentes em classes de uso genérico. Assim, surgem três componentes essenciais desta biblioteca: *contentores* (que, de um modo geral, se podem considerar estruturas de dados normalizadas), *iteradores* e *algoritmos*.

Os contentores, que se encontram por sua vez divididos em duas grandes categorias (contentores de sequência e contentores de associação) são estruturas de dados (ou classes) que implementam diferentes propriedades (relacionadas com o acesso a um determinado elemento, com a inserção ou eliminação de um novo elemento quer no início, quer no final ou noutra posição específica da estrutura, entre outras) e possuem um conjunto comum de funções que executam operações genéricas, tais como retornar o tamanho, limpar todas posições, ou verificar se a estrutura se encontra vazia. De todos os contentores existentes nesta biblioteca, destacam-se os *vector* pelo seus rápidos acessos e rápidas inserções na última posição da estrutura, as *list* pelo seus tempos de execução constantes a inserir e a eliminar elementos da estrutura e, por último, as *deque*, que apresentam vantagens similares aos *vector* mas também ao nível da inserção de elementos no início da estrutura, todos contentores de sequência. Dentro dos contentores de associação, destacam-se os *set*, os *multiset*, os *map* e os *multimap*, que não foram de todo utilizados no desenvolvimento do trabalho, dadas as suas características de ordenação *não sequencial* de dados, não efectivas para o problema em questão.

Os iteradores apresentam enormes similaridades com os apontadores comuns e funcionam precisamente para percorrer e *apontar* para os elementos que os contentores, acima referidos, guardam. Constituem a característica fundamental que confere generalidade à STL.

Os algoritmos são, naturalmente, funções pré-definidas que, como já referido, podem ser utilizadas indistintamente nos diferentes contentores que implementam iteradores. Assim, funções de inserção, eliminação, procura ou ainda de ordenação podem ser facilmente usadas numa determinada aplicação porque se encontram já implementadas na STL.

Por último salienta-se ainda uma importante característica da STL relacionado com o facto de existir uma abstracção do tipo de dados concreto utilizado, derivado do uso de *templates*.

Relativamente ao uso específico do C++ para a resolução do PDPC, distinguem-se duas aplicações distintas mas complementares no sentido da estrutura lógica de funcionamento do programa. Primeiro, um *parser* de ficheiros XML, o *Expat XML parser* de distribuição livre, que, com algumas pequenas alterações ao código original em C, nomeadamente inserindo componentes da STL, irá organizar a informação necessária do ficheiro

⁶Por uma questão de conveniência será utilizado o acrónimo de *Standard Template Library*.

XML de entrada nos referidos contentores e outras estruturas de dados. Em seguida, a aplicação principal de determinação de percursos de corte, que trabalhará sobre as estruturas implementadas sobre STL e, juntamente com um conjunto variado de funções e classes, irá implementar uma heurística para a resolução do problema.

3.3.2 PHP

Esta subsecção foi incluída porque considerou-se importante fazer uma pequena referência à linguagem de programação usada no que respeita a pequenas aplicações *web*.

O PHP, do inglês *PHP: Hypertext Preprocessor*, é normalmente utilizado no desenvolvimento de páginas *web* dinâmicas pela sua fácil inclusão no HTML. É amplamente aceite na maior parte dos sistemas operativos e é um tipo de *script* que corre do lado do servidor (por oposição aos *scripts* que correm no lado do cliente), com grande utilidade nos contextos de formulários, como é exemplo disso a página inicial de *webmail* da Faculdade de Engenharia da Universidade do Porto. Apresenta algumas diferenças significativas para o C++, nomeadamente ao nível da definição de variáveis, que por falta de enquadramento no âmbito deste trabalho não serão aqui abordadas.

No seguimento da ideia de oferecer algum dinamismo à página *web* desenvolvida em paralelo com este trabalho, nomeadamente a intenção de integrar quer a aplicação de *parsing* dos XMLs de saída do problema de posicionamento (com consequente apresentação das soluções a um utilizador) quer a aplicação, em C++, que irá correr sobre essas soluções de modo a determinar um percurso de corte, foram analisadas algumas noções básicas desta linguagem de programação para que no final fosse possível criar uma interface *web* que permitisse efectuar uma apresentação *online* de todo o trabalho realizado no âmbito desta dissertação.

3.4 Comentários Finais

Neste capítulo foram introduzidas as linguagens de programação e de marcação como principais ferramentas a utilizar no desenvolvimento deste trabalho. Referiram-se as potencialidades do XML e do SVG enquanto formato de representação gráfica adequado, assim como as potencialidades da STL ao nível da implementação. Estudaram-se ainda as ferramentas necessárias para o desenvolvimento da página *web* imposta, de modo a que esta pudesse não só conter toda a informação actualizada referente ao trabalho, como também, no final, uma interface dinâmica para apresentação dos resultados obtidos.

Capítulo 4

Algoritmo de Determinação de Percursos de Corte

4.1 Introdução

Dadas as referências sobre o contexto do problema tratado nesta dissertação e as ferramentas para lidar com ele, tornou-se a certa altura necessário efectuar uma abordagem *inteligente* sobre a melhor maneira de as aplicar. O trabalho realizado nesse sentido passou por conhecer, compreender e interpretar uma aplicação informática existente desenvolvida com o objectivo de resolver um PDPC e trabalhá-la em diversos sentidos: modificar a sua linguagem, actualizar detalhes de implementação, corrigir erros de funcionamento e propor novas abordagens heurísticas para a resolução do problema.

Este capítulo refere, de um modo muito geral, todo o longo processo de implementação, desde o ficheiro de entrada com a colocação das peças até ao percurso de corte que sai no fim, abordando todos os detalhes que o compõem. Encontra-se dividido de forma sequencial relativamente à cronologia da ordem de trabalho. Na secção 4.2 estudam-se todos os detalhes relacionados com o *input* da aplicação principal sob a forma de um ficheiro XML específico. Na secção 4.3 refere-se a base heurística que suporta o algoritmo principal, a heurística “Vértice Mais Acima”, e analisa-se, de alto nível, a sua perspectiva de funcionamento. Em seguida referem-se situações relevantes que surgiram no contexto da recodificação e que se acharam pertinentes analisar mais detalhadamente, abordando questões específicas da implementação utilizada, nomeadamente ao nível das estruturas de dados. Na secção 4.4 detalham-se os problemas encontrados, como a maior contribuição deste trabalho. Dado o largo contexto do PDPC, entende-se que existirá sempre um mundo de possíveis questões por resolver ou restrições complexas por implementar. Ainda, da análise séria desses problemas encontrados, surgem propostas de correcção (algumas implementadas) e uma proposta para um novo critério de decisão, o critério “Peça Mais Próxima”. Este critério sustenta uma abordagem heurística, explicada na secção 4.5. Por fim, tecem-se comentários diversos sobre as questões abordadas.

4.2 O ficheiro de entrada

A aplicação original sobre a qual se desenvolveu este trabalho, que será seguidamente abordada, foi terminada por volta do ano 2002, altura em que o *output* de dados dos problema de posicionamento não era feito para um ficheiro XML, com um *schema* criteriosamente definido, tal como se faz hoje em dia. Existia um conjunto de diversos ficheiros de texto onde a informação se encontrava dispersa e sujeita a grande ambiguidade. Assim, seguindo o referido *schema*, é criado um *template* em XML para representar a solução destes problemas de *nesting*¹ num único ficheiro. Fica então automaticamente definido que o *input* dos PDPCs será também normalizado por este mesmo XML.

Uma vez que os problemas de posicionamento de formas irregulares tratam com uma geometria computacional incrivelmente densa e detalhada, é natural esperar que exista um conjunto considerável de informação que não seja de todo útil para a abordagem do PDPC. No entanto, sabe-se que o conjunto de peças colocadas numa determinada placa, completamente pronto a ser analisado para a determinação do seu percurso de corte, não se encontra explicitamente descrito no XML e carece de uma análise lógica da informação aí existente. Assim, para que se possa perceber um pouco melhor o pré-processamento das estruturas de dados, o XML contém a informação que se pode ver representada na figura 4.1, numa estrutura em árvore, onde os diversos nós são elementos do referido ficheiro².

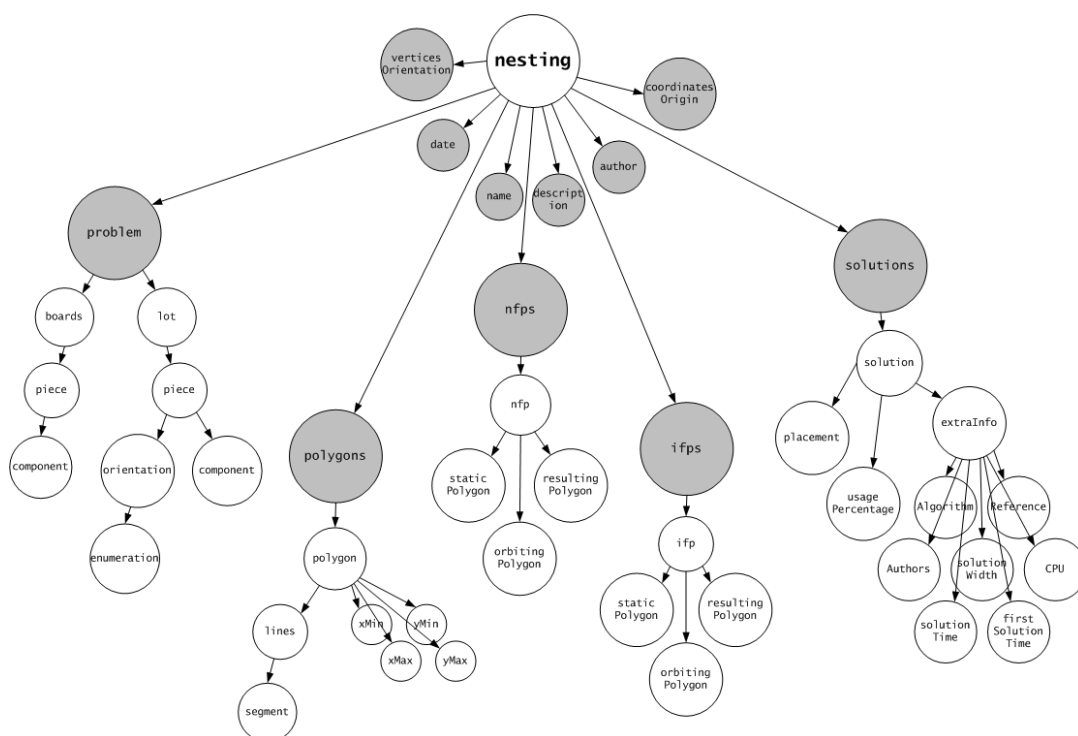


Figura 4.1: Estrutura em árvore do ficheiro XML usado como input do PDPC.

¹Definição comum na literatura existente para os problemas de posicionamento.

²Por uma questão de fidelidade manteve-se a notação original em inglês.

Tendo em consideração a estrutura representada na figura 4.1, consideram-se relevantes para o problema específico da determinação de um percurso de corte, as *boards*, com informação relativa aos componentes da placa, o *lot*, com informação relativa aos componentes de cada peça, os *polygons*, onde se encontra a descrição de cada figura geométrica (genericamente definida como *polígono*) utilizada no *nesting*, e, por último, os elementos de *placement*. Estes contêm a seguinte informação sobre as peças que foram colocadas na placa: identificação da peça e da placa onde esta se encontra, ângulo de rotação e espelhamento da peça, e ainda as coordenadas relativas da colocação do primeiro vértice. Importa referir que, como o algoritmo de determinação de percursos de corte só trata uma placa de cada vez, só é então considerada uma placa por cada solução de *nesting* no momento da interpretação da informação do XML, a seguir explicado.

O *parsing*, ou interpretação automática do ficheiro XML é então efectuado de modo a que, sequencialmente, se efectuem as seguintes etapas até à formação de uma estrutura de dados que contenha um *input* adequado para a determinação de um percurso de corte. Assim, dada uma linha de *placement* do referido ficheiro XML:

1. Identifica-se a peça em questão que foi colocada;
2. Identificam-se os componentes (polígonos) que constituem essa peça;
3. Identificam-se os segmentos correspondentes a cada um desses componentes e respectivas coordenadas absolutas (iniciais e finais);
4. Executam-se, sequencialmente, as seguintes operações sobre a totalidade das coordenadas pertencentes ao polígono em causa: rotação e translação (pelo ângulo de rotação e pelas coordenadas relativas definidas no referido *placement*), por esta ordem, respectivamente.

Após percorrer todas as linhas de *placement* de cada solução, é então possível criar um conjunto de dados de entrada adequados à aplicação existente, tal como se verá mais adiante no anexo A, nas figuras que representam as instâncias utilizadas para os testes realizados.

4.3 A Heurística VMA

Referida no capítulo 2 deste documento, a heurística construtiva que implementa o critério VMA³ serviu de ponto de partida para o trabalho a desenvolver nesta dissertação.

Implementada numa aplicação em Java⁴, esta heurística é o reflexo da tendência da prioridade atribuída aos cortes obrigatórios serem efectuados numa região da placa mais afastada do suporte. Como já foi referido, a construção do percurso de corte faz-se, simplificada e do seguinte modo: dado o *vértice actual*, a escolha do *vértice seguinte* recai

³Vértice Mais Acima.

⁴Conhecida linguagem de programação orientada a objectos.

sobre o *vértice mais acima*⁵ de entre os possíveis candidatos, que satisfaçam determinados requisitos, tal como especificado em 2.3.1.1.

Dado um padrão de corte sobre a placa de diamante, este irá gerar um grafo constituído pelos vértices de todas as peças colocadas. O conjunto de arestas deste grafo é então formado pelo subconjunto de arestas obrigatórias, correspondente às arestas que ligam os vértices da mesma peça, e pelo subconjunto de arestas facultativas, ou pontes, que são todas as outras ligações entre vértices, naturalmente visíveis, de peças diferentes. Sobre este grafo determina-se, em cada iteração, qual o vértice seguinte a introduzir no percurso de corte dado um vértice actual, segundo o critério VMA. A informação de cada peça é mantida em memória.

A inclusão de um vértice seleccionado (possível candidato a vértice seguinte) no percurso de corte, é precedida pela verificação da admissibilidade desse mesmo vértice, relativamente à formação de algum ciclo. Assim, caso a inclusão de um vértice seja válida relativamente à formação de um ciclo admissível, que corresponde às situações de queda de uma peça ou queda de uma porção de desperdício, ou não exista sequer formação de ciclo associada a essa inclusão, o vértice é realmente introduzido no percurso de corte, seguindo-se a correspondente actualização das estruturas de dados. Se, por outro lado, a inclusão desse vértice origina um ciclo não admissível (isto é, queda de uma porção de placa com peças não completamente cortadas), ele não é introduzido no percurso de corte nessa iteração. O percurso é então redireccionado através do mecanismo de *backtracking*, atrás explicado na secção 2.3.1.

Sempre que uma peça é completamente cortada da placa, a sua queda irá possibilitar a utilização da área que ocupava anteriormente. Tal modificação sobre a placa conduz a uma alteração na estrutura do grafo, através da actualização das visibilidades. A partir desse momento, a determinação do percurso de corte faz-se sobre o grafo modificado. Ao longo de todo o problema, o grafo sofrerá tantas modificações, quantas as peças posicionadas na placa menos uma (é evidente que após a queda da última peça não se irá proceder a mais actualizações de visibilidades).

Ainda, convém salientar que o contexto particular de um PDPC sobre uma placa suspensa, como é o caso, impossibilita a ocorrência de situações onde o corte de uma determinada aresta (dupla) provoque a queda simultânea de duas peças encostadas. Esta situação implicaria, obrigatoriamente, a formação prévia de um ciclo não admissível contendo as duas peças. Assim, na abordagem efectuada neste trabalho, só pode ocorrer a queda sequencial de peças, ou seja, uma peça de cada vez.

No fluxograma da figura 4.2 apresenta-se uma perspectiva de alto nível relativa à heurística acima descrita, dadas as restrições impostas para o PDPC referido.

⁵Atingíveis por arestas não cortadas.

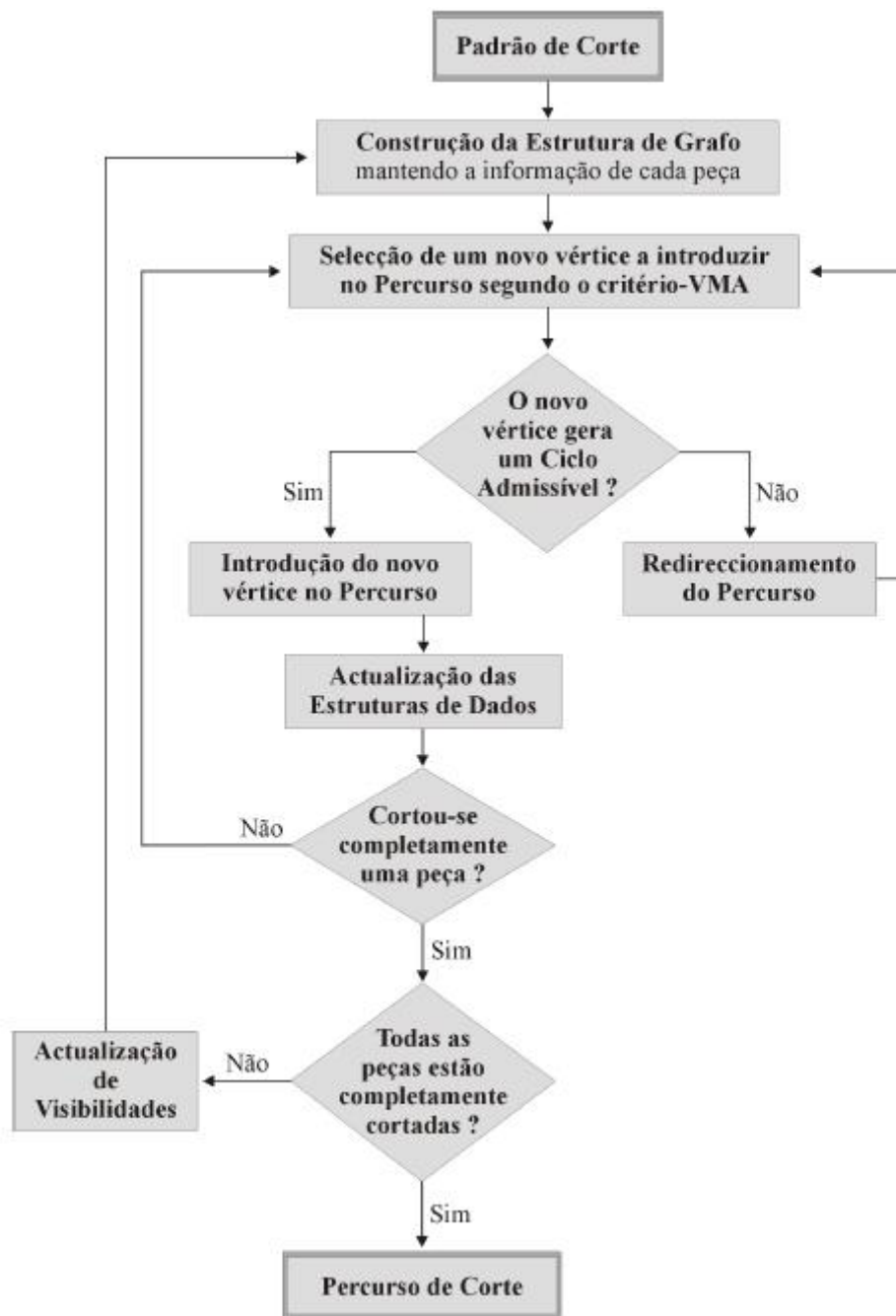


Figura 4.2: Fluxograma da heurística construtiva VMA (adaptado de [22]).

4.3.1 Revisões de Implementação

A opção pela linguagem C++ deveu-se às suas características que facilitam o desenvolvimento de um algoritmo de tamanha complexidade, nomeadamente ao nível da gestão dinâmica de memória (como será revisto mais à frente) e das potencialidades da STL (como referido em 3.3.1). No entanto, nenhuma análise foi efectuada para se apurar se de facto é a escolha mais adequada.

O processo de recodificação da aplicação existente em Java para C++ foi demorado e exaustivo na medida em que o programa original apresenta uma estrutura de implementação de elevado grau de complexidade. Ainda que não se pretendesse a criação de uma aplicação de raiz, tornava-se fundamental conhecer todos os detalhes de funcionamento relacionados não só com a heurística principal, como também com as funções, classes e estruturas de dados (sob a forma de variáveis globais) que foram criadas. Pretendia-se saber, no fim da recodificação, o significado de todas as variáveis e a lógica por detrás de todas as funções auxiliares à heurística utilizada.

Dada a complexidade da aplicação, o projecto passou pela planificação de duas fases de trabalho complementares: em primeiro, desenvolver uma aplicação em C++ que iria seguir uma estrutura lógica exactamente igual à original, com a respectiva adaptação da sintaxe e de outros pequenos detalhes de implementação e, num segundo caso, adaptar esta versão em C++ às suas reais potencialidades, como se tivesse sido originalmente *desenhada* nesta linguagem. No primeiro caso, a ideia foi desenvolver uma versão da aplicação existente numa linguagem diferente, tal e qual como se tivesse sido criada pelo programador original transportando conseqüentemente, e de modo um pouco *virtual*, todas as suas opções e linhas de raciocínio para esta última versão. Usando uma linguagem de baixo nível para melhor definir as duas fases do trabalho, pode considerar-se que a primeira aplicação é uma aplicação de *Java em C++* e a segunda aplicação, uma aplicação de *C++ em C++*.

Não só por ter sido o primeiro contacto com a aplicação existente, a complexidade desta primeira fase de trabalho reveste-se ainda de maior importância quando se toma em consideração as diferenças semânticas que ambas as linguagens apresentam. Não tendo sido abordado o estudo da linguagem Java por uma questão de orientação e gestão do tempo disponível, quer-se, no entanto, salientar dois aspectos relativos a estas diferenças semânticas, que se revelaram preponderantes na forma como foi efectuada a referida recodificação e que irão ser certamente úteis num futuro próximo: passagem por valor vs. passagem por referência e a gestão automática de memória.

Nas declarações em C++, a declaração do nome da classe declara um objecto dessa classe como um valor, por oposição ao Java, cuja declaração do nome da classe declara uma *referência*⁶ para um objecto dessa classe. Assim, os objectos em Java não são *nunca*

⁶Uma referência significa uma variável que possui o valor existente num determinado endereço de memória de outra variável e não uma cópia do seu valor. Assim, quando se altera o valor desta última variável, o valor da primeira é também alterado.

valores. A maneira de contornar esta situação passa por utilizar apontadores em C++, que, pela sua complexidade de manuseamento e inúmeras vezes associados a complicados problemas de programação, não existem em Java. Sejam a e b duas declarações de objectos de uma determinada classe genérica (objectos de valor no caso de C++ e referências para objectos no caso do Java) e x um membro-dado (um valor inteiro neste caso) dessa mesma classe. Deste modo, teve-se em atenção o paralelismo das seguintes situações:

- Quando se declara, em C++, que $b = a$, está a copiar-se o conteúdo de a para b . A operação equivalente em Java é $b = a.clone()$. Deste modo pode modificar-se a (através da sintaxe $a.x = 10$, por exemplo) sem se alterar b .
- Quando se declara, em Java, que $b = a$, está a passar-se para b uma referência de a . A operação equivalente em C++ é $*b = a$. Deste modo pode modificar-se a (através da sintaxe $a->x = 10$ em C++ e através da sintaxe $a.x = 10$ em Java), alterando logicamente b .

Relativamente à gestão automática de memória, sem entrar em grande detalhe, pretende-se apenas referir que ambas as linguagens abordam maneiras diferentes de o fazer, com especial referência para o *colector de lixo*⁷ ser obrigatório em Java e voluntário em C++, ainda que praticamente nunca utilizado. Este aspecto, não sendo facilmente exemplificável sem se aprofundar devidamente o assunto, poderá estar associado a relevantes variações dos tempos de execução, mediante o *peso* da aplicação, aspecto intimamente relacionado com a quantidade de memória que está ser utilizada.

O primeiro grande passo dado na recodificação da aplicação foi no sentido de se implementarem as classes existentes com o principal objectivo da familiarização com os tipos de objectos que se iriam trabalhar. A referida complexidade do programa passava pela existência de inúmeras classes, das quais se seleccionaram aquelas relacionadas com a estrutura lógica do programa. O outro conjunto de classes encontra-se associado a questões específicas do Java e a questões relativas com a interface gráfica que a aplicação original possui e não se viu necessidade de as recodificar. As classes assim implementadas encontram-se intimamente ligadas à geometria do problema e foram deste modo desenvolvidas de modo a conferir uma sintaxe facilmente comparável com a realidade envolvente.

Seguindo uma linha de herança lógica, começa-se por definir a classe *vértice* como o elemento geométrico básico. O vértice aqui implementado é constituído por dois *doubles*⁸, que são os valores das suas coordenadas (naturalmente x e y), por três valores inteiros relacionados com o número de arestas incidentes no vértice em questão, nomeadamente o número de arestas incidentes não cortadas, o número de arestas incidentes cortadas e o número de arestas incidentes cortadas activas (sendo este último conceito de *activo* associado a uma peça totalmente cortada) e ainda um booleano que permite saber se o

⁷Tradução do inglês *garbage collection*, como processo de gestão automática de memória.

⁸Tipo de variável básica do C++ que representa um número em vírgula flutuante com 8 *bytes*.

vértice em causa pertence à estrutura original do polígono ou, se pelo contrário, surgiu após o *seccionamento* de uma aresta (ver secção 2.3.1).

A partir desta classe, encontram-se definidas duas ramificações de heranças, como representado na figura 4.3. Ainda que sem existir realmente alguma separação estrutural entre as classes, entende-se ser mais lógico organizar o raciocínio baseado nesta divisão. Sem aprofundar grandes detalhes de implementação, vai-se efectuar uma breve explicação das classes criadas, por se considerar que foi um ponto de estudo importante na compreensão geral do algoritmo existente. Assim, distinguem-se as classes associadas à perspectiva geométrica de alto nível, a saber:

- Classe *rectângulo*;
- Classe *aresta*;
- Classe *peça*.

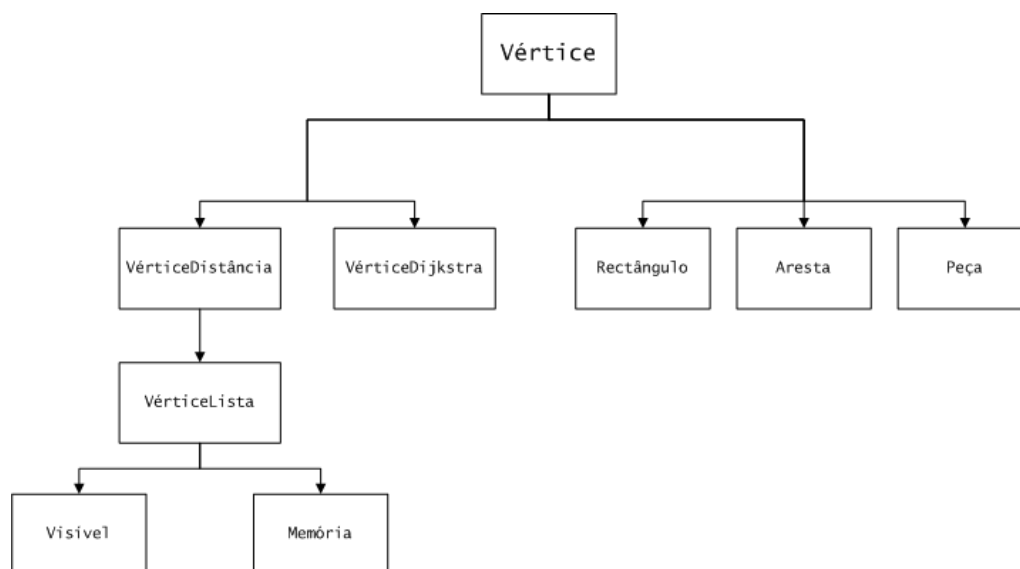


Figura 4.3: Diagrama de herança das classes definidas.

A classe *rectângulo* desempenha o seu maior papel aquando do cálculo das *envolventes rectangulares*. Tal como já referido, é um procedimento de elevado rácio *utilidade / peso computacional*, daí que se tenha considerado importante a sua criação. É uma classe que, pelas razões já apresentadas, implementa simplesmente dois vértices, nomeadamente o vértice superior esquerdo e o vértice inferior direito de um rectângulo.

A classe *aresta* foi naturalmente desenvolvida a pensar na estrutura de dados que contém todas as arestas do problema inicial. Dada uma aresta genérica, esta classe implementa o seu vértice inicial e o seu vértice final, o número da peça à qual pertence e um número único atribuído no momento da sua criação no caso de ser uma aresta *dupla* e ainda as duas características distintas possíveis que ela pode assumir: cortada ou não cortada e

activa ou não activa, representadas por duas variáveis booleanas, respectivamente. Dado o contexto dos PRAs que envolve tão assumidamente este problema, torna-se evidente notar que esta classe representa uma das implementações fundamentais da aplicação.

A classe *peça* surge no topo da importância crescente das classes que têm vindo a ser descritas. Ainda que modelado através de um grafo, tal como descrito no capítulo 2, o conjunto de dados de entrada deste PDPC é um conjunto de peças ordenadamente dispostas dentro de uma peça maior (a placa). Assim, de modo a poder fazer-se a transformação da geometria inicial em nós e arcos e ainda ser possível acompanhar o estado evolutivo do processo de corte para a actualização de visibilidades, torna-se necessário criar uma classe que consiga generalizar as referidas necessidades. A classe *peça* implementa um vector de vértices, associado aos pontos originais da peça e aos resultantes de eventuais contactos, o número total de vértices que possui, um número único atribuído a cada peça no momento da sua criação e a sua identificação original.

A segunda ramificação de classes *herdeiras* da classe *vértice* prende-se com as diversas utilizações dos vértices ao longo de toda a implementação. Ainda que não transparecendo boas práticas de programação, considera-se que o programador original abordou a questão numa perspectiva de melhor organizar as suas ideias e facilitar o seu trabalho de implementação. No entanto, dada a complexidade da aplicação para o tempo disponível, acabou por se verificar proveitoso o modo como as classes foram criadas, na medida em que existem deste modo várias classes “pequenas” menos complexas de entender do que classes maiores e mais completas ao nível da estruturas, naturalmente mais complexas de se assimilar dado o tempo disponível. Assim, directamente relacionadas com as variadas utilizações dos vértices, definem-se as seguintes classes:

- Classe *vérticeDistância*;
- Classe *vérticeDijkstra*;
- Classe *vérticeLista*;
- Classe *visível*;
- Classe *memória*.

A classe *vérticeDistância* apresenta uma forte utilidade em toda a aplicação. A diferença lógica entre a classe *vértice* e esta, reside no facto de se associar à classe *vértice* um objecto (com um par de coordenadas e os outros membros-dado) que existe em qualquer polígono fora do contexto do PDPC. Transportando esse vértice para dentro do contexto deste problema específico de minimização de percursos de corte, fica claro notar que cada vértice irá estar associado a uma distância para outro vértice.

Assim, esta classe implementa, naturalmente, um *vértice* e um valor de distância a ele associado, assim como um conjunto de variáveis que permitem cobrir a multiplicidade das situações em que o *vértice* é utilizado: uma variável que regista o número da peça que caiu

quando esse vértice é introduzido no percurso, uma variável que identifica a forma (corte de aresta, aresta já cortada ou ponte) como se atingiu esse vértice, uma variável que determina a admissibilidade de um ciclo formado por esse vértice e uma variável que explicita se esse vértice foi ponto de chegada ou de partida para o algoritmo de Dijkstra implementado. Dado o *upgrade* que foi efectuado ao vértice, por adição de mais informação relevante para o problema, justifica-se que as restantes classes que utilizam vértices, implementem *vérticesDistância*, tal como se pode observar na figura 4.3.

A classe *vérticeDijkstra* é uma excepção à afirmação acima referida porque, na realidade, é uma classe cuja implementação serve o propósito muito específico de resolver o algoritmo de Dijkstra. Assim, é uma classe de pouco uso genérico que implementa um *vértice*, um valor de distância mínima a esse vértice em cada instante (vulgarmente definido como *etiqueta*), um estado dessa etiqueta e ainda a peça a que esse vértice pertence.

A classe *vérticeLista* surge como a principal herdeira da classe *vérticeDistância* e é uma classe criada tendo em vista as duas grandes estruturas de dados suportadas pela aplicação: estrutura de visibilidades e a estrutura de memória, ambas referidas em 2.3.1. Assim, esta classe implementa um vértice, uma sublista de vértices (que em termos de implementação serão *vérticesDistância*) *visíveis* a partir desse vértice, a respectiva dimensão dessa sublista e uma variável que permite saber o índice da posição de um determinado objecto desta classe, quando inserido num vector de objectos da mesma classe.

Com a implementação das classes *vérticeLista* e *vérticeDistância* é possível criar uma estrutura bidimensional de vértices e respectivas distâncias entre eles, da seguinte forma: um objecto do tipo *vérticeLista* implementa, como se viu em cima, uma sublista de *vérticesDistância* pelo que, se se tiver uma lista de *vérticesLista*, irá obter-se uma matriz de vértices como a que se encontra representada na figura 4.4.

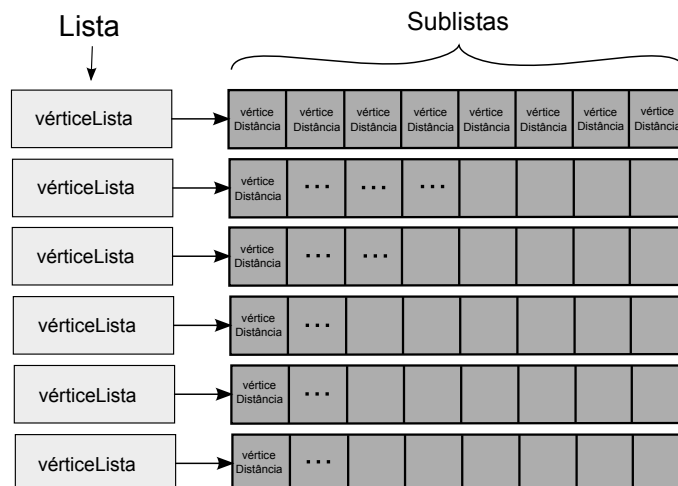


Figura 4.4: Representação de alto nível da matriz criada pela interacção das duas classes.

Esta matriz é constituída por uma lista de vértices de saída (vértices pertencentes a uma determinada peça, a partir dos quais é possível atingir outro vértice qualquer,

inclusive da mesma peça, respeitando o conceito de *invisibilidade*) e por uma sublista de vértices de chegada (conjunto de vértices atingíveis dado um determinado vértice de saída) para cada vértice de saída existente. Tendo em consideração os membros-dado das classes que implementam esta estrutura, faz sentido que esta seja definida como *matriz das distâncias*.

As duas restantes classes foram implementadas de modo a poderem tirar partido da matriz de vértices criada da interacção das duas classes superiores na estrutura de heranças. Assim, quer a classe *visível*, quer a classe *memória* implementam uma lista de *vérticesLista* e a respectiva dimensão dessa lista. Em termos específicos, a classe *visível* implementa uma variável binária que toma o valor 1 quando existe visibilidade entre os objectos da classe e 0 caso contrário. Por outro lado, a classe *memória* implementa uma variável que distingue objectos deste tipo e outra que permite saber qual o índice da posição de um determinado *vérticeLista* no seu respectivo vector.

Ainda que já abordadas em 2.3.1 mas dadas as questões de implementação acima descritas, torna-se relevante efectuar uma perspectiva geral sobre as duas grandes estruturas de dados sobre as quais assenta o bom funcionamento da aplicação:

- A *matriz das visibilidades* é uma matriz tetradimensional que segue a seguinte lógica: é criada uma matriz quadrada bidimensional, de dimensão igual ao número de peças colocadas na placa, dado o posicionamento inicial. Esta matriz é de mais alto nível e é formada por objectos do tipo *visível* onde se vai verificar, através de um valor binário, a visibilidade entre duas peças, isto é, se a peça da linha i vê a peça da coluna j da matriz. Caso esse valor seja 1 (sinónimo de visibilidade), é então criada nessa posição da matriz, uma matriz de distâncias (de baixo nível), cuja estrutura se encontra representada na figura 4.5. Obviamente que nas situações em que não existe visibilidade de i para j , não existirá nada nessa posição da matriz.

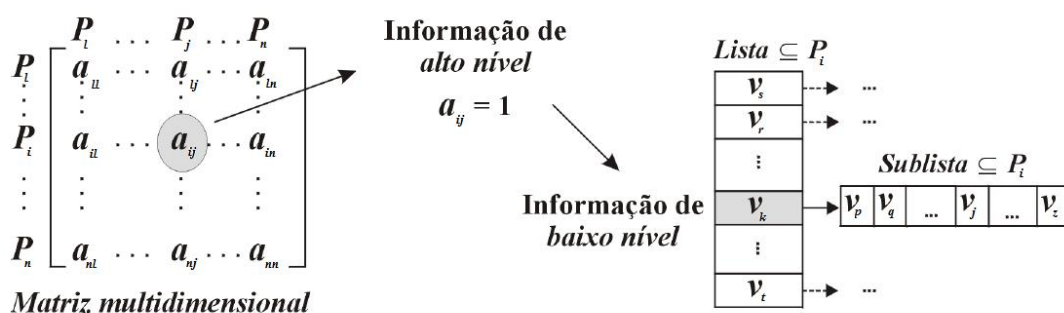


Figura 4.5: Arquitectura da *matriz das visibilidades* (adaptado de [22]).

- A estrutura que simula a memória da aplicação apresenta, relativamente à estrutura anterior, uma lógica de implementação menos complexa. Assim, entende-se a matriz da memória como uma matriz que, de alto nível, implementa um vector de objectos do tipo memória. Em cada uma destas posições irá estar associada uma

lista de vértices, mais correctamente uma lista de *vérticesLista*, cujo topo da lista é encabeçado pelo vértice que originou a chamada da estrutura de memória, isto é, o vértice que originou um ciclo admissível⁹. Para cada um desses vértices irá estar associada a referida sublista de vértices, mais correctamente de *vérticesDistância*, que neste caso, e sem grande detalhe, irão representar os vértices onde existe uma alternativa para “fuga” à inadmissibilidade de um ciclo. Assim, a estrutura de memória implementa uma matriz a três dimensões, tal como se pode observar na figura 4.6.

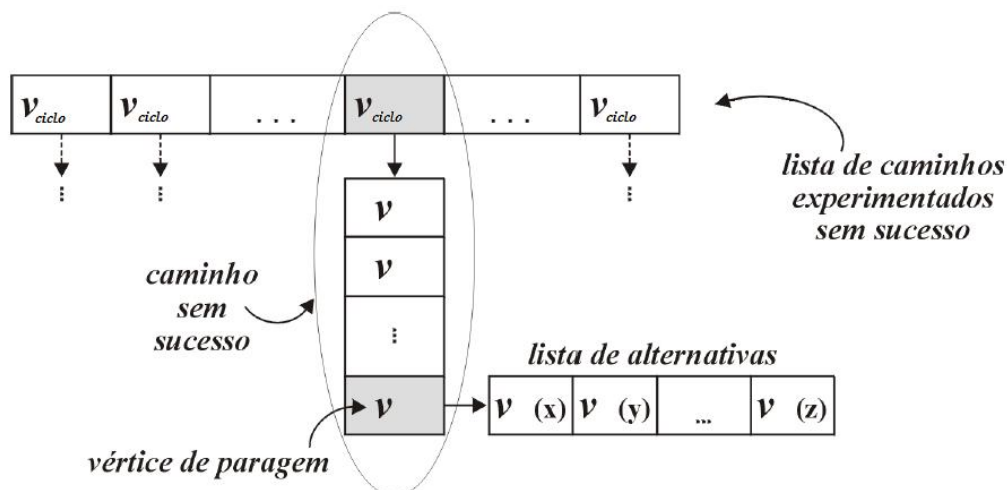


Figura 4.6: Arquitectura da estrutura de memória (adaptado de [22]).

Ainda que de elevado grau de complexidade, a representação adoptada para a *matriz das visibilidades* apresenta fundamentalmente dois argumentos a seu favor no que diz respeito à construção, óbvia, de uma matriz completa de todos os vértices. Primeiro, a economia de informação, pois apenas se registam os vértices envolvidos na visibilidade, e segundo, a perspectiva de alto nível pois é possível identificar todas as peças visíveis, a partir de um dada peça.

4.3.2 Questões de Eficiência

Esta subsecção procura referir que, após concluído o processo de recodificação, foram encontradas algumas questões de eficiência relativas ao desempenho da nova aplicação. Para todos os casos testados, a versão recodificada apresenta tempos de execução superiores aos da aplicação original, situação claramente agravada quando se tratam instâncias com elevado número de vértices. Refira-se que não foi encontrada, em tempo útil, uma resposta suficientemente aceitável para tal facto.

Dando prioridade a outras questões que foram surgindo, nomeadamente às que serão descritas na secção seguinte, considerou-se o desempenho da nova aplicação como uma questão secundária, face à generalidade do problema. Na origem poderão estar diversos

⁹Anteriormente definido como *vértice ciclo*.

motivos, muito provavelmente relacionados com a gestão automática da memória, referida na secção 4.3.1, existente no Java e que não foi cuidada no C++. Deste modo, pretende-se referir que não foram efectuados testes de competitividade entre as duas versões da aplicação, tendo sido dado ênfase aos problemas de funcionamento que a aplicação original apresentou quando foi submetida a instâncias mais elaboradas.

4.4 Problemas encontrados

Ao longo de todo o processo de recodificação, assim como posteriormente, quando se efectuaram testes computacionais, foram surgindo inúmeros detalhes impeditivos de um progresso evolutivo constante. Ainda que alguns desses detalhes fossem devidos a pequenas questões de implementação (algumas das quais referidas em 4.3.1), foram encontrados dois problemas de implementação na aplicação original que resultavam em erros de funcionamento. Assim, corrigidos os detalhes relacionados com a diferença de linguagem de programação utilizada e contornados (dentro do possível) os detalhes relacionados com o problema da representação de números em vírgula flutuante que surgirem em algumas instâncias, referem-se duas situações em que a estrutura da aplicação original não se encontra devidamente implementada e, conseqüentemente, não é capaz de determinar o percurso de corte respectivo. É sugerida e implementada ainda uma correcção possível para uma das situações.

A primeira situação de erro que se identificou, encontra-se relacionada com a questão das visibilidades numa situação muito específica do posicionamento de duas peças, devido ao seu elevado grau de encaixe. A situação que se analisa em seguida tem como base a figura 4.7, onde se chama a atenção para os dois vértices unidos por traço interrompido.

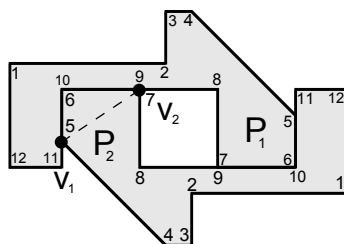


Figura 4.7: Problema de visibilidade causado pela posicionamento específico de P_1 e P_2 .

Dada a posição em que se encontram as peças representadas, os vértices em questão serão analisados relativamente à sua visibilidade para os seguintes casos:

1. v_1 e v_2 pertencem ambos a P_1 ;
2. v_1 e v_2 pertencem ambos a P_2 ;
3. v_1 pertence a P_1 e v_2 pertence a P_2 ;
4. v_1 pertence a P_2 e v_2 pertence a P_1 .

Antes de se abordar o problema convém relembrar as condições específicas a que esta situação se encontra associada: ambos os vértices v_1 e v_2 pertencem a ambas as peças P_1 e P_2 e não existe qualquer obstáculo¹⁰ entre eles. A multiplicidade de testes a que se vão sujeitar estes vértices é natural, mas, não obstante e dada a geometria do problema, estes têm de ser *necessariamente invisíveis*.

Tendo em conta a situação descrita, apresentam-se agora dois novos conceitos simples, fundamentais na compreensão do problema. Assim, define-se *visibilidade interna* como o tratamento dos casos, criação das listas e sublistas, que envolvem a visibilidade de uma peça P_i para ela mesma, e *visibilidade externa* como o tratamento dos casos, criação de listas e sublistas, que envolvem a visibilidade, quando existe, de uma peça P_i para uma peça P_j .

Os oito testes de visibilidade relevantes (por uma questão de simplificação não estão a considerar-se os testes de visibilidade directa, uma vez que não existe nenhum obstáculo entre os vértices considerados, tal como explicado em 2.3.1) a que se submeteram v_1 e v_2 são fruto da bidimensionalidade do problema aliada à lógica de implementação aplicada. Assim, no exemplo da figura 4.7 efectuaram-se dois testes de visibilidade interna (casos 1 e 2), cada um com dois testes de invisibilidade indirecta e dois testes de visibilidade externa (casos 3 e 4), cada um com dois testes de invisibilidade indirecta.

Por uma questão de simplificação de processos, refere-se que não existe qualquer problema ao nível dos testes dos casos 3 e 4 e que estes foram apenas aqui incluídos para melhor contextualização da situação. Ainda, também se sabe que, pelas condições do problema, todos os resultados de testes de v_2 relativamente a v_1 , são iguais aos resultados dos testes de v_1 relativamente a v_2 , pelo que só se irá analisar a situação em que v_2 é testado (ou seja, se v_2 é visível a partir de v_1). A ideia por detrás destas simplificações consiste em tentar cingir o problema às suas questões fundamentais, dada a enorme complexidade que normalmente as rodeiam.

Assim, fica-se reduzido aos casos relacionados com os testes da visibilidade interna. Para melhor se entender a natureza destes testes, separaram-se as duas peças da figura 4.7, que se encontram agora representadas na figura 4.8, tal como elas vão ser analisadas na implementação original, nomeadamente pelo teste da invisibilidade indirecta.

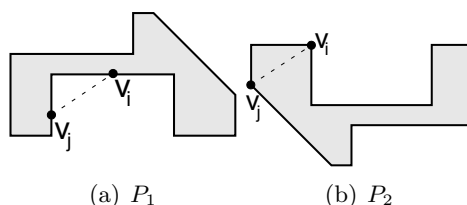


Figura 4.8: Representação geométrica das duas peças em análise.

¹⁰Tal como anteriormente referido, um obstáculo pode ser um aresta ou um vértice da mesma ou de outra peça.

O teste da invisibilidade indirecta encontra-se implementado por negação, ou seja, recorre a um teste auxiliar denominado como *visibilidade indirecta* que vai testar a seguinte situação: se v_j é adjacente de v_i . Caso tal não se verifique, esse mesmo teste determina, com base numa função que determina a posição relativa de um ponto face a um segmento de recta, se v_i é um vértice convexo (ou colinear) ou côncavo e determina a posição relativa de v_j face às arestas (v_{i-1}, v_i) e (v_i, v_{i+1}) . Esta análise da posição relativa permitirá concluir se v_j é visível a partir de v_i (se a aresta (v_i, v_j) não sobrepõe o interior de P_1) ou não (se a aresta (v_i, v_j) sobrepõe o interior de P_1).

Dada a forma como o teste se encontra implementado, rapidamente se conclui que para o caso da figura 4.8(a) os vértices são visíveis e que para o caso da figura 4.8(b), estes mesmos vértices não o são. A situação específica do encaixe destas duas peças faz com que seja criada uma estrutura de dados, no mínimo, ambígua. Isto é, na referida matriz *multidimensional* das visibilidades, existem duas entradas cuja informação é contraditória: em determinada posição da matriz considera-se que v_i não vê v_j (e vice-versa) e noutra posição da matriz considera-se que v_i vê v_j (e vice-versa). Tal não pode ocorrer sob pena de mau funcionamento do algoritmo, que se poderá traduzir em cortes irreversíveis de pedaços de peça.

Relativamente ao comportamento do algoritmo, considera-se que este realmente não apresenta nenhum resultado que não fosse esperado, dada a sua lógica de implementação. No entanto, e tal como já referido, estes vértices são, no contexto real, *invisíveis* e não pode haver outra interpretação. E uma vez que se pretende, ao nível dos problemas de posicionamento, cada vez maior aproveitamento do espaço (traduzido por um maior encaixe das peças), sabe-se que a tendência natural do *input* dos PDPC é incorporar inúmeras situações como a que foi aqui analisada, pelo que se considera que tal implementação não pode ser validada.

No seguimento do trabalho desenvolvido, exigia-se que o algoritmo principal funcionasse devidamente para que o processo de recodificação chegasse ao seu término em condições de bom funcionamento. Deste modo, e porque a análise efectuada à situação descrita foi conclusiva quanto à origem do problema, foi implementada uma solução para que a aplicação pudesse então funcionar devidamente, sem apresentar o referido problema de ambiguidade nas suas estruturas de dados.

Em termos práticos, e sem grandes detalhes sobre a codificação utilizada, foi introduzida uma alteração no teste da *invisibilidade indirecta* da função que executa a atribuição das visibilidades para a própria peça, já definida como *visibilidade interna*. Assim, e uma vez que a condição de paragem destes testes consiste na primeira vez que é encontrada uma *invisibilidade* entre os dois vértices, alterou-se a aplicação original de modo a que, para os dois vértices dados, estes sejam testados para a situação de invisibilidade *indirecta* não só para a peça a que pertencem, como também para todo o conjunto de peças colocadas na placa. Usando o exemplo que tem vindo a ser analisado, explicam-se os resultados obtidos para os seguintes testes (relevantes para o caso) efectuados:

- v_i e v_j pertencem ambos a P_1 : o *novo* teste da invisibilidade indirecta testa v_i e v_j para todos os casos em que ambos pertençam à mesma peça. Assim, v_i e v_j serão testados como pertencentes a P_1 e como pertencentes a P_2 , sendo que a aplicação irá agora determinar que estes são invisíveis após o teste com P_2 . No final, tem-se para este teste de visibilidade *interna* que os vértices em causa são *invisíveis*
- v_i e v_j pertencem ambos a P_2 : o *novo* teste da invisibilidade indirecta segue o procedimento acima referido e, no final, devolve que os vértices são, igualmente, *invisíveis*.

Relativamente à solução implementada convém, deste modo, realçar que a clara redundância dos testes efectuados é intencional. Dada a complexidade da aplicação e a sua lógica de implementação muito presa à lógica do programador original, torna-se extremamente precário fazer alterações sem, deste modo, provocar situações de mau funcionamento geral. Analisado o peso computacional da alteração introduzida, conclui-se que, pelo facto de só se efectuar o novo teste em casos em que ambos os vértices pertencem à mesma peça (o que não acontece tão frequentemente, daí não se terem previsto estas situações), que este é insignificante relativamente ao todo. Assim, tendo em conta este aspecto, refere-se que a intencionalidade da redundância apenas beneficia a robustez da aplicação.

Para todo o caso, conclui-se que a aplicação deixou de apresentar problemas na situação referida (e outras semelhantes) dada a coerência que a alteração confere às suas estruturas de dados.

A segunda situação de erro de funcionamento diz respeito a um problema de dualidade de critérios de implementação. É um erro de origem estrutural o que implica, por um lado, relativa importância em corrigi-lo e por outro, maior dificuldade em fazê-lo.

A explicação do problema que se vai abordar é complexa ao primeiro contacto, com a particularidade de não ser possível generalizar sem entrar em diversos detalhes. Assim, opta-se pela descrição exacta da ocorrência, apontando os detalhes importantes no seu decorrer assim como o erro encontrado. Como valioso auxiliar gráfico refere-se a figura 4.9, que contém a situação actual do percurso de corte: a tons de cinzento escuro encontram-se as peças já cortadas, a branco as peças ainda por cortar na placa, a traço engrossado as arestas já cortadas (excluindo todas as que pertenciam a peças já cortadas), a traço normal as arestas não cortadas e v_{actual} representa o vértice onde se encontra o percurso de corte neste instante. Este vértice foi escolhido por motivos que se prendem com a sequência de corte efectuada, ou seja, o corte da aresta ($v_{anterior}, v_{actual}$) provocou a queda de uma região de desperdício, representada na figura 4.9 pela zona de tom cinzento claro.

Antes de se pormenorizar a sequência de operações que geram o erro do programa convém dar uma perspectiva geral do que ocorre nesta situação particular. Quando testada esta instância, foi verificado que, dado um certo ponto da determinação do percurso, a aplicação entrava num *loop* infinito, sendo necessário terminá-la *à bruta*. Fazendo o paralelismo para a lógica de implementação utilizada, sabe-se que o referido ponto é o vértice

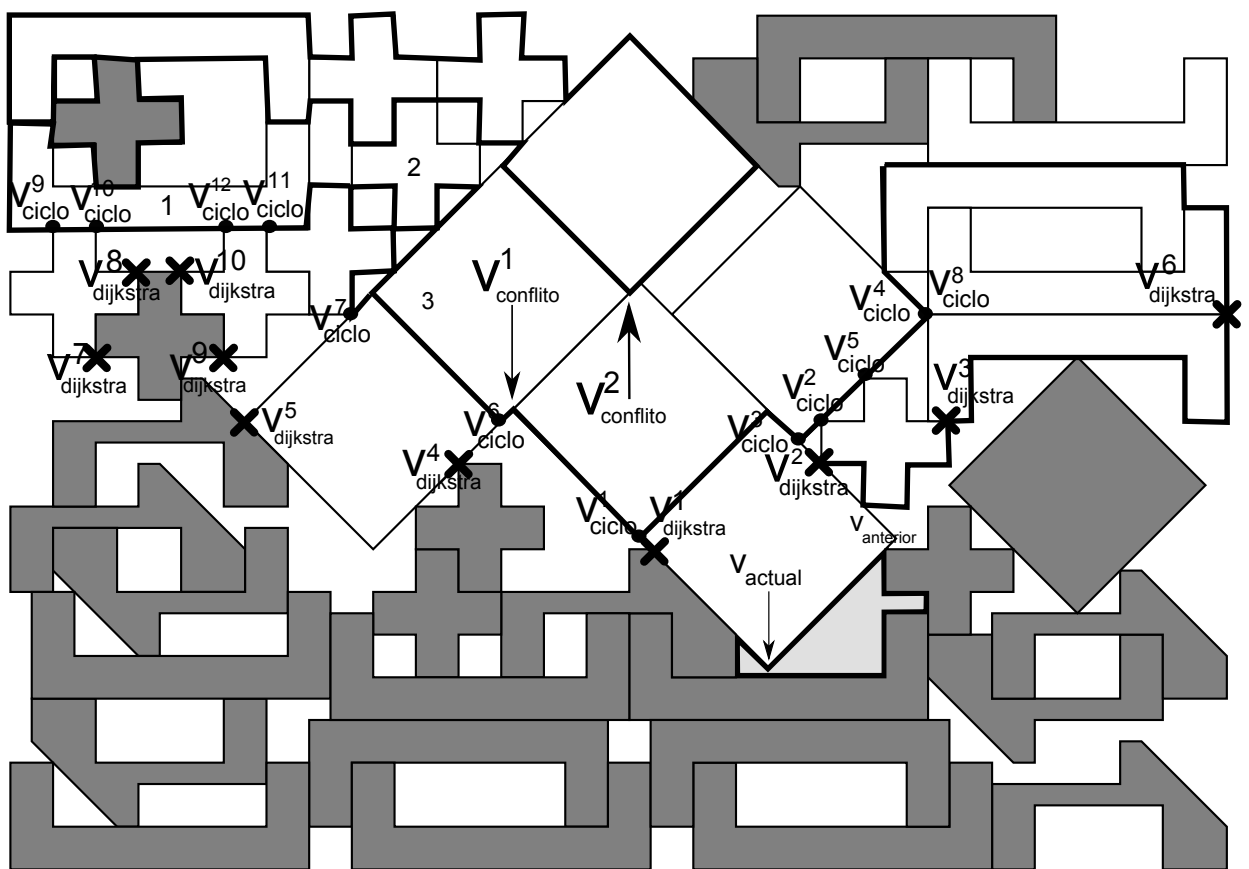


Figura 4.9: Representação geométrica da situação que provoca o erro em análise, por corte da aresta $(v^1_{conflito}, v^2_{conflito})$.

definido como v_{actual} na figura 4.9 e que o *loop* infinito em que o algoritmo entra tem origem na seguinte dualidade conflituosa de informação, num sentido muito lato: é encontrado um caminho mínimo para prosseguir o corte, mas esse caminho não é válido. Refira-se imediatamente que está subentendido na afirmação anterior que o caminho mínimo encontrado é válido para o algoritmo de determinação de caminho mínimo. Dada a natureza conflituosa do problema, não se irá tentar definir se o caminho mínimo encontrado é, de facto, válido ou não, uma vez que toda esta situação deriva de uma situação de erro. O importante a reter é realmente a situação de conflito no cerne da aplicação.

Assim, ao tentar validar um caminho que não é válido, a aplicação entra no referido *loop* infinito, do qual não consegue sair. Considere-se ainda na referida figura que as cruzes representam os vértices encontrados pelo Algoritmo de Dijkstra¹¹ quando não há um vértice atingível por aresta não cortada a partir de v_{actual} , que os círculos representam os vértices onde se formam ciclos inadmissíveis, definidos como v_{ciclo} , e que a seta aponta para um vértice, que aqui se vai tratar como $v_{conflito}^1$.

A situação de erro que se procura explicar, tem início após a queda da referida região de desperdício. Definiu-se como ponto de partida esta específica situação por se encontrar relacionada com um aspecto fundamental da lógica de implementação: o *reset* da memória. Relativamente a este aspecto, refere-se a sua importância pelo facto de já não ser possível retroceder no percurso, pelo que, qualquer tentativa de determinar o vértice seguinte do percurso que resulte numa inadmissibilidade, irá sempre retroceder até v_{actual} . Assim, partindo de v_{actual} , donde não existem vértices atingíveis por arestas não cortadas, ocorrem sequencialmente as seguinte etapas, relevantes:

1. O Algoritmo de Dijkstra (AD) encontra $v_{dijkstra}^1$ que vai terminar em v_{ciclo}^1 . Retrocede-se até v_{actual} .
2. O AD encontra $v_{dijkstra}^2$ que vai terminar em v_{ciclo}^2 e v_{ciclo}^3 . Esta situação ocorre porque $v_{dijkstra}^2$ possuía mais que uma alternativa a percorrer. Retrocede-se até v_{actual} .
3. O AD encontra $v_{dijkstra}^3$ que vai terminar em v_{ciclo}^4 e v_{ciclo}^5 . Esta situação ocorre porque $v_{dijkstra}^3$ possuía mais que uma alternativa a percorrer. Retrocede-se até v_{actual} .
4. O AD encontra $v_{dijkstra}^4$ que vai terminar em v_{ciclo}^6 . Retrocede-se até v_{actual} .
5. O AD encontra $v_{dijkstra}^5$ que vai terminar em v_{ciclo}^7 . Retrocede-se até v_{actual} .
6. O AD encontra $v_{dijkstra}^6$ que vai terminar em v_{ciclo}^8 (que é o mesmo vértice de v_{ciclo}^3 , simplesmente o percurso até ele é diferente). Retrocede-se até v_{actual} .
7. O AD encontra $v_{dijkstra}^7$ que vai terminar em v_{ciclo}^9 . Retrocede-se até v_{actual} .

¹¹Algoritmo de pesquisa que implementa a determinação do caminho mínimo para um grafo com arestas de custos não-negativos.

8. O AD encontra $v_{dijkstra}^8$ que vai terminar em v_{ciclo}^{10} . Retrocede-se até v_{actual} .
9. O AD encontra $v_{dijkstra}^9$ que vai terminar em v_{ciclo}^{11} . Retrocede-se até v_{actual} .
10. O AD encontra $v_{dijkstra}^{10}$ que vai terminar em v_{ciclo}^{12} . Retrocede-se até v_{actual} .
11. O AD encontra $v_{conflito}^1$.

Dado o estado actual do programa, sabe-se que o corte da aresta $(v_{conflito}^1, v_{conflito}^2)$ vai formar um ciclo inadmissível, provocando a queda das peças 1, 2 e 3 assim como as regiões de desperdício que as unem, detectado pela função implementada para o efeito. No entanto, uma vez que nem $v_{conflito}^1$, nem $v_{conflito}^2$ são marcados como vértices ciclo, o programa recursivamente chama o AD, que invariavelmente retorna o mesmo caminho, por *achar* que este é admissível. Assim, compreende-se o erro como uma dualidade de critérios de implementação que, naturalmente origina uma situação de conflito, quando verificadas as referidas condições.

A correcção deste erro implica mudanças estruturais na implementação do algoritmo. Dada a já referida complexidade do algoritmo, compreende-se que alterar o procedimento de validação de ciclos ou alterar o AD implicaria, de modo a serem coerentes entre si, destruir muita da lógica de implementação do programador original. Em jeito de alternativa propõe-se a criação de um procedimento que actualize as estruturas implicadas no AD, de modo a que este possa *saber* na iteração seguinte que a aresta $(v_{conflito}^1, v_{conflito}^2)$ origina um ciclo inadmissível. Deste modo encontram-se novos $v_{dijkstra}$ e evita-se o *loop* infinito. No entanto refere-se que, por questões de tempo, não foi possível terminar essa implementação.

Para terminar a exposição dos problemas encontrados, refere-se uma situação que não diz respeito a nenhum erro de implementação ou mau funcionamento da aplicação mas que, não obstante, se considerou um *problema encontrado*.

Dada a multiplicidade de estruturas de dados que são criadas e o elevado grau de complexidade que atingem, pela interacção das diversas classes implementadas, considera-se que não faria diferença no peso computacional da aplicação, se a classe *vértice* possuísse uma variável que guardasse a peça a que pertence. Independentemente dos detalhes de implementação, o que se pretende em termos práticos é que, numa estrutura lógica tão orientada ao vértice, este pudesse saber a peça que o contém. A relação unilateral (peça / vértice) existente permite, naturalmente, à peça saber os vértices que a constituem, no entanto, um determinado vértice existe, no contexto da aplicação, como objecto *solto*. Por diversas situações, uma ligação no sentido inverso (vértice / peça) teria tido elevada utilidade, associada a uma economia de inúmeros processos complexos, que foram utilizados na estrutura do programa para colmatar essa *falha*.

A questão referida envolve dois aspectos práticos que influenciaram a decisão de prosseguir ou não com o desenvolvimento de tal implementação: por um lado, é uma alteração

simples de fazer ao nível da sintaxe, mas por outro acarreta enormes alterações na estrutura lógica, muito *presa* à do programador original.

Assim, no seguimento desta indecisão, aliada à falta de tempo necessário para proceder à correcta alteração e respectivos testes, deixou-se a questão estudada e devidamente preparada para uma posterior implementação.

4.5 Heurística Construtiva “Peça Mais Próxima”

A ideia por detrás desta heurística procura implementar um critério de alto nível menos orientado ao vértice. Não só pela sua designação, mas também pela lógica que segue ao longo de todo o seu processo, o critério VMA é extremamente *guloso* em cada escolha do vértice seguinte e demasiado dependente das inúmeras estruturas de dados criadas para suportar esta *gulodice*. Assim, e tirando partido da implementação que permite ao vértice saber a que peça pertence, desenvolveu-se um critério que visa a diminuição considerável do peso computacional da aplicação e consequentes tempos de execução, assumindo um deliberado compromisso com a possibilidade de se determinarem percursos de corte *piores*, no sentido de mais longos, que os actuais.

O critério “Peça Mais Próxima” (PMP) surgiu por observação criteriosa dos resultados que vinham a ser obtidos com o algoritmo original. Ou seja, ainda que extremamente eficiente para algumas instâncias testadas, nomeadamente as instâncias reais para as quais a aplicação foi otimizada, esta apresentava alguns resultados mais fracos para outro conjunto de instâncias. Assim, um pouco por influência da abordagem descrita na secção 2.3.2, surgiu o critério PMP.

O nome escolhido para este critério encontra-se relacionado com o facto de se orientar o percurso de corte à *peça* e não *ao vértice*, como no critério VMA. Ou seja, dá-se prioridade ao corte completo de peças para evitar o adiar de situações complicadas e a formação do efeito *serpente* (relacionado com a forma como os movimentos da ferramenta de corte se efectuem sobre a placa, que será abordado em seguida). Um bom exemplo destas duas situações, pode ser observado na figura 4.9. Mantendo a perspectiva *gulosa* em cada escolha, o critério PMP pode então ser entendido como a escolha da peça mais *próxima* da peça actual que, naturalmente, ainda não foi totalmente cortada da placa. Na perspectiva da implementação deste critério encontra-se o seguinte aspecto, que irá ser detalhadamente abordado em seguida: maior independência sobre as inúmeras estruturas de dados existentes, nomeadamente a complexa matriz multidimensional das visibilidades entre vértices e a estrutura de memória.

Dado um ponto actual qualquer na determinação de um percurso de corte, o critério PMP só é aplicável quando tiver sido efectuado um movimento da ferramenta de corte que origine a queda de uma peça da placa. Dada esta situação, a escolha da peça seguinte recai sobre a peça mais próxima. Quando não existirem mais peças para cortar na placa, o processo é concluído.

Assim, é importante realçar dois conceitos simples que se encontram parcialmente implícitos na definição deste critério, por serem fundamentais para uma melhor compreensão do funcionamento da heurística:

- O corte de uma peça da placa é efectuado de *uma só vez*;
- A distância entre peças é representada pela distância entre os seus vértices.

Cortar uma peça de *uma só vez* equivale a dizer, no contexto deste problema, que escolhida uma determinada peça para ser cortada, pelo critério PMP, só se efectuará uma nova escolha quando *todas* as arestas da peça escolhida se encontrarem já cortadas. Ou seja, uma vez iniciado o corte de uma peça, todas as arestas dessa peça irão ser percorridas pela ferramenta de corte até que a peça já não possua mais arestas por cortar.

Uma vez que a distância entre vértices visíveis já se encontra disponível nas estruturas de dados originais, nomeadamente a matriz das *visibilidades*, resolveu adaptar-se a distância entre duas peças como a distância entre os vértices das suas peças. Tendo em conta a multiplicidade de valores de distância que irão surgir entre os vértices da peça actual e os restantes vértices das peças visíveis¹², refere-se que este detalhe fica imediatamente simplificado por apenas se considerar, para efeitos de distância à próxima peça, o vértice final da aresta que provocou o corte da peça à passagem da ferramenta de corte. Assim, dado o *vértice actual* pertencente à *peça actual*, onde se encontra a ferramenta de corte após a queda de uma peça, a escolha da *peça seguinte* recai na escolha da peça por cortar que possui o vértice (naturalmente visível a partir do *vértice actual*) com menor valor de distância entre eles.

Refere-se que esta adaptação acaba por ter vantagens relativamente ao conceito *real* de peças mais próximas, que seria matematicamente calculado através dos centros geométricos das peças. No entanto, facilmente se percebe que, dada a variada geometria das peças, o centro geométrico destas não apresenta vantagens para a escolha da *peça mais próxima*.

A partir do momento em que se definiu uma relação bilateral entre o vértice e a peça a que este pertence, torna-se evidente que esta nova relação possibilita uma boa alternativa para testar a mudança de heurística, dado que imediatamente transparecem dois factores relacionados com a diminuição da complexidade da aplicação: numa perspectiva de alto nível, as *visibilidades* relevantes fazem-se ao nível das peças e, para aproveitar algum legado da implementação original, através de uma matriz quadrada de entradas binárias, o que liberta capacidade de processamento não só na determinação do candidato a vértice seguinte como também na actualização da referida matriz. Se tivermos em consideração que, como referido em [22], a aproximação de linhas curvas por linhas poligonais conduz a um aumento do número de vértices de uma peça na ordem dos 1500%, torna-se evidente observar que a matriz multidimensional das visibilidades implementada na heurística VMA irá tomar dimensões absurdas quando existe um número considerável

¹²O conceito de visibilidade para este critério é o mesmo apresentado para o critério VMA, na secção 2.3.1.

de figuras com linhas curvas no problema. Ou seja, uma vez que a matriz multidimensional das visibilidades é integralmente percorrida em cada iteração do algoritmo original para determinar o candidato a vértice seguinte, o *multitamanho* que esta pode atingir é naturalmente preponderante nos tempos de execução do programa.

Na sequência da linha de pensamento que justifica a implementação deste critério pelo aumento de independência relativamente às estruturas de dados que este acarreta, torna-se necessário referir que a estrutura de memória implementada para o critério VMA toma um peso inferior na heurística PMP.

Como consequência directa dos aspectos inerentes ao critério PMP, a ferramenta de corte irá deixar de efectuar longos movimentos *em serpente*, tal como acontecia na heurística VMA, pelo seu critério de escolha do vértice mais acima, independentemente da peça a que este pertence. Foi observado que, em diversas situações, os movimentos da ferramenta de corte oscilavam entre um lado e outro da placa, num sentido positivo das ordenadas¹³ originando aquilo que se designou como movimentos *em serpente*: longas *tiras* de placa em forma de “S”, criadas pelo movimento do fio de cobre. Ainda que, tal como já referido, esta situação possa ter claras vantagens num determinado conjunto de instâncias (por exemplo, quando a placa se orienta sobre a vertical), verifica-se que, para alguns casos, esta pode ser extremamente prejudicial na medida em que se criam condições demasiado propícias para a formação de ciclos inadmissíveis. Assim, ao reduzir drasticamente este efeito *serpente*, por consequência directa do corte efectuado de *uma só vez*, reduz-se a possibilidade de ocorrerem movimentos de contorno das peças por cortar o que invariavelmente leva a situações de inadmissibilidades.

Directamente relacionada com as referidas inadmissibilidades, encontra-se a estrutura de memória. Esta estrutura, criada precisamente com o intuito da aplicação *decorar* os caminhos experimentados sem sucesso num passado recente, irá naturalmente perder algum peso, dada a muito menor frequência com que se verificam situações de inadmissibilidade de ciclos.

Assim, ficam definidos os traços fundamentais e detalhados alguns aspectos de implementação para uma alternativa na abordagem da resolução de um PDPC.

4.6 Comentários Finais

Neste capítulo foram abordados inúmeros tópicos importantes relacionados com a implementação do algoritmo de determinação de percursos de corte. Refere-se, para melhor entendimento do funcionamento geral da aplicação, o modo de processamento dos dados de entrada com relevante detalhe sobre a ponte efectuada para a criação das estruturas de dados da aplicação, usando um *parser* de XML.

¹³A orientação dos eixos é a mesma definida em 2.3.1.1.

Ao nível do processo de recodificação encontraram-se algumas dificuldades naturais, fruto da falta de conhecimento de Java. Todas essas pequenas questões foram ultrapassadas na revisão da implementação, onde se detalharam classes e estruturas de dados implementadas enquanto principais alicerces da heurística VMA implementada.

No entanto, todo o cerne do trabalho desta dissertação centra-se na secção dos problemas encontrados. Foi onde se encontraram as maiores dificuldades, mas foi também onde se batalharam os maiores desafios. A realidade da optimização combinatória toma assim formas imprevisíveis que ditam pontos de viragem nas decisões e planos de trabalho. Assume-se que os objectivos iniciais eram outros, mas conclui-se que foi uma experiência igualmente enriquecedora. Detectaram-se problemas e falhas na implementação original que impediam o correcto funcionamento do algoritmo de determinação de percursos de corte. Apresentaram-se soluções para os problemas e falhas encontrados, excepto para uma situação que ficou em aberto, ainda que tenha sido cuidadosamente analisada e referida uma possível solução.

Por fim, ainda que sem uma implementação concreta, cumpre-se um objectivo pessoal que passava por propor um novo critério de decisão para o algoritmo de determinação de percursos de corte. Este é devidamente fundamentado e são ainda referidas as suas principais diferenças, vantagens ou desvantagens, para o critério originalmente implementado.

Capítulo 5

Testes Computacionais

5.1 Instâncias Utilizadas

Neste capítulo foram utilizados dois conjuntos distintos de instâncias como base para os testes computacionais efectuados. Ainda que se considere relevante mostrar resultados obtidos com a aplicação original *lado a lado* com a aplicação recodificada, o objectivo destes testes passou fundamentalmente por testar o correcto funcionamento de ambas as versões, para um conjunto de instâncias geometricamente variadas. Tal como já referido, desvalorizam-se questões comparativas de *performance* em detrimento da robustez conferida às aplicações, face a casos extremamente específicos.

Assim, introduzem-se dois conjuntos de instâncias, que se encontram graficamente representados no anexo A, e cujas principais diferenças serão detalhadas em seguida:

- Instâncias “CCORTE”;
- Instâncias “ESICUP”.

Estes dois conjuntos de instâncias foram utilizados por motivos relacionados com a sua formatação e com a sua geometria. Tal como já referido em 4.2, sabe-se que o *input* das aplicações apresenta ficheiros de formatos diferentes, o que é naturalmente condicionante na escolha das instâncias que vão ser testadas em cada caso. Na nova aplicação é utilizado um ficheiro com o formato XML que segue um *schema* normalizado conhecido, ao passo que a aplicação original utiliza um conjunto de ficheiros de texto, por onde essa mesma informação se encontra espalhada. Refira-se, deste modo, que o processo de conversão entre os dois formatos de input é *manual* e, por isso, demorado. Ainda, foi uma conversão obviamente efectuada unilateralmente dado o interesse que há em transformar os ficheiros de *input* antigos no actual formato XML *standard*, e não o contrário.

No entanto, para efeitos práticos, associam-se os dois conjuntos de instâncias ao contraste de complexidade geométrica que apresentam. Por um lado, o conjunto de instâncias “CCORTE”, que tem origem nas instâncias usadas num contexto industrial real, é mais simples e, por outro, o conjunto de instâncias “ESICUP”, que tem origem

num conjunto de soluções de *nesting* trabalhadas e apresentadas pelo EURO Special Interest Group on Cutting and Packing (ESICUP), apresenta maior grau de complexidade, como explicado em seguida. O conjunto de instâncias “ESICUP” pode ser encontrado em <http://www.fe.up.pt/esicup/>.

5.1.1 Instâncias “CCORTE”

Nos testes computacionais foram utilizadas 10 instâncias do problema de determinação de percursos de corte. Estas instâncias derivam, tal como referido, de problemas reais da indústria metalomecânica, representando as peças as “cabeças de corte” que integram a produção de algumas das ferramentas de corte. Refera-se ainda que este foi o conjunto de instâncias utilizado em [22] e [23].

Na tabela 5.1 é apresentada uma descrição das principais características geométricas de cada instância, nomeadamente no que diz respeito ao número de peças diferentes que constituem o padrão de corte, número total de peças posicionadas e número de vértices do padrão após o seccionamento.

	<i>Peças diferentes</i>	<i>Total de peças</i>	<i>Total de vértices</i>
<i>p01</i>	2	19	153
<i>p02</i>	4	25	175
<i>p03</i>	4	30	248
<i>p04</i>	17	38	267
<i>p05</i>	2	58	232
<i>p06</i>	1	35	280
<i>p07</i>	1	26	312
<i>p08</i>	1	29	435
<i>p09</i>	2	52	372
<i>p10</i>	1	47	517

Tabela 5.1: Dados relativos às instâncias “CCORTE” utilizadas nas experiências computacionais.

5.1.2 Instâncias “ESICUP”

Nos testes computacionais que envolveram ficheiros XML, foram utilizadas 10 (de um total de 15) instâncias, que, como já referido, resultam de soluções de problemas de *nesting*. Estas instâncias representam, quer casos reais, quer casos propositadamente criados com intuito de aperfeiçoar o método de resolução dos problemas de posicionamento. Refira-se que este conjunto de instâncias, por não seguir um contexto *real*, não segue a orientação *topdown* que justifica a implementação do critério VMA, assim como tratando-se de um conjunto de instâncias cujo posicionamento foi efectuado com as peças encostadas à borda, podem acontecer situações de *aparente* queda de peça. Por razões óbvias, não se apresentam as situações em que o algoritmo não corre até ao fim, devido ao erro detalhadamente apresentado em 4.4.

Tal como já referido, este conjunto de instâncias apresenta maior complexidade geométrica, pelo que acaba por ser mais exigente em termos de determinar uma solução para o problema. Não só cada peça individualmente, como também o seu complexo empacotamento, fazem destes problemas uma boa prova de validação da aplicação recodificada.

Na tabela 5.2 é apresentada uma descrição das principais características geométricas de cada instância, concretamente no que diz respeito ao número de peças diferentes que constituem o padrão de corte, número total de peças e número de vértices iniciais do padrão. Na última coluna é possível perceber o âmbito em que cada instância se contextualiza, nomeadamente os casos em que se apresentam peças muito simples mas com enorme grau de encaixe entre elas. Estas situações proporcionam diferentes dificuldades ao problema, sendo o caso extremo as situações de *puzzle* das instâncias *DIGHE1* e *DIGHE2*.

	<i>Peças diferentes</i>	<i>Total de peças</i>	<i>Total de vértices</i>	<i>Tipo de problema</i>
<i>FU</i>	12	12	43	convexo
<i>JAKOBS1</i>	25	25	150	artificial
<i>JAKOBS2</i>	25	25	134	artificial
<i>BLAZ</i>	7	28	176	artificial
<i>DIGHE1</i>	16	16	62	puzzle
<i>DIGHE2</i>	10	10	47	puzzle
<i>ALBANO</i>	8	24	164	confeções
<i>DAGLI</i>	10	30	186	confeções
<i>MAO</i>	9	20	174	confeções
<i>MARQUES</i>	8	24	162	confeções
<i>SHIRTS</i>	8	99	599	confeções
<i>SWIM</i>	10	48	960	confeções
<i>TROUSERS</i>	17	64	388	confeções
<i>SHAPES0</i>	4	43	391	artificial
<i>SHAPES1</i>	4	43	391	artificial

Tabela 5.2: Dados relativos às instâncias “ESICUP” utilizadas nas experiências computacionais.

5.2 Experiências Computacionais

A principal intenção destas experiências computacionais é validar a aplicação recodificada. Refira-se ainda que, dadas as correcções que foram introduzidas de modo a colmatar os erros encontrados e descritos no capítulo anterior, não é de esperar a obtenção de resultados similares, apenas de uma ordem de grandeza comparativa. Ainda, salienta-se que não são esperados grandes ganhos na eficiência dos algoritmos, pelas razões já mencionadas.

Todas as experiências computacionais foram efectuadas usando um computador pessoal equipado com um processador *Intel Core 2 Duo* com dois núcleos de processamento, cada um a funcionar a 2Ghz, e ainda 1.5Gb de memória RAM. Todos os algoritmos correm sobre *Cygwin*, um emulador de Linux para o Sistema Operativo Windows Vista, 32 bits.

O processo de experiências foi organizado em duas fases: primeiro correram-se as instâncias “CCORTE”, para as quais existe um conjunto de dados para comparação, obtidos em [22] e em seguida testaram-se as instâncias do conjunto “ESICUP”, que serve apenas como forma de validação para posicionamentos de peças com outras características geométricas.

Os resultados obtidos para as instâncias “CCORTE” encontram-se apresentados na figura 5.3. Na primeira coluna apresenta-se a designação da instância e na coluna seguinte o número total de vértices obtidos após o seccionamento das arestas que, pela lógica implementada no algoritmo é naturalmente igual ao número de arestas seccionadas existentes no posicionamento. Nas duas colunas seguintes apresenta-se, respectivamente, o tempo total de execução, em segundos e a distância total do percurso, em milímetros, como sendo a distância *total* dos movimentos efectuados pela ferramenta de corte desde que inicia o seu movimento até à sua paragem final. Finalmente, nas últimas duas colunas apresentam-se os comprimentos do percurso de corte do algoritmo recodificado e do percurso de corte do algoritmo original, baseados na heurística VMA ([22]), como sendo a distância que a ferramenta de corte percorre *na placa* desde a primeira aresta que corta até que executa o movimento de saída na horizontal.

	<i>Total de vértices</i>	<i>Tempo de execução (s)</i>	<i>Distância percorrida (mm)</i>	<i>Percurso de corte da recodificação (mm)</i>	<i>Percurso de corte original (mm)</i>
<i>p01</i>	183	82	971,47	941,40	941,40
<i>p02</i>	230	98	1014,71	965,74	972,38
<i>p03</i>	302	173	1206,43	1170,60	1017,50
<i>p04</i>	348	261	1384,87	1317,02	1317,02
<i>p05</i>	351	350	1143,06	1100,55	1100,56
<i>p06</i>	372	280	1274,88	1198,03	1198,03
<i>p07</i>	372	262	1141,76	1086,07	1087,43
<i>p08</i>	493	444	1207,87	1145,43	1122,24
<i>p09</i>	513	541	1436,48	1407,71	1407,37
<i>p10</i>	639	501	1554,29	1473,91	1476,01

Tabela 5.3: Resultados obtidos para as instâncias “CCORTE”.

Salienta-se que a última coluna é apenas para comparação relativa, ficando validada a aplicação recodificada para o conjunto de instâncias originais. Os valores a negrito representam a igualdade nos percursos de corte obtidos comparativamente aos originais ou, em caso de desigualdade, os valores inferiores para o par de resultados obtidos.

Nas instâncias *p01*, *p04*, *p05*, *p06* e *p09* salienta-se que a nova versão obteve os mesmos percursos que a versão original. Já nas instâncias *p02* e *p07* a nova versão consegue obter melhores resultados, enquanto em duas das restantes instâncias, *p03* e *p08*, os resultados são piores que a versão original. A explicação para estas diferenças não foi encontrada, mas poderá dever-se às alterações introduzidas anteriormente descritas. Finalmente a instância *p10* merece uma referência especial, já que a solução obtida pela nova versão apresenta um erro: duas peças são cortadas *a meio* (situação claramente visível quando se efectua o percurso passo a passo). Este erro não ocorre na versão original do algoritmo. Deste

modo admite-se que existe algum detalhe que carece de resolução numa futura abordagem ao problema.

Os resultados obtidos para as restantes instâncias encontram-se representados na figura 5.4. Na primeira coluna apresenta-se a designação da instância e na coluna seguinte o número total de vértices obtidos após o seccionamento das arestas que, pela lógica implementada no algoritmo é naturalmente igual ao número de arestas seccionadas existentes no posicionamento. Nas duas colunas seguintes apresenta-se, respectivamente, o tempo total de execução, em segundos e a distância total do percurso, em milímetros.

	<i>Total de vértices</i>	<i>Tempo de execução (s)</i>	<i>Distância percorrida (mm)</i>
<i>FU</i>	56	0	596,69
<i>JAKOBS1</i>	197	7	838,88
<i>JAKOBS2</i>	168	6	1377,21
<i>BLAZ</i>	204	13	658,28
<i>DIGHE1</i>	62	1	1618,28
<i>DIGHE2</i>	47	1	1266,42
<i>ALBANO</i>	182	11	177590
<i>DAGLI</i>	240	20	1785,39
<i>MAO</i>	178	7	66559,8
<i>MARQUES</i>	199	9	2123,05
<i>SHAPES0</i>	-	-	-
<i>SHAPES1</i>	-	-	-
<i>SHIRTS</i>	-	-	-
<i>SWIM</i>	-	-	-
<i>TROUSERS</i>	-	-	-

Tabela 5.4: Resultados obtidos para as instâncias “ESICUP”.

Salienta-se que, dado o aumento do grau de complexidade, ao nível geométrico, para este conjunto de instâncias, a aplicação foi recodificada com sucesso como se pode constatar pela validação dos resultados observados. Em algumas das instâncias, salienta-se o facto de não se determinar um percurso de caminho mínimo, naturalmente devido a erros externos ao contexto deste trabalho (relacionados com uma errada solução de posicionamento na instância *SHAPES0*), ou com origem no erro apresentado no capítulo anterior, na secção 4.4. Comparando tempos de execução para os dois conjuntos de instâncias, pode-se constatar que estes apresentam diferenças bastante significativas, para uma mesma ordem de grandeza no número de vértices associados ao posicionamento. Tal facto é perfeitamente justificado por se considerar uma placa circular (aproximada a um polígono de 729 vértices) no primeiro conjunto de instâncias e uma placa rectangular (naturalmente com 4 vértices) no segundo conjunto de instâncias.

Capítulo 6

Conclusões

Termina-se este trabalho de dissertação efectuando um conjunto de considerações gerais sobre os principais aspectos abordados.

Relativamente ao primeiro capítulo refere-se que constitui uma importante contextualização do tema desta dissertação, abordando não só o enquadramento genérico como também o contexto industrial específico.

No capítulo 2, segue-se uma linha de raciocínio que visa chegar ao Problema da Determinação dos Percursos de Corte (PDPC) como uma situação particular dos Problemas de Rotas em Arcos. Estudaram-se referências importantes do Problema do Carteiro Rural (PCR), como principal modelo teórico que suporta o PDPC. Foi detalhadamente analisada a variante *dinâmica* do PCR, por ser a única referência conhecida sobre o assunto. Tendo em consideração o PDPC, começa-se depois por explicar o que são estes problemas e a sua utilidade no contexto quotidiano. Dentro da variante dinâmica do problema, estudaram-se duas abordagens heurísticas e apresentaram-se os seus principais conceitos.

O capítulo 3 não permite grande margem de interpretação, uma vez que é uma descrição detalhada, com incidência na utilidade específica do PDPC, das ferramentas que se consideraram necessárias para apoiar o trabalho a desenvolver. Incide-se sobre a necessidade de se adquirirem conhecimentos para o desenvolvimento de uma página *web* para suporte ao trabalho.

O trabalho realizado e o verdadeiro desafio desta dissertação surge no capítulo 4. A principal secção apresenta questões ligada à heurística VMA: análise de implementação, processo de recodificação, problemas encontrados e soluções propostas. Detalha-se tudo na medida do possível, sempre tendo em vista o cuidado de não entrar em descrições demasiadamente complexas para quem não apresenta as bases necessárias dos PDPC. O culminar do trabalho efectuado é atingido com a proposta de um novo critério de decisão com o objectivo de implementar uma heurística alternativa à *gulodice* da heurística original. São perspectivas futuras num âmbito contextual que é muito vasto e pode tomar diversos rumos.

As experiências computacionais não carecem de grandes comentários, uma vez que

se julgam ser auto-explicativas. Ou seja, dado o facto de não serem efectuados testes comparativos, os valores apresentados servem “apenas” para validar a boa recodificação implementada assim como confirmar as correcções que haviam sido introduzidas.

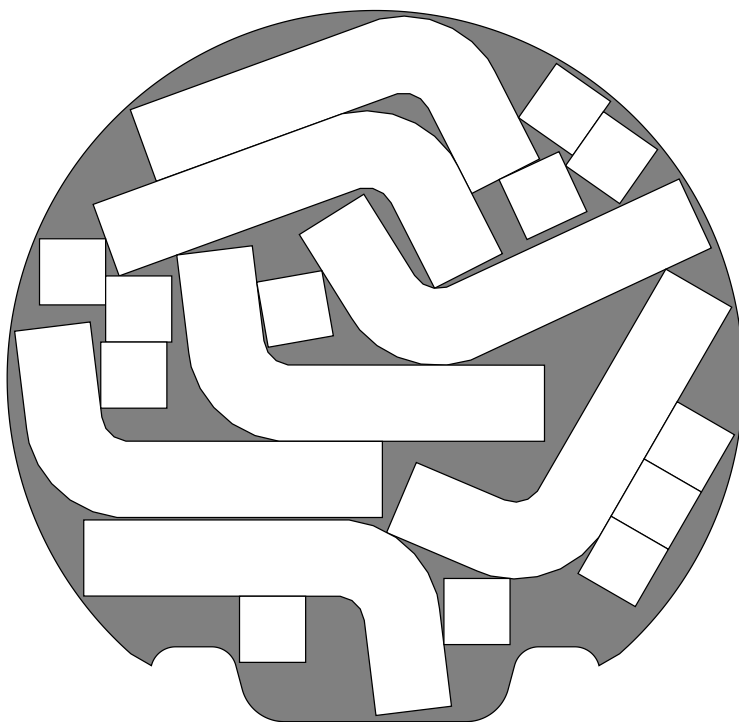
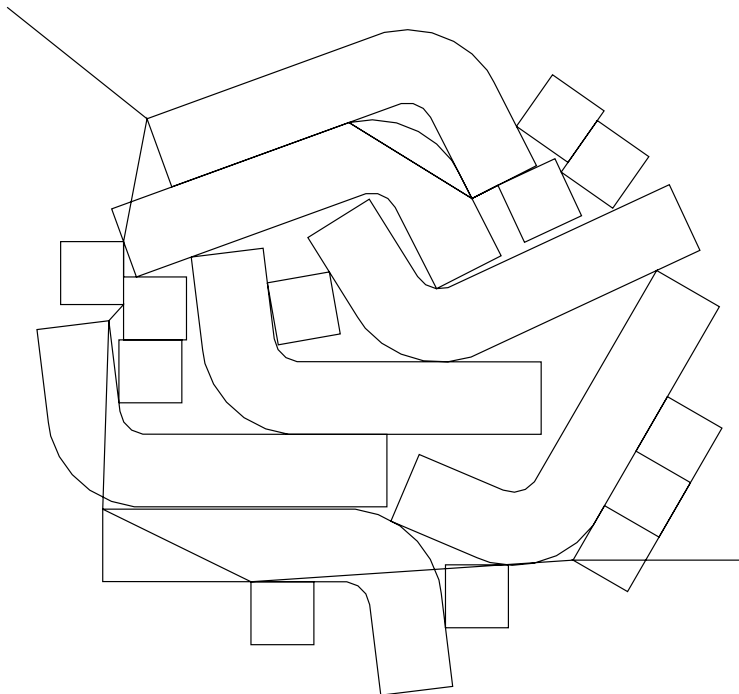
Os principais objectivos foram claramente cumpridos, sem desvios significativos da proposta de trabalho apresentada inicialmente. Termina-se com uma nota pessoal de enorme satisfação e um recompensador sentimento de *fulfilling*.

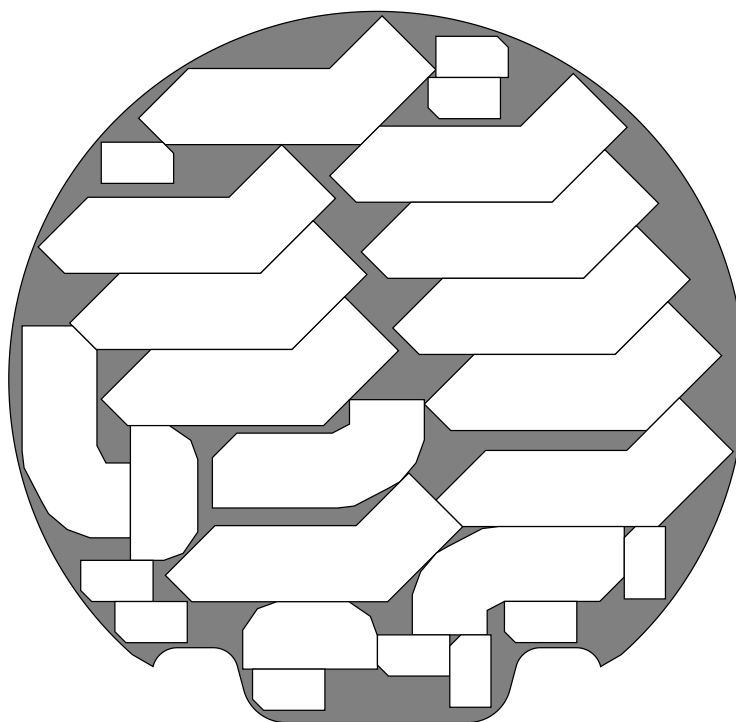
Anexo A

Percursos de corte obtidos nas experiências computacionais

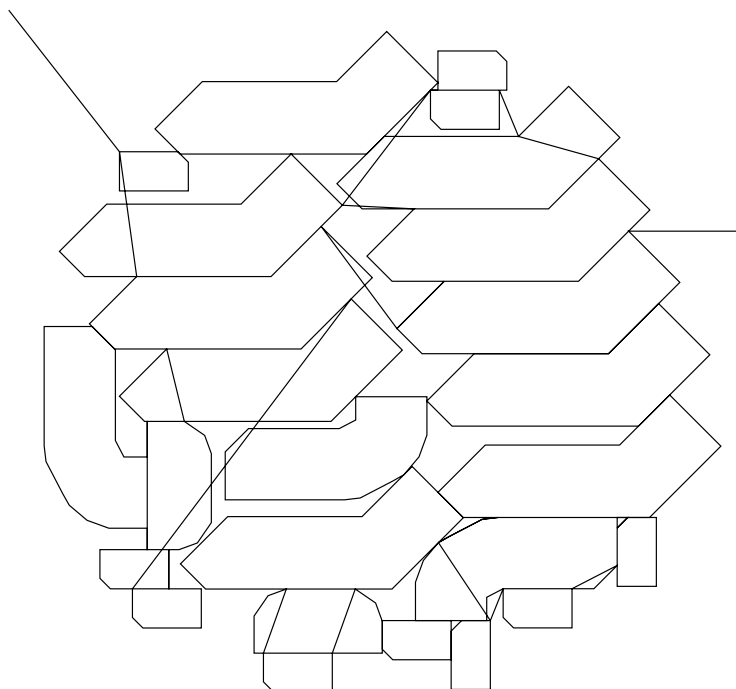
Neste anexo apresentam-se as instâncias utilizadas nas experiências computacionais descritas no capítulo 5 desta dissertação. Cada imagem representada segue a seguinte lógica: em cima encontra-se o padrão de corte e em baixo o respectivo percurso encontrado pela versão recodificada do algoritmo. As figuras que representam o percurso de corte são representadas por uma linha que começa na origem do referencial utilizado (num ponto acima e à esquerda da placa) e irá terminar num ponto necessariamente à direita da placa. Por questões de tamanho das imagens, salienta-se que algumas das figuras se encontram lado a lado ao invés do que tinha sido referido.

Os problemas de *nesting* passam por encontrar dois tipos de soluções distintos: maximização do número de peças colocadas ou minimização do desperdício causado pela colocação das placas. Em termos práticos para o contexto deste anexo, a diferença reside no facto de no primeiro caso existir uma placa de tamanho finito, que irá ser representada nas figuras, e no segundo caso, a placa possuir tamanho infinito, pelo que naturalmente não irá ser representada.

(a) Padrão de corte $p01$.(b) Percurso de corte para $p01$.Figura A.1: Instância $p01$.

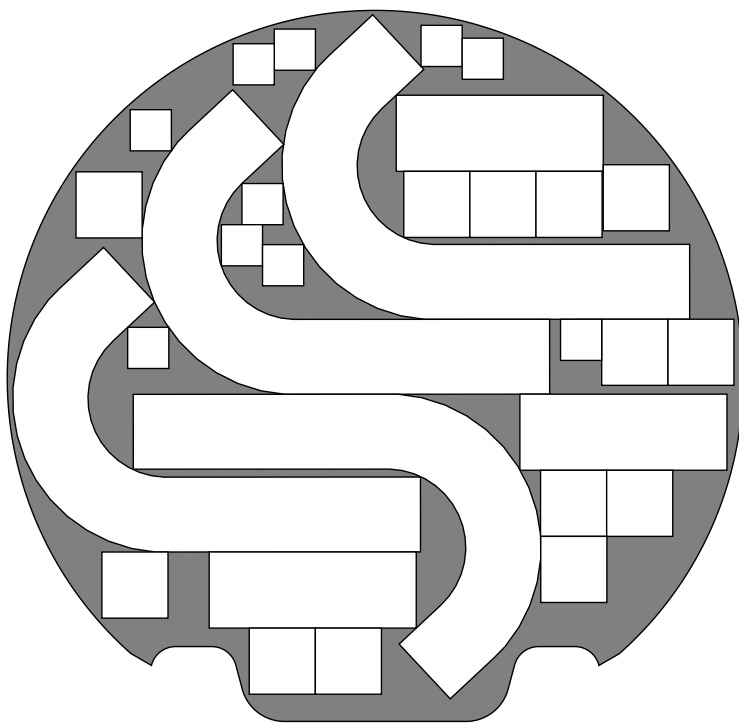
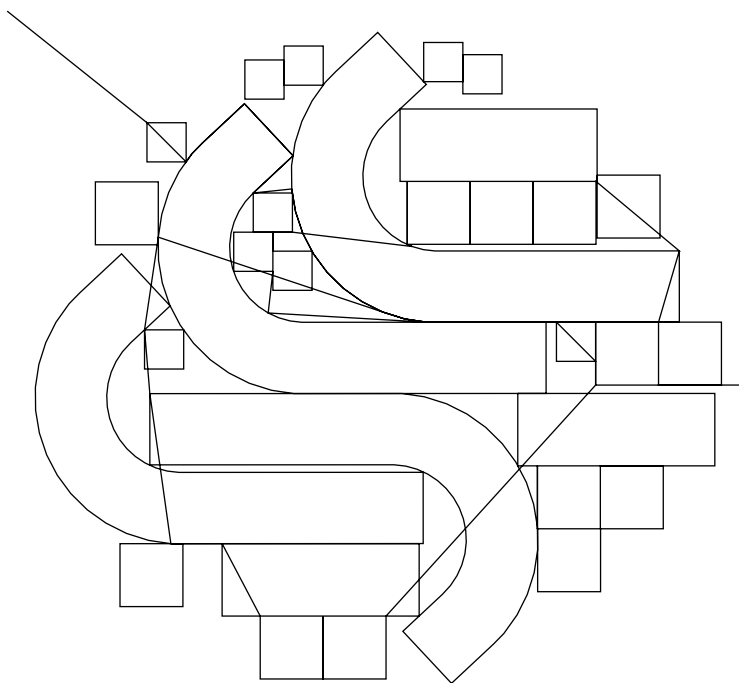


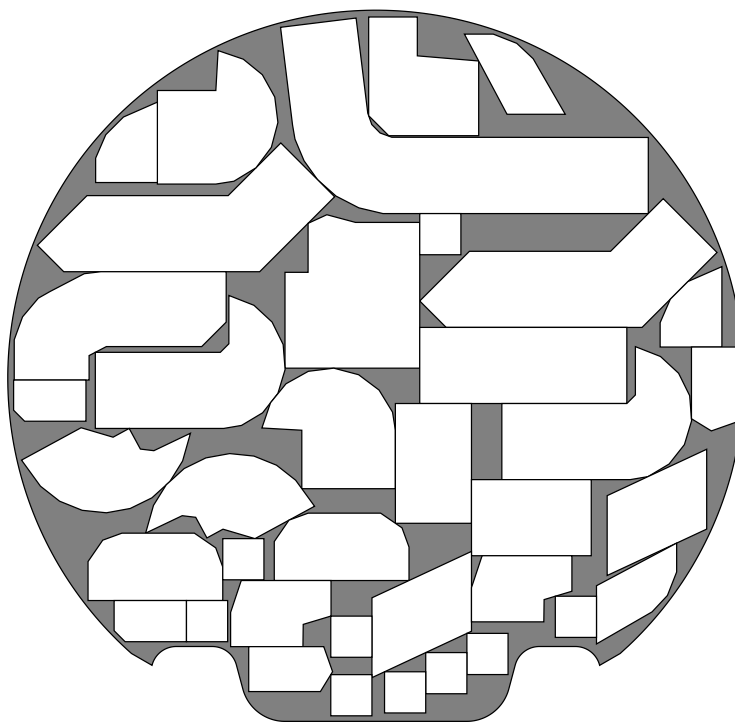
(a) Padrão de corte $p02$.



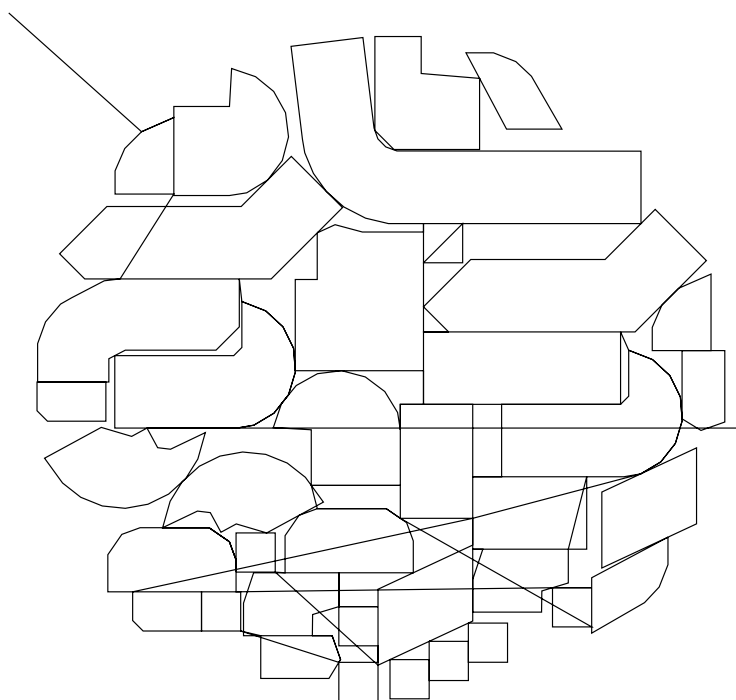
(b) Percurso de corte para $p02$.

Figura A.2: Instância $p02$.

(a) Padrão de corte $p03$.(b) Percurso de corte para $p03$.Figura A.3: Instância $p03$.



(a) Padrão de corte $p04$.



(b) Percurso de corte para $p04$.

Figura A.4: Instância $p04$.

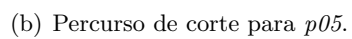
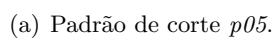
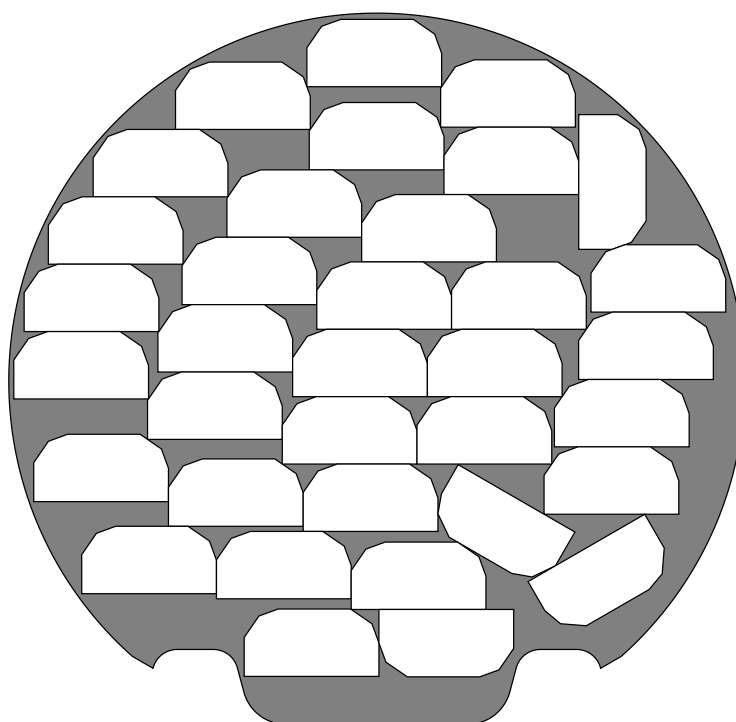
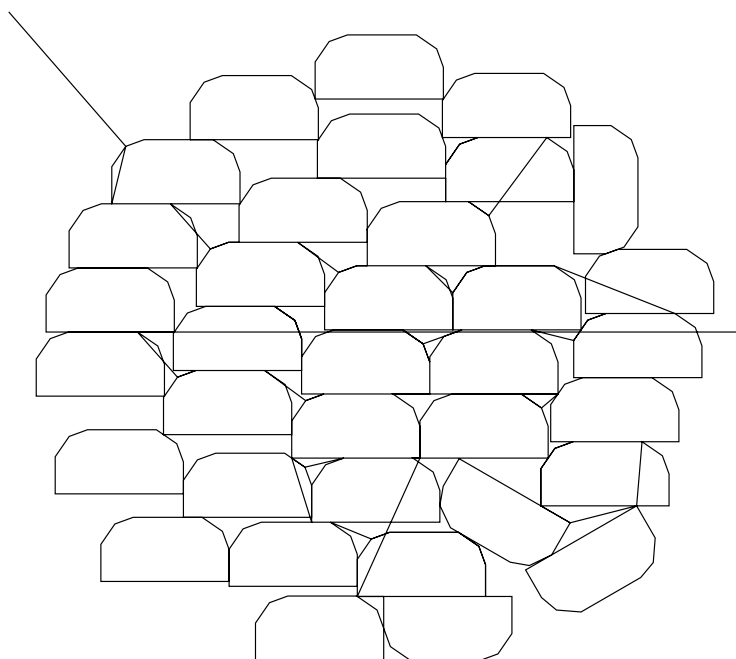


Figura A.5: Instância *p05*.

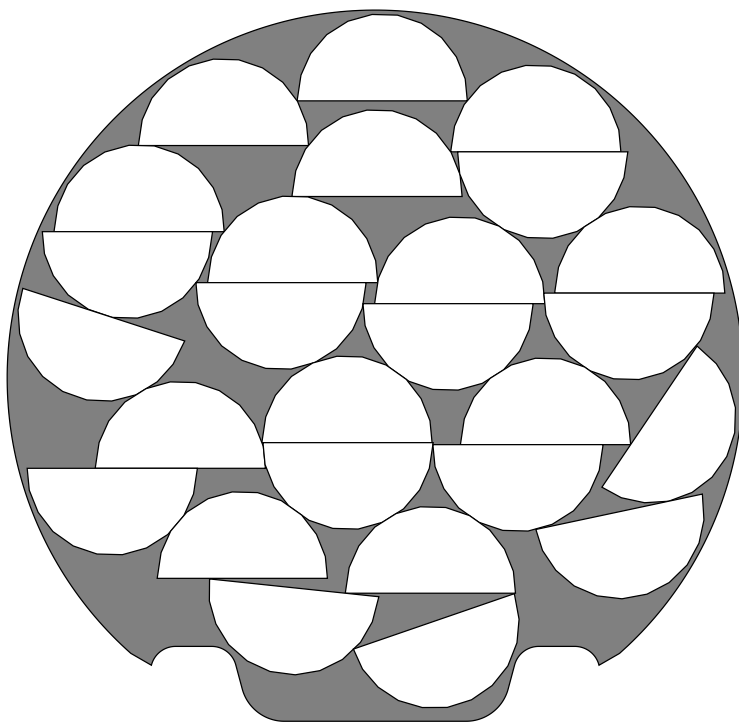
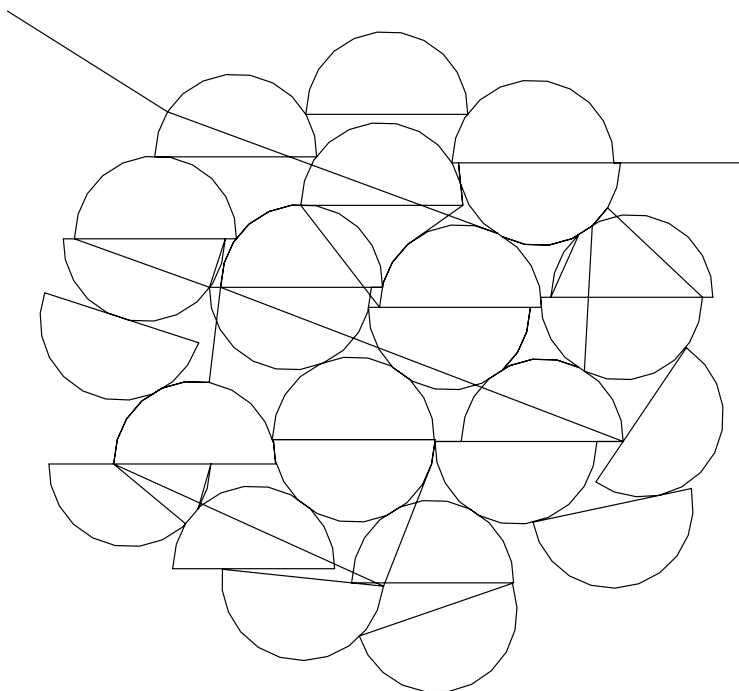


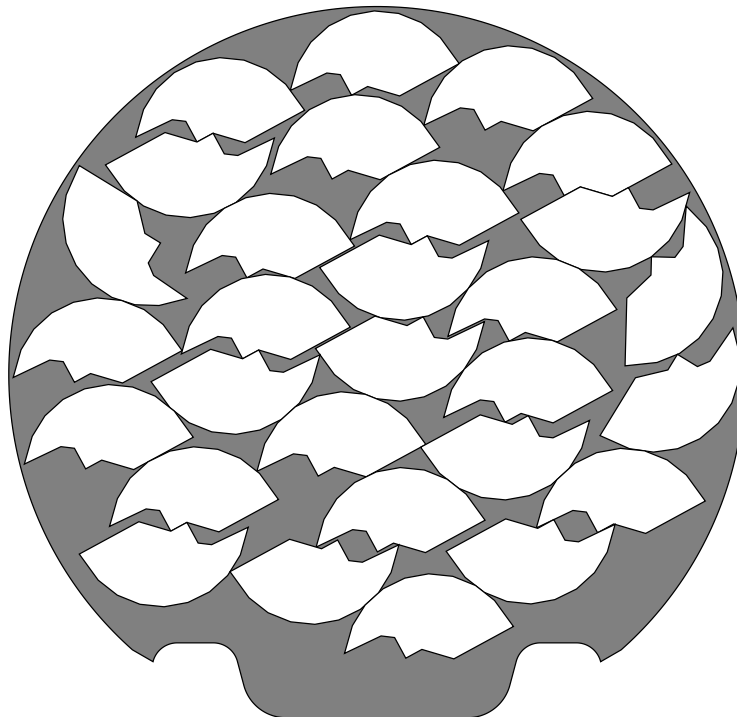
(a) Padrão de corte $p06$.



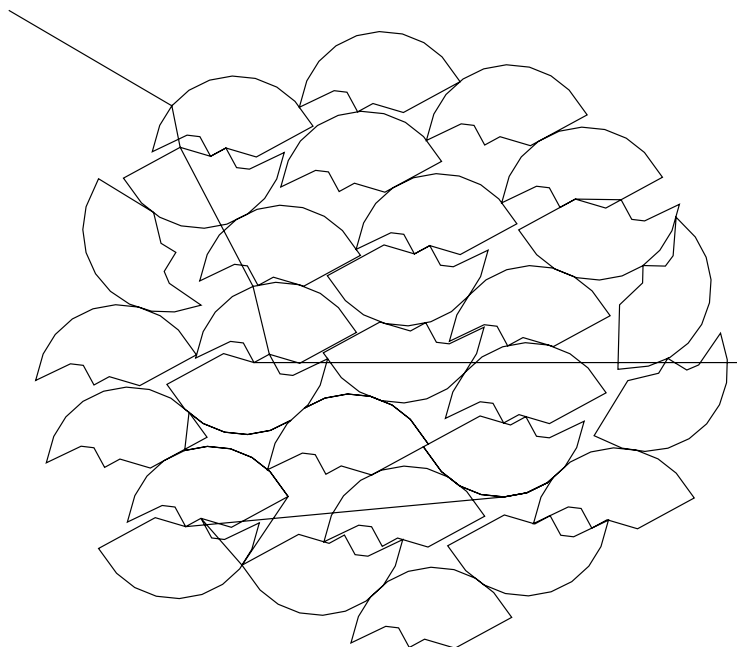
(b) Percurso de corte para $p06$.

Figura A.6: Instância $p06$.

(a) Padrão de corte $p07$.(b) Percurso de corte para $p07$.Figura A.7: Instância $p07$.

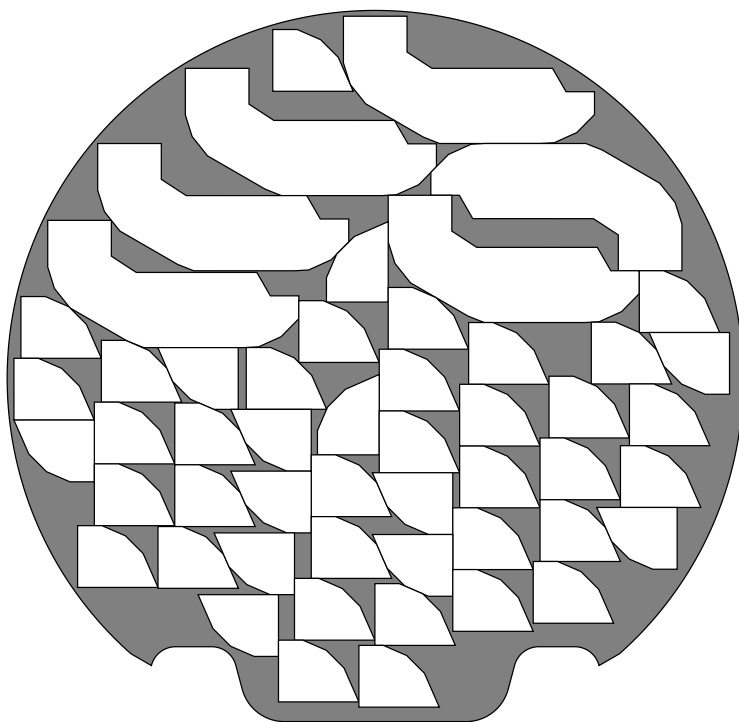
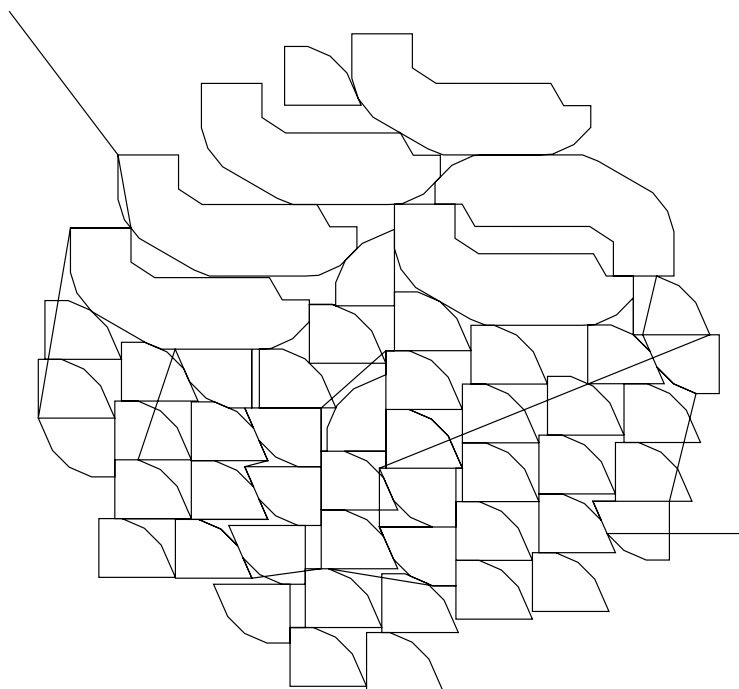


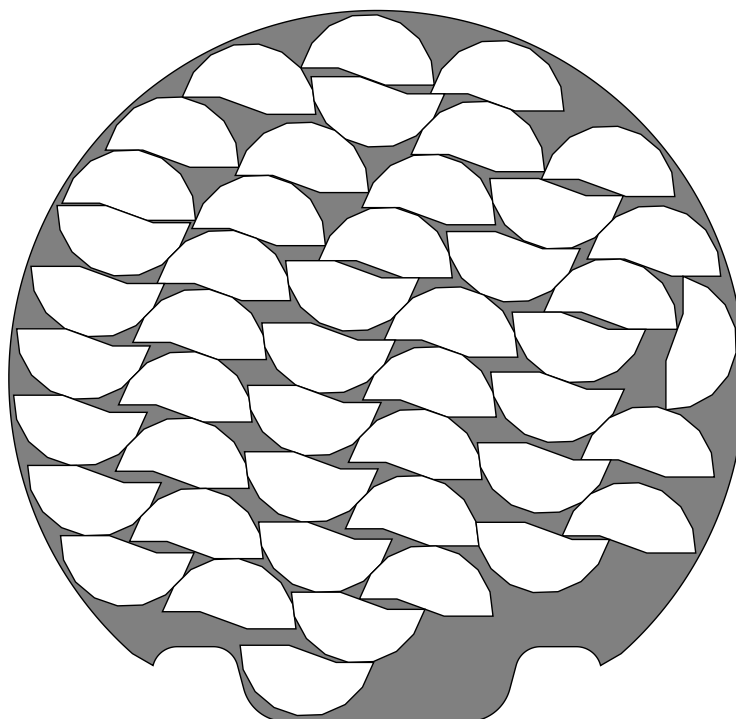
(a) Padrão de corte *p08*.



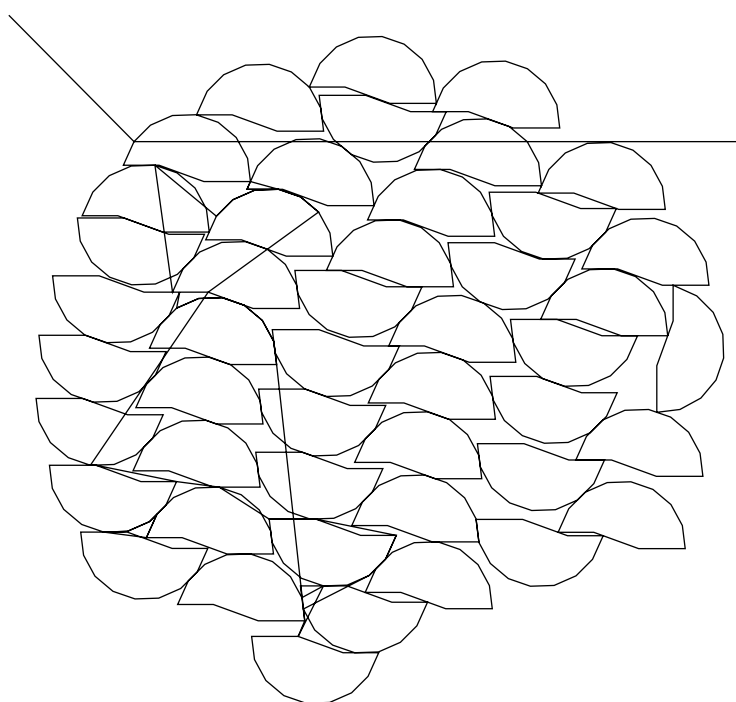
(b) Percurso de corte para *p08*.

Figura A.8: Instância *p08*.

(a) Padrão de corte $p09$.(b) Percurso de corte para $p09$.Figura A.9: Instância $p09$.

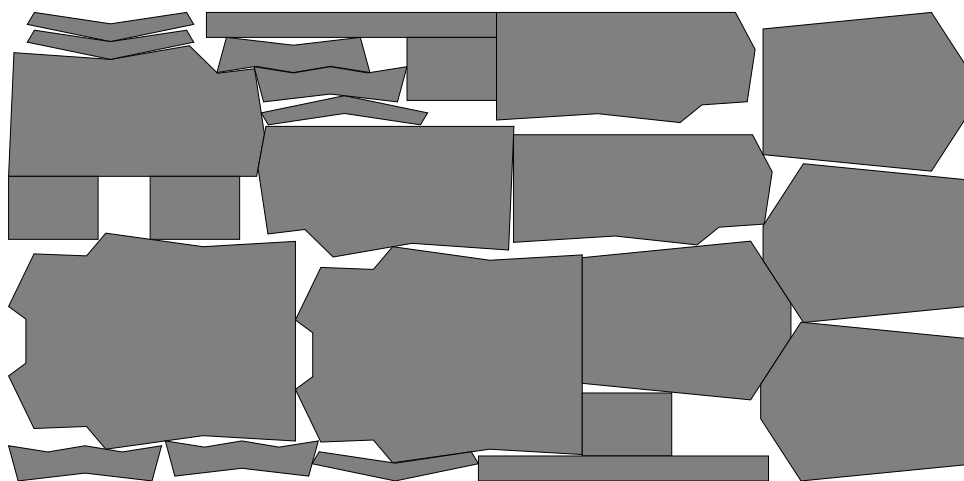


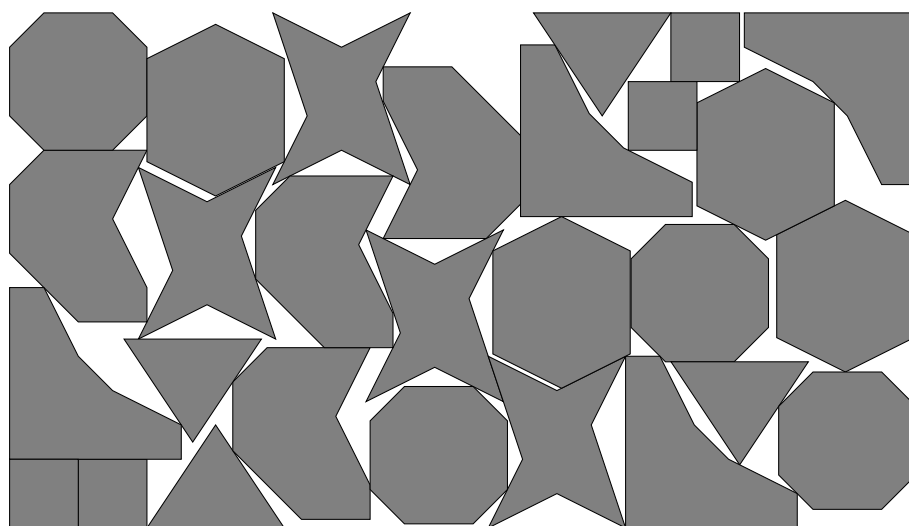
(a) Padrão de corte $p10$.



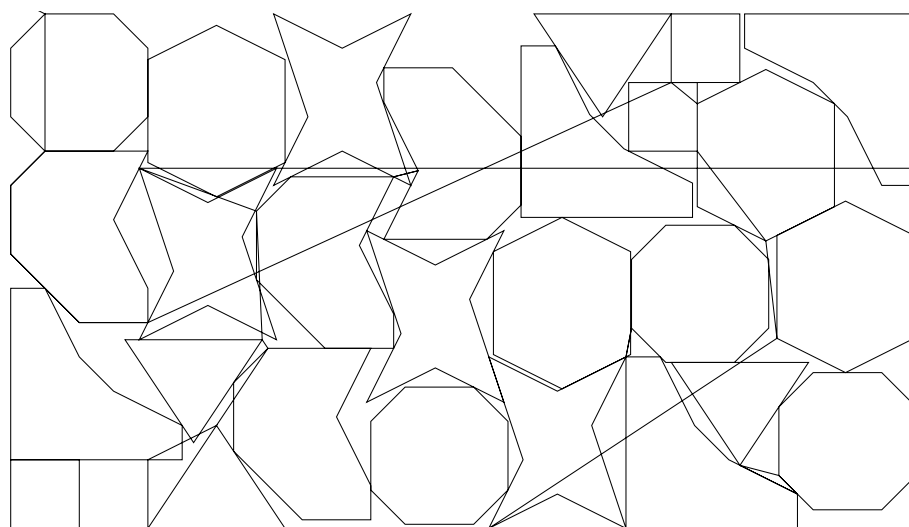
(b) Percurso de corte para $p10$.

Figura A.10: Instância $p10$.

(a) Padrão de corte *ALBANO*.(b) Percurso de corte para *ALBANO*.Figura A.11: Instância *ALBANO*.

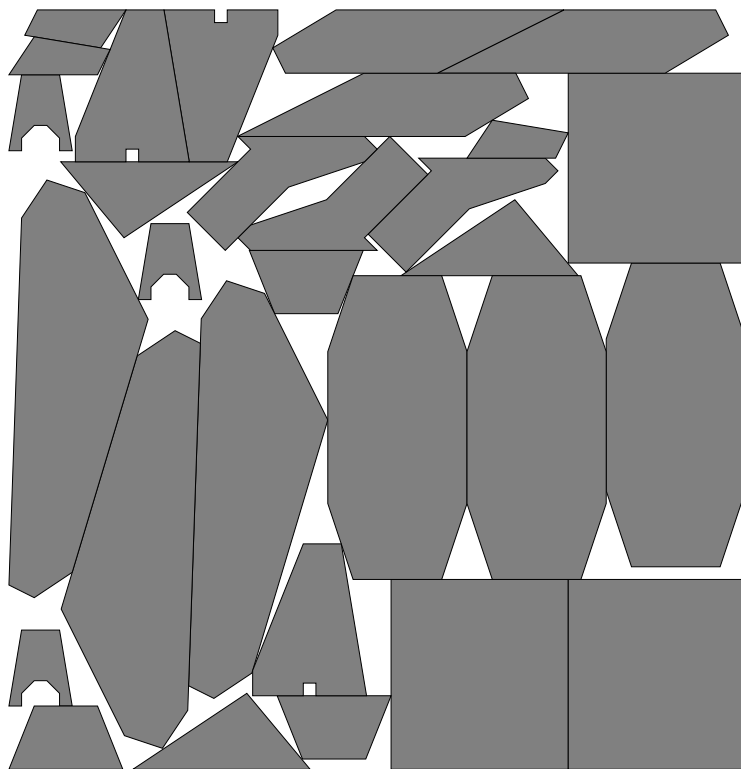
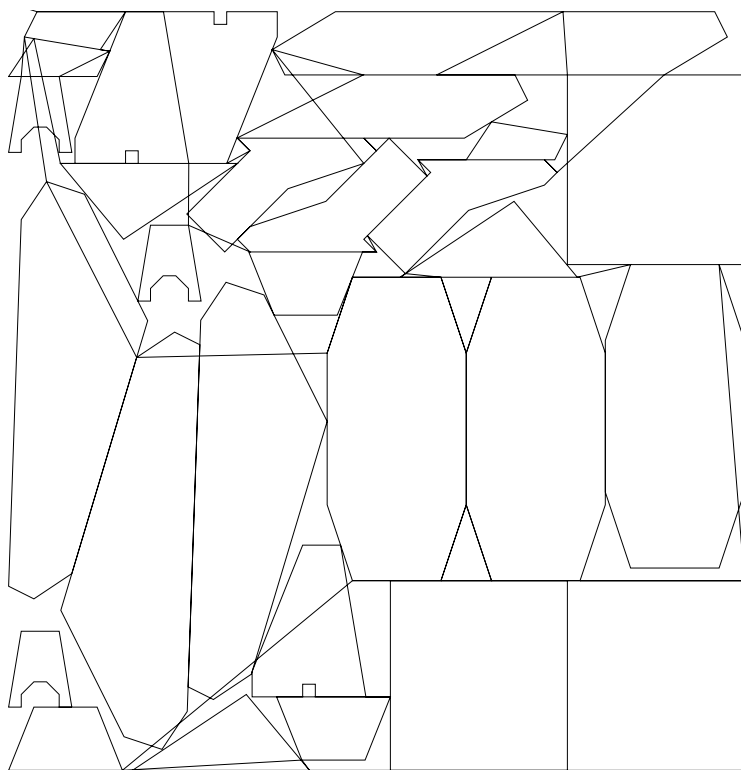


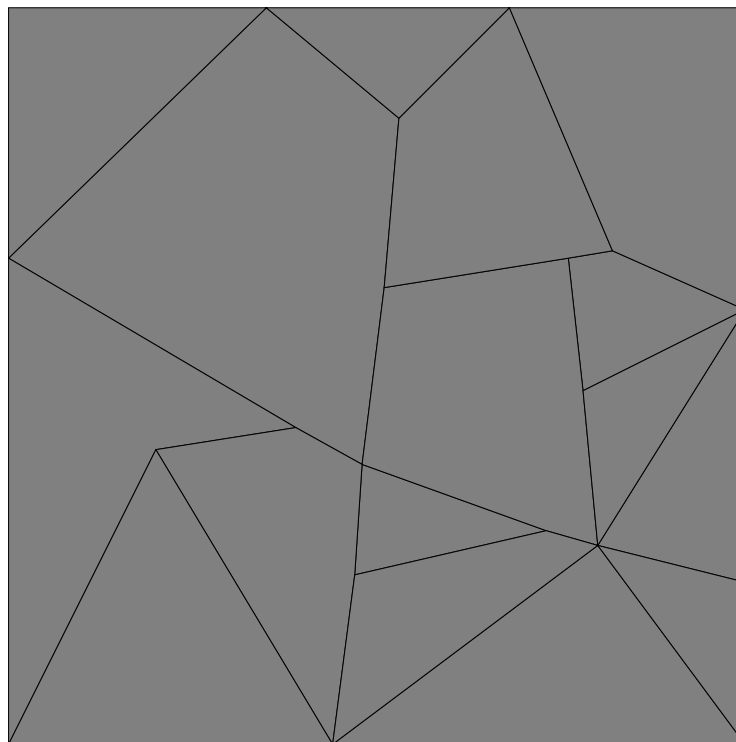
(a) Padrão de corte *BLAZ*.



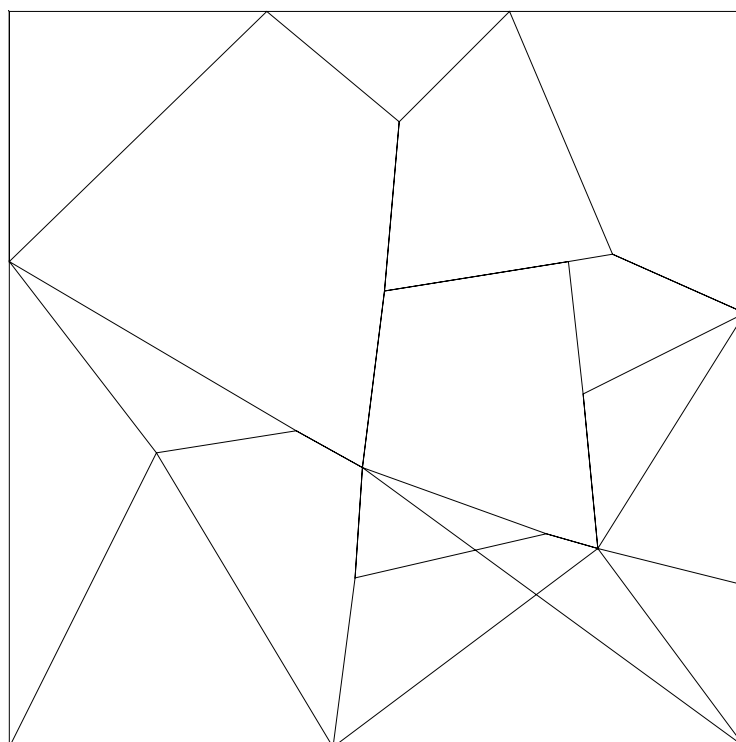
(b) Percurso de corte para *BLAZ*.

Figura A.12: Instância *BLAZ*.

(a) Padrão de corte *DAGLI*.(b) Percurso de corte para *DAGLI*.Figura A.13: Instância *DAGLI*.

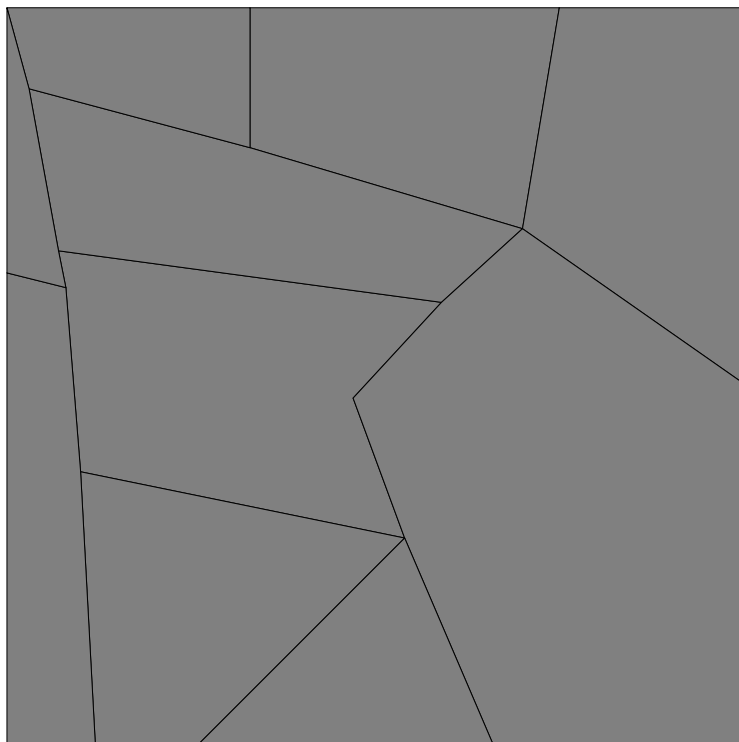
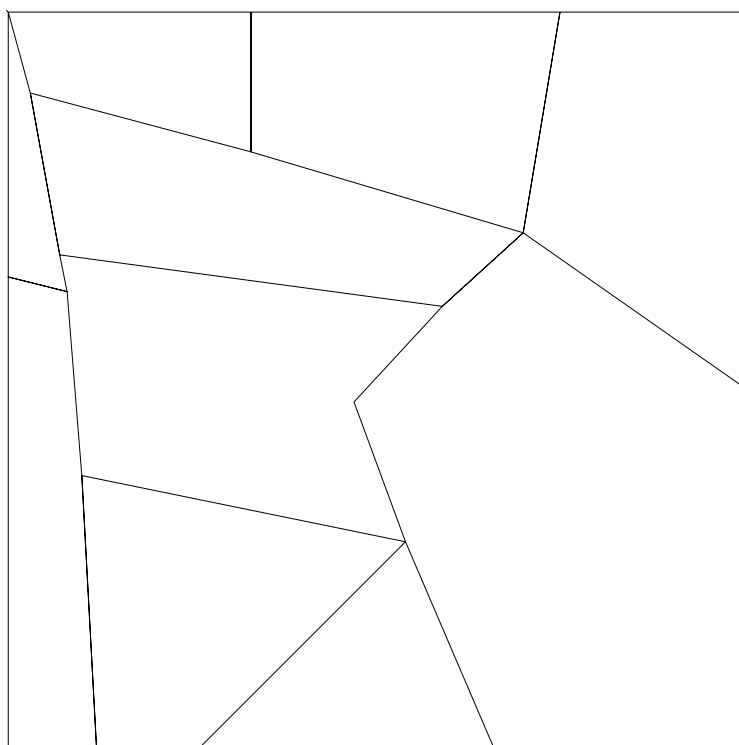


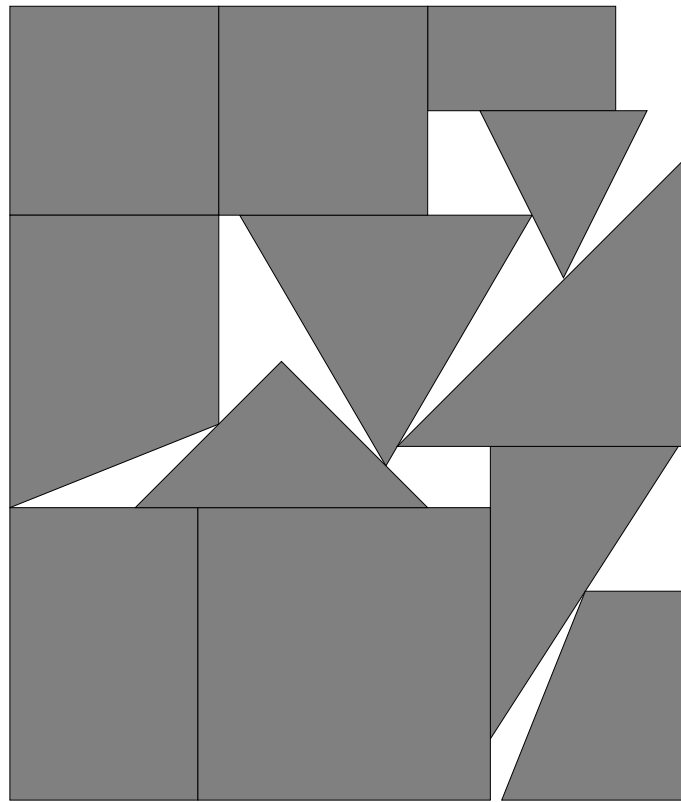
(a) Padrão de corte *DIGHE1*.



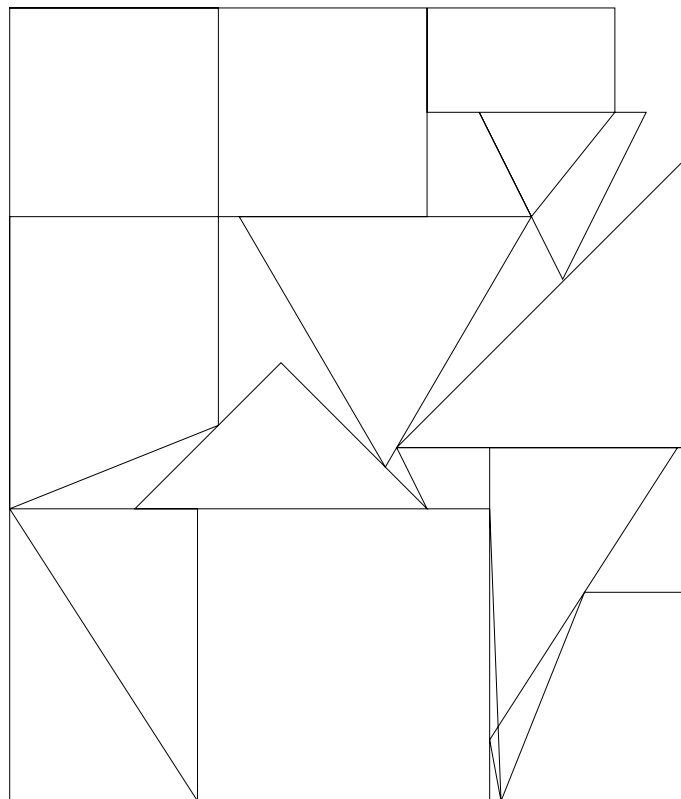
(b) Percurso de corte para *DIGHE1*.

Figura A.14: Instância *DIGHE1*.

(a) Padrão de corte *DIGHE2*.(b) Percurso de corte para *DIGHE2*.Figura A.15: Instância *DIGHE2*.

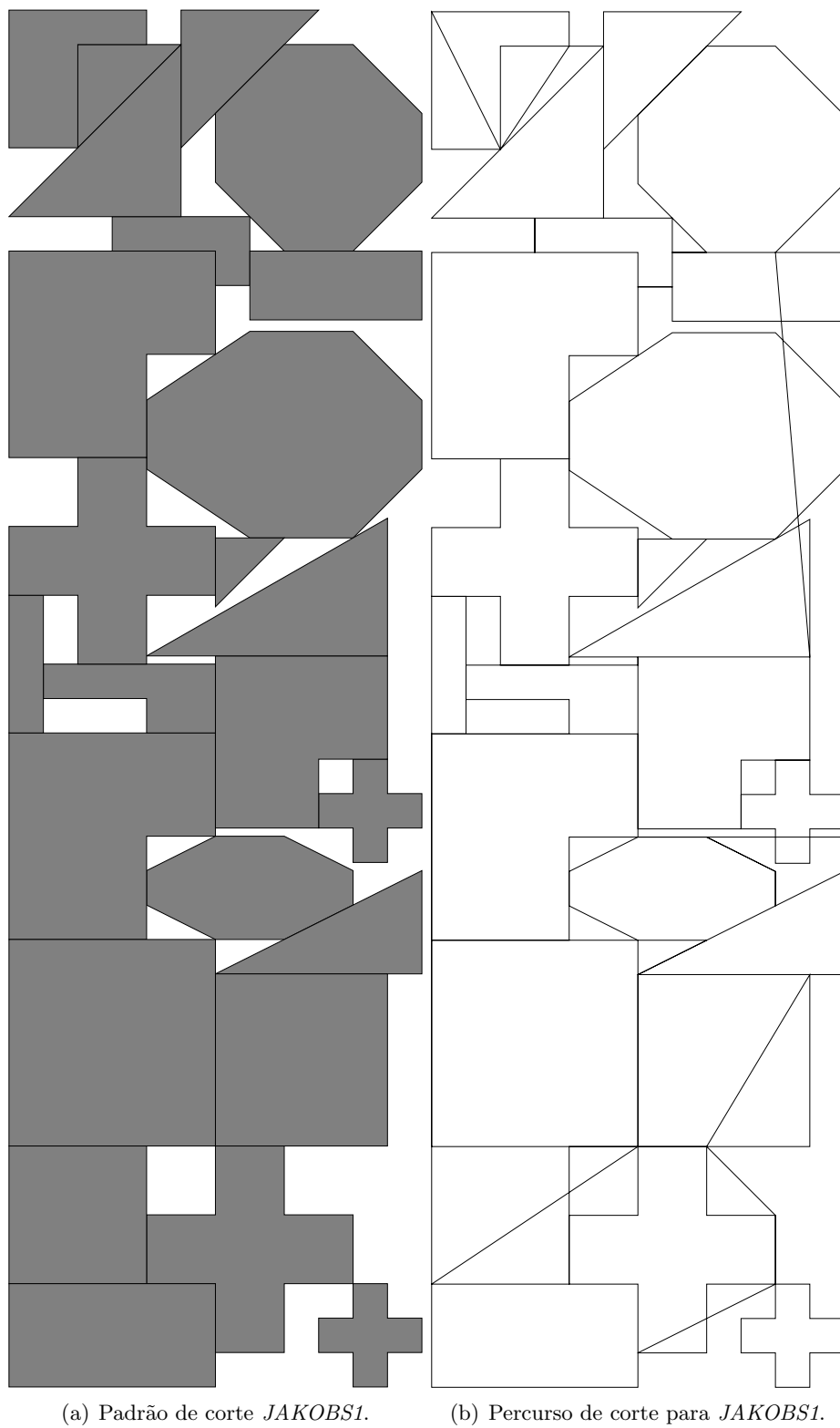


(a) Padrão de corte FU .



(b) Percurso de corte para FU .

Figura A.16: Instância FU .

Figura A.17: Instância *JAKOBS1*.

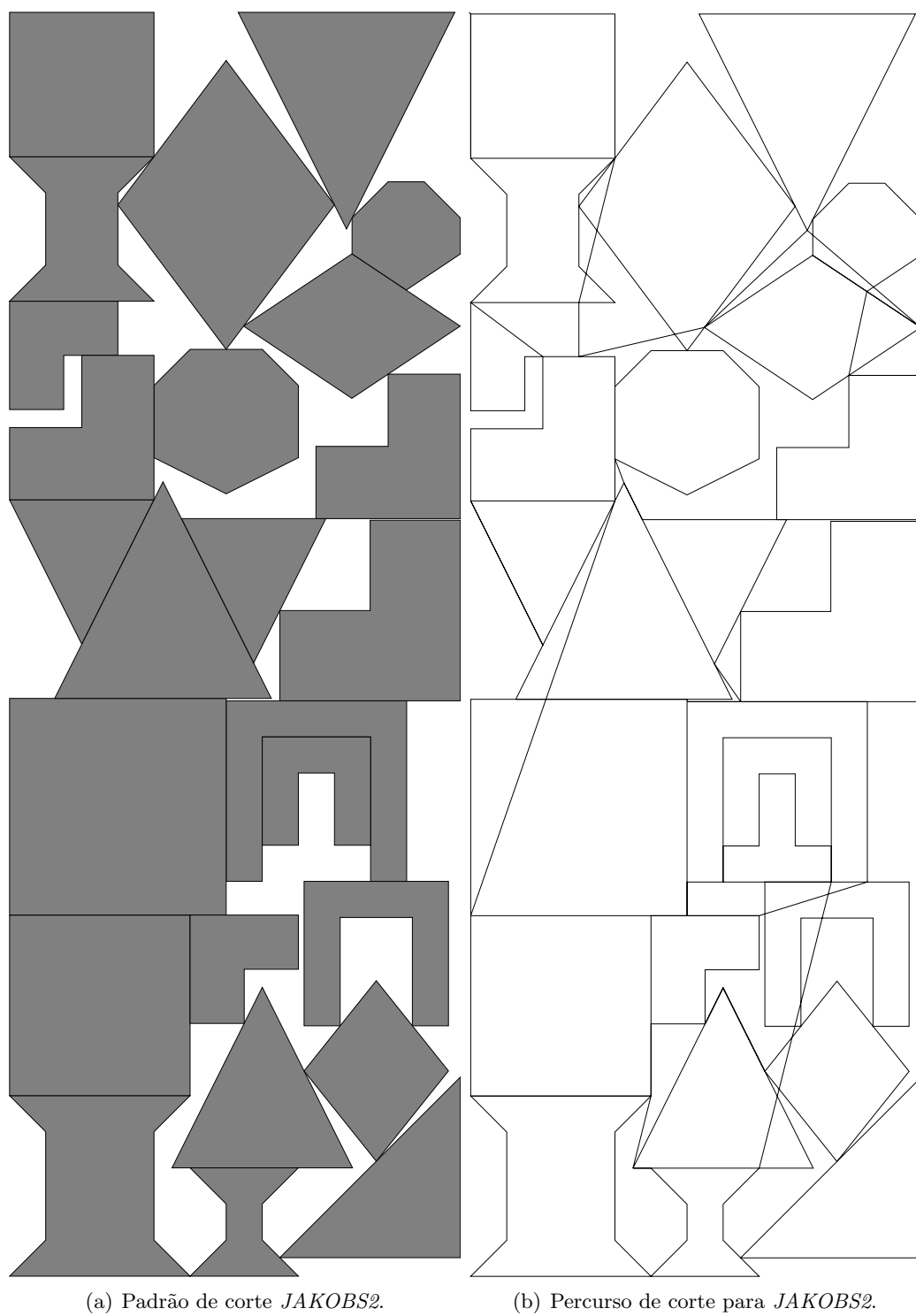
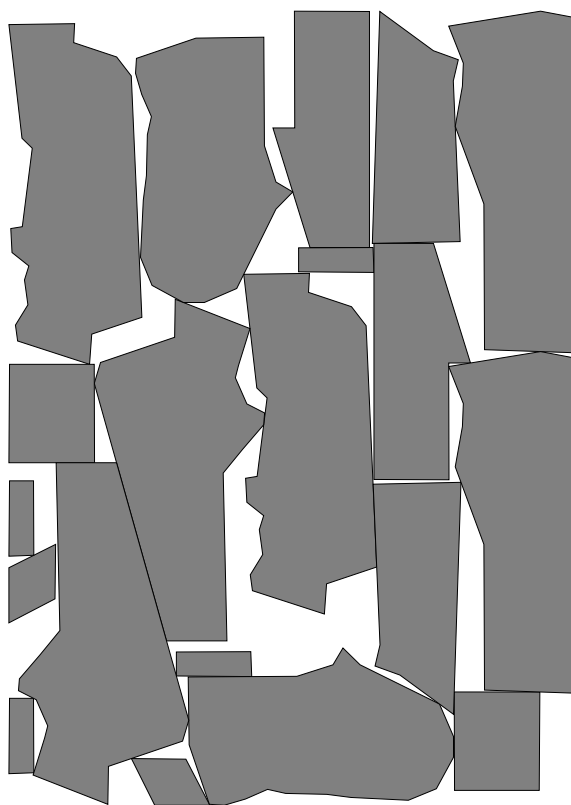
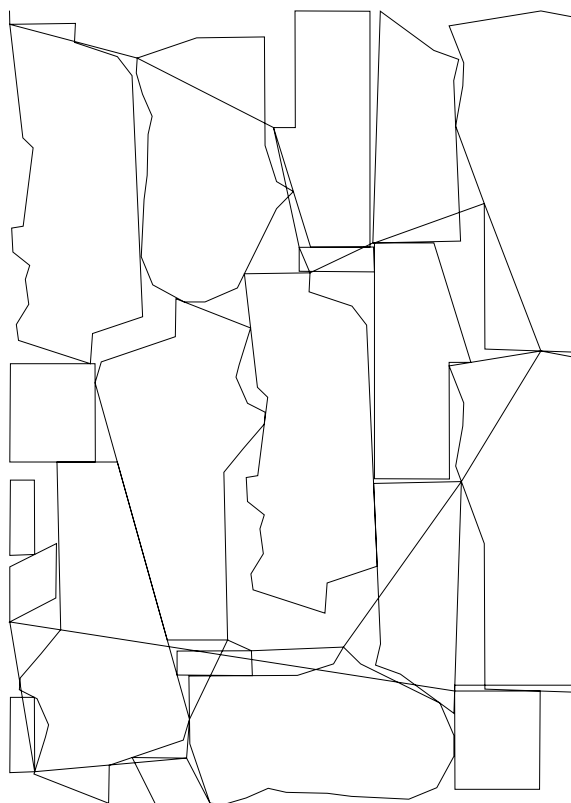
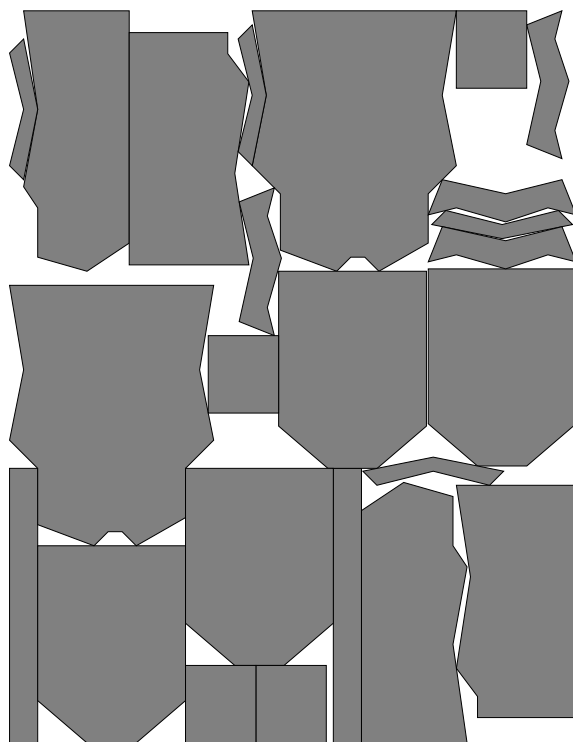
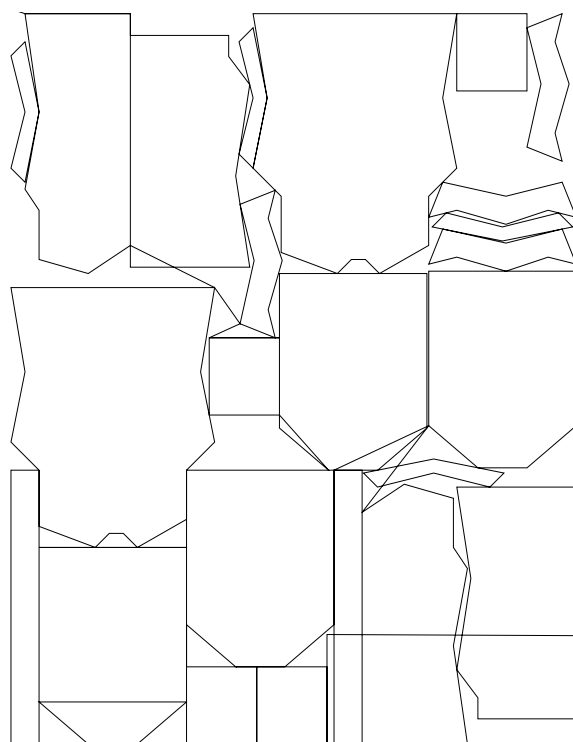


Figura A.18: Instância *JAKOBS2*.

(a) Padrão de corte *MAO*.(b) Percurso de corte para *MAO*.Figura A.19: Instância *MAO*.



(a) Padrão de corte *MARQUES*.



(b) Percurso de corte para *MARQUES*.

Figura A.20: Instância *MARQUES*.

Referências

- [1] J. Aráoz, E. Fernández e C. Zoltan. Privatized rural postman problems. *Comput. Oper. Res.*, 33:3432–3449, 2006.
- [2] E. Benavent, A. Corberán, E. P. nana, I. Plana e J. M. Sanchis. New heuristic algorithms for the windy rural postman problem. *Comput. Oper. Res.*, 32:3111–3128, 2005.
- [3] E. Benavent e D. Soler. The directed rural postman problem with turn penalties. *Transportation Science*, 33:408–418, 1999.
- [4] N. L. Biggs, E. K. Lloyd e R. J. Wilson. *Graph Theory 1736-1936*. Oxford University Press, USA, 1999. ISBN 0198539169.
- [5] A. Corberán, R. Martí, E. Martínez e D. Soler. The rural postman problem on mixed graphs with turn penalties. *Computers & Operations Research*, 29:887–903, 2002.
- [6] J. Edmonds e E. L. Johnson. Matching, euler tours and the chinese postman. *Mathematical Programming*, 5:88–124, 1973.
- [7] J. Egeblad, B. K. Nielsen e A. Odgaard. Fast neighborhood search for two- and three-dimensional nesting problems. *European Journal of Operational Research*, 183:1249–1266, 2007.
- [8] H. A. Eiselt, M. Gendreau e G. Laporte. Arc routing problems, part i: The chinese postman problem. *Operations Research*, 43:231–242, 1995. ISSN 0030364X.
- [9] H. A. Eiselt, M. Gendreau e G. Laporte. Arc routing problems, part ii: The rural postman problem. *Operations Research*, 43:399–414, 1995. ISSN 0030364X.
- [10] L. Euler. Solutio problematis ad geometriam situs pertinentis. *Commentarii Academiae Scientiarum Imperialis Petropolitanae*, Opera Omnia (1), Vol. 7:128–140, 1741.
- [11] E. Fernández, O. Meza e J. Aráoz. Solving the prize-collecting rural postman problem. *Research Report DR-2007/12, EIO Departament, Technical University of Catalonia (Spain)*, 2007.
- [12] L. R. Ford e D. R. Fulkerson. *Flows in Networks*. Princeton Univ Pr, 1962. ISBN 0691079625.
- [13] G. Ghiani, R. Musmanno, G. Paletta e C. Triki. A heuristic for the periodic rural postman problem. *Comput. Oper. Res.*, 32:219–228, 2005.

- [14] A. M. Gomes e J. F. Oliveira. Solving irregular strip packing problems by hybridising simulated annealing and linear programming. *European Journal of Operational Research*, 171:811–829, 2006.
- [15] M. Guan. Graphic programming using odd and even points. *Chinese Math*, 1:273–277, 1962.
- [16] P. D. . A. I. Harvey Deitel. *C++ How to Program*. Prentice Hall, 6 edition, 2007. ISBN 0136152503.
- [17] S. Imahori, M. Kushiya, T. Nakashima e K. Sugihara. Generation of cutter paths for hard material in wire edm. *Journal of Materials Processing Technology*, In Press, Corrected Proof, 2007.
- [18] S. Irnich. A note on postman problems with zigzag service. *INFOR*, 43:33–39, 2005.
- [19] S. Irnich. Solution of real-world postman problems. *European Journal of Operational Research*, 190:52–67, 2008.
- [20] J. Lenstra e A. R. Kan. On general routing problems. *Networks*, 6:273–280, 1976.
- [21] R. B. Martins, M. A. Carravilla e C. Ribeiro. Solving combinatorial problems: An xml-based software development infrastructure. 2006.
- [22] L. M. Moreira. *Determinação de Percursos de Corte numa Fábrica de Ferramentas de Alta Precisão*. Tese de mestrado, Escola de Gestão do Porto da Universidade do Porto, 2001.
- [23] L. M. Moreira, J. F. Oliveira, A. M. Gomes e J. S. Ferreira. Heuristics for a dynamic rural postman problem. *Computers & Operations Research*, 34:3281–3294, 2007.
- [24] J. F. Oliveira. *Problemas de Posicionamento de Figuras Irregulares - Uma perspectiva de optimização*. Tese de doutoramento, Faculdade de Engenharia da Universidade do Porto, 1995.