Análise de Alguns Problemas Geométricos em Polígonos Ortogonais

Ana Paula Tomás

Technical Report Series: DCC-04-03

Departamento de Ciência de Computadores – Faculdade de Ciências

&

Laboratório de Inteligência Artificial e Ciência de Computadores

---

Universidade do Porto

Rua do Campo Alegre, 823 4150 Porto, Portugal

Tel: +351+2+6078830 – Fax: +351+2+6003654

http://www.ncc.up.pt/fcup/DCC/Pubs/treports.html

Análise de Alguns Problemas Geométricos em Polígonos Ortogonais

Ana Paula Tomás*
DCC & LIACC, Univ. do Porto
R. do Campo Alegre, 823
4150-180 Porto, Portugal
apt@ncc.up.pt

Junho 2004

Resumo

Neste trabalho são propostos métodos para resolver alguns problemas geométricos em polígonos ortogonais. É motivado pelo desenvolvimento duma implementação do algoritmo de minimização do número de vigilantes proposto por Tomás, Bajuelos e Marques em [14]. Trata questões relativas à determinação de regiões de visibilidade de vértices e construção de partições.

1 Introdução

Em [14], Tomás, Bajuelos e Marques propuseram um método para minimização do número de guardas necessários para vigiar um dado polígono simples supondo que os guardas se localizarão em vértices do polígono. Para a sua avaliação experimental, F. Marques desenvolveu uma implementação protótipo para polígonos simples ortogonais, o que requereu a análise dalguns sub-problemas do problema considerado, incluindo: a construção da partição $\Pi_{\text{r-cut}}$ dum dado polígono ortogonal em rectângulos (designados por r-peças) obtida por extensão das arestas incidentes em vértices reflexos; a determinação da região de visibilidade de cada vértice; a construção da partição induzida pelas secções de visibilidade para cada r-peça.

Neste relatório descrevem-se as ideias principais dos algoritmos propostos para tratar estes sub-problemas e que serviram de fundamento à implementação desenvolvida por Marques [11]. Serão usadas a notação e nomenclatura introduzidas em [14].

2 Construção da Partição $\Pi_{\text{r-cut}}$

A Fig. 1 apresenta a partição $\Pi_{\text{r-cut}}$ dum dado polígono ortogonal. O método que propomos para efectuar tal partição usa uma técnica fundamental em Geometria Computacional – o varrimento – que neste caso, será horizontal e vertical. No varrimento horizontal encontram-se os pontos de intersecção dos prolongamentos das arestas horizontais com a fronteira do polígono. Marcam-se esses pontos nas arestas verticais e efectua-se, em seguida, um varrimento vertical para cortar as r-peças.

*Trabalho parcialmente financiando pelo LIACC através do *Programa de Financiamento Plurianual da Fundação para a Ciência e Tecnologia* e do *Programa POSI*.

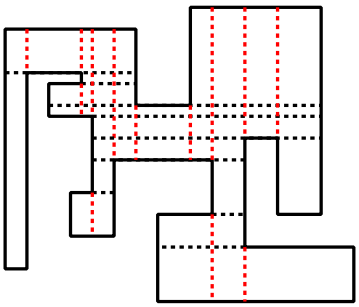

Figura 1: Partição $\Pi_{\text{r-cut}}$ dum polígono ortogonal.

Para simplificar a descrição das ideias centrais do método, vamos assumir que os polígonos estão em posição geral, ou seja não têm arestas colineares, como o polígono dado na Fig. 1.

2.1 Descrição dum algoritmo para polígonos em posição geral

Supomos que a sequência de arestas horizontais é ordenada de forma que a primeira seja o topo e a última a base do polígono ortogonal. À semelhança dos métodos de varrimento será necessário manter e alterar convenientemente o *estado da linha de varrimento* em cada *evento*. Durante o varrimento horizontal, cada aresta horizontal do polígono determina um evento. A Fig. 2 mostra os quatro primeiros estados da linha de varrimento horizontal para o exemplo considerado.

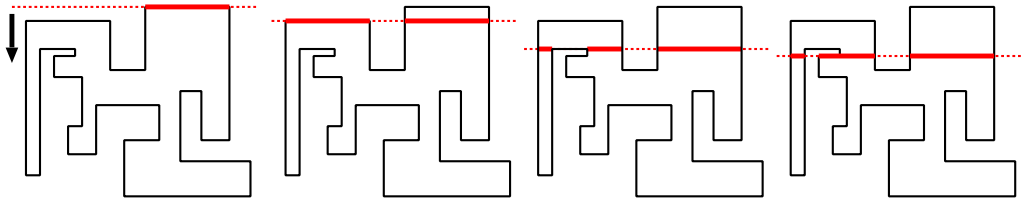

Figura 2: Varrimento horizontal: sequência de estados da linha de varrimento.

Atendendo à geometria do polígono, teremos, em cada evento, uma das situações representadas esquematicamente na Fig. 3, onde a aresta $((a, y), (b, y))$ que determina o evento está identificada pelas abcissas dos seus pontos extremos. O estado da linha de varrimento é uma sequência ordenada de segmentos de recta $(A_1, B_1), \ldots, (A_f, B_f)$, com $f \geq 0$. Supõe-se que os valores dos extremos crescem no sentido do varrimento. Sendo (A_i, B_i) o segmento que satisfaz $B_{i-1} < a \leq B_i$, a alteração de estado em cada um dos casos pode ser traduzida formalmente do modo seguinte, onde γ_1 e γ_2 denotam os restantes segmentos do estado actual.

$$
\begin{array}{lcll}
\gamma_1\ (A_i, B_i)\gamma_2 & \Rightarrow & \gamma_1\ (A_i, a), (b, B_i)\ \gamma_2 & \text{sse } A_i < a \wedge B_i > b \\
\gamma_1\ (A_i, B_i)\gamma_2 & \Rightarrow & \gamma_1\ (a, B_i)\ \gamma_2 & \text{sse } b = A_i \\
\gamma_1\ (A_i, B_i)\ \gamma_2 & \Rightarrow & \gamma_1\ (A_i, b)\ \gamma_2 & \text{sse } a = B_i \wedge (b < A_{i+1} \vee i = f) \\
\gamma_1\ (A_i, B_i), (A_{i+1}, B_{i+1})\ \gamma_2 & \Rightarrow & \gamma_1\ (A_i, B_{i+1})\ \gamma_2 & \text{sse } a = B_i \wedge b = A_{i+1} \\
\gamma_1\ (A_i, B_i)\ \gamma_2 & \Rightarrow & \gamma_1\ (a, b), (A_i, B_i)\ \gamma_2 & \text{sse } b < A_i \\
\gamma_1\ (A_i, B_i)\ \gamma_2 & \Rightarrow & \gamma_1\ (b, B_i)\ \gamma_2 & \text{sse } a = A_i \wedge b < B_i \\
\gamma_1\ (A_i, B_i)\ \gamma_2 & \Rightarrow & \gamma_1\ (A_i, a)\ \gamma_2 & \text{sse } a < A_i \wedge b = B_i \\
\gamma_1\ (A_i, B_i)\ \gamma_2 & \Rightarrow & \gamma_1\ \gamma_2 & \text{sse } a = A_i \wedge b = B_i
\end{array}
$$

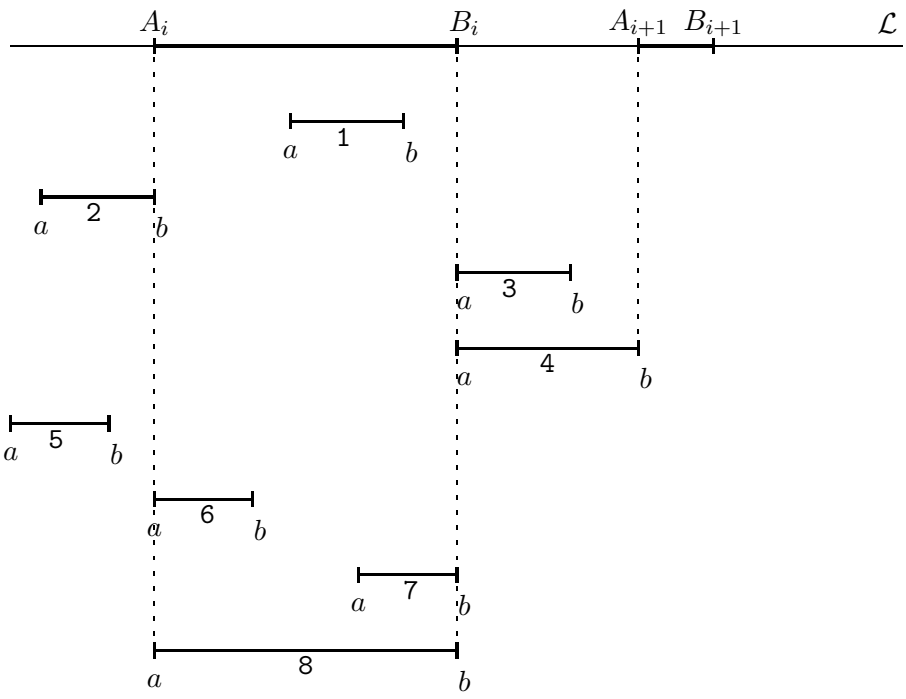

Figura 3: Oito casos possíveis no evento determinado por uma aresta. Resta apenas o caso `9`, em que (a, b) se situa à direita do último segmento marcado na linha de varrimento.

Se $b \leq A_1$ ou $a > B_f$, o segmento (a, b) será inserido, respectivamente, no início e no fim da sequência.

2.1.1 Intersecção das cordas horizontais com a fronteira

Em todos os casos representados na Fig. 3, com excepção dos casos `5` e `8`, a aresta horizontal $((a, y), (b, y))$ dará origem a uma ou duas cordas da partição horizontal. Em cada evento, os pontos de intersecção dessas cordas com a fronteira do polígono são definidos do modo seguinte, não sendo obtidos pontos de intersecção nos casos não indicados.

$(A_i, y), (B_i, y)$	sse $A_i < a \wedge B_i > b$
(B_i, y)	sse $b = A_i$
(A_i, y)	sse $a = B_i \wedge (b < A_{i+1} \vee i = f)$
$(A_i, y), (B_{i+1}, y)$	sse $a = B_i \wedge A = B_{i+1}$
(B_i, y)	sse $a = A_i \wedge b < B_i$
(A_i, y)	sse $a < A_i \wedge b = B_i$

Para ver que assim é, observe-se as Figs. 2 e 4. Durante o deslocamento da linha de varrimento horizontal, os extremos dos segmentos que definem o estado de varrimento estão sempre sobre arestas verticais do polígono. Nos casos `1`, `6`, `7` e `8` a aresta delimita inferiormente uma região do polígono enquanto que nos restantes é topo duma região.

Uma implementação. Apresentamos a seguir uma implementação desta primeira fase do método, em Prolog. Na chamada de `horizontal_sweep/3`, o primeiro argumento é uma

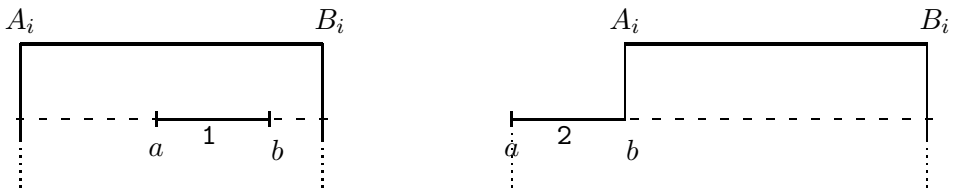

Figura 4: Análise de eventos para determinação de pontos de intersecção resultantes da extensão das arestas incidentes em vértices reflexos (para o interior do polígono).

lista de termos `Y:[A,B]`, que representam as arestas horizontais ordenadas por valor de `Y`. O segundo termo deverá estar instanciado com `[]`. O resultado `IntPoints` é a lista de pontos de intersecção, os quais são representados por termos `s(X,Y)`.

```
horizontal_sweep([],[],[]).
horizontal_sweep([Y:[A,B]|HEdges],SweepState,IntPoints) :-
   hSweep(SweepState,A,B,Y,IntPoints,RIntPoints,NewSweepState),
   horizontal_sweep(HEdges,NewSweepState,RIntPoints).

hSweep([],A,B,_,RS,RS,[[A,B]]).   %% 9
hSweep([[A,B]|PEs],A,B,_,RS,RS,PEs) :- !.  %% 8
hSweep([[B,Bi]|PEs],A,B,Y,[s(Bi,Y)|RS],RS,[[A,Bi]|PEs]) :- !.  %% 2
hSweep([[A,Bi]|PEs],A,B,Y,[s(Bi,Y)|RS],RS,[[B,Bi]|PEs]) :- !.  %% 6
hSweep([[Ai,A]|PEs],A,B,Y,[s(Ai,Y)|RSi],RS,PEsf) :-
  (PEs = [[B,Bi1]|RPEs] ->
     (RSi = [s(Bi1,Y)|RS], PEsf = [[Ai,Bi1]|RPEs]); %% 4
     (RSi = RS, PEsf = [[Ai,B]|PEs])).              %% 3
hSweep([[Ai,B]|PEs],A,B,Y,[s(Ai,Y)|RS],RS,[[Ai,A]|PEs]) :- !.  %%  7
hSweep([[Ai,Bi]|PEs],A,B,Y,[s(Ai,Y),s(Bi,Y)|RS],RS,[[Ai,A],[B,Bi]|PEs]):-
    Ai < A, Bi > B, !.    %% 1
hSweep([[Ai,Bi]|PEs],A,B,_,RS,RS,[[A,B],[Ai,Bi]|PEs]) :- B < Ai, !. %% 5
hSweep([E|PEs],A,B,Y,Sps,RSps,[E|PEsf]) :- %% procura [Ai,Bi]
   hSweep(PEs,A,B,Y,Sps,RSps,PEsf).
```

Passamos a descrever a segunda fase do método, em que usaremos os pontos de intersecção agora calculados para detectar a formação de r-peças durante o varrimento vertical.

2.1.2 Varrimento vertical para determinação das r-peças

Associamos a cada aresta vertical $((x,a),(x,b))$, o conjunto S_{ab} dos pontos de intersecção das cordas horizontais com o interior dessa aresta, ordenados por valor de y. As Figs. 5 e 6 ilustram o papel desses conjuntos na formação de r-peças durante o varrimento vertical.

No exemplo dado à esquerda na Fig. 6, pode ver-se uma correspondência entre alguns pontos assinalados no segmento (A_i, B_i) do estado de varrimento e os assinalados na aresta. A existência dessa correspondência é uma propriedade geométrica deste problema. Esta propriedade resulta trivialmente da definição dos pontos de intersecção e é crucial para a correcção do método de corte. Para definir formalmente este método, designemos por S_i o

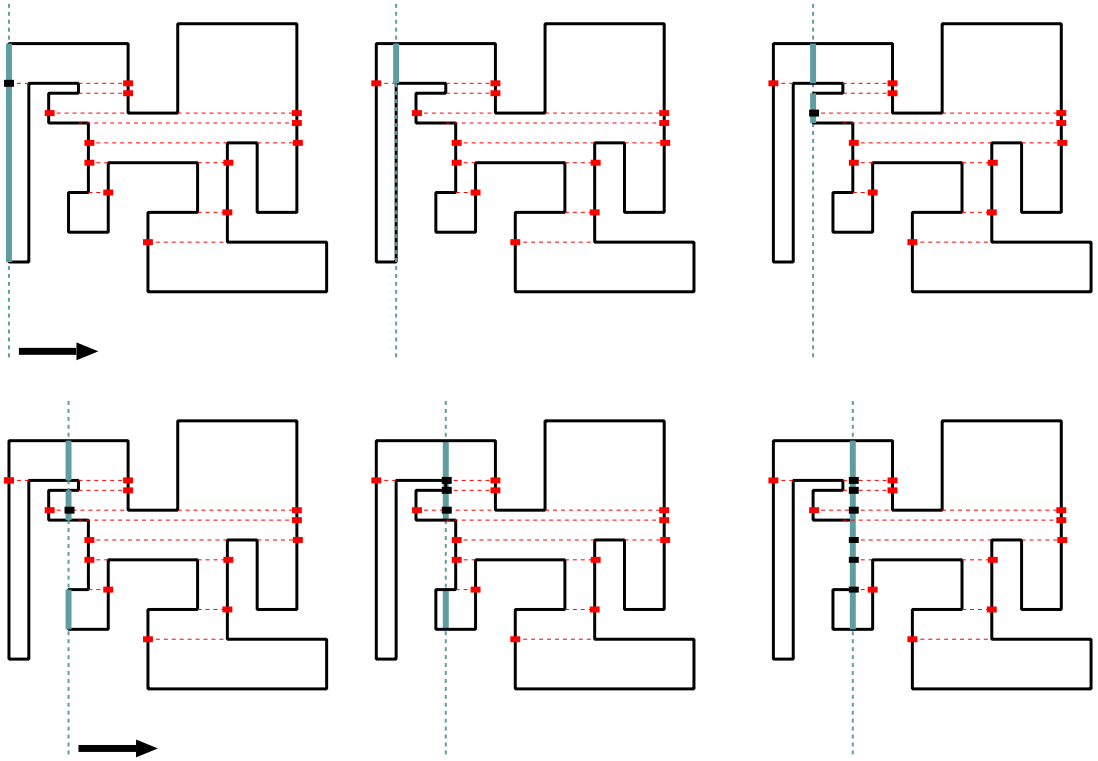

Figura 5: Sequência de estados durante varrimento vertical. Note-se que os pontos marcados em segmentos desses estados assinalam intersecções com cordas horizontais.

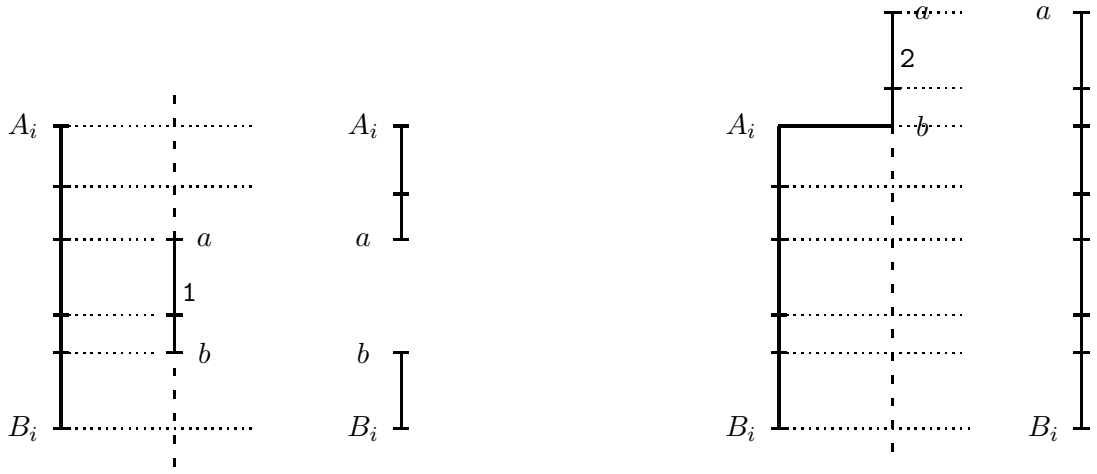

Figura 6: Varrimento vertical: corte de r-peças e propagação das marcas de intersecção com cordas horizontais. No evento representado à esquerda formam-se cinco r-peças e ficam três abertas para o próximo evento. À direita, formam-se cinco r-peças e ficam sete abertas.

conjunto de pontos marcados no interior do segmento (A_i, B_i). Recorde-se que acima representámos por S_{ab} o conjunto de pontos marcados no interior da aresta vertical $((x,a),(x,b))$. As alterações de estado durante o varrimento vertical são traduzidas do modo seguinte, onde ϵ significa que o segmento foi retirado.

(A_i, B_i, S_i)	$\Rightarrow$	$(A_i, a, S_i \cap]A_i, a[), (b, B_i, S_i \cap]b, B_i[)$	caso `1`
(A_i, B_i, S_i)	$\Rightarrow$	$(a, B_i, S_i \cup S_{ab} \cup \{b\})$	caso `2`
(A_i, B_i, S_i)	$\Rightarrow$	$(A_i, b, S_i \cup S_{ab} \cup \{a\})$	caso `3`
$(A_i, B_i, S_i), (A_{i+1}, B_{i+1}, S_{i+1})$	$\Rightarrow$	$(A_i, B_{i+1}, S_i \cup S_{i+1} \cup S_{ab} \cup \{a, b\})$	caso `4`
(A_i, B_i, S_i)	$\Rightarrow$	$(a, b, S_{ab}), (A_i, B_i, S_i)$	caso `5`
(A_i, B_i, S_i)	$\Rightarrow$	$(b, B_i, S_i \cap]b, B_i[)$	caso `6`
(A_i, B_i, S_i)	$\Rightarrow$	$(A_i, a, S_i \cap]A_i, a[)$	caso `7`
(A_i, B_i, S_i)	$\Rightarrow$	ϵ	caso `8`

A condição em cada caso é idêntica à apresentada para o varrimento horizontal, tendo sido omitida por questões de espaço. Nos restantes casos a aresta é inserida no príncipio ou fim do estado, criando um novo segmento isolado (a, b, S_{ab}), como no caso `5`.

Analisamos agora a formação de r-peças em cada evento. Para isso calculamos os pontos de intersecção das cordas verticais com as cordas horizontais que são pontos interiores do polígono ou de arestas horizontais. Estes pontos são os vértices das r-peças situados no interior do polígono ou no interior de arestas horizontais. O conjunto de pontos que se obtem em cada evento e o número de r-peças que se formam são definidos por:

$\{(x,y) \mid y \in (S_i \cup \{A_i, B_i\}) \setminus (S_{ab} \cup \{a, b\})\}$	$\lvert S_i \rvert + 1$	caso `1`
$\{(x,y) \mid y \in S_i \cup \{B_i\}\}$	$\lvert S_i \rvert + 1$	caso `2`
$\{(x,y) \mid y \in S_i \cup \{A_i\}\}$	$\lvert S_i \rvert + 1$	caso `3`
$\{(x,y) \mid y \in S_i \cup S_{i+1} \cup \{A_i, B_{i+1}\}\}$	$\lvert S_i \rvert + \lvert S_{i+1} \rvert + 2$	caso `4`
$\emptyset$	0	caso `5`
$\{(x,y) \mid y \in (S_i \cup \{B_i\}) \setminus (S_{ab} \cup \{b\})\}$	$\lvert S_i \rvert + 1$	caso `6`
$\{(x,y) \mid y \in (S_i \cup \{A_i\}) \setminus (S_{ab} \cup \{a\})\}$	$\lvert S_i \rvert + 1$	caso `7`
$\emptyset$	$\lvert S_i \rvert + 1$	caso `8`

Uma implementação. À semelhança do que fizemos na secção anterior, apresentamos a seguir uma implementação da segunda fase do método, em Prolog. Na chamada de `vertical_sweep/4`, o primeiro argumento é uma lista de termos `X:e([A,B,Sab])` que representam as arestas verticais ordenadas por valor de `X`. O termo `Sab` é a lista de pontos "marcados" sobre a aresta na primeira fase (ou seja, S_{ab}). O segundo parâmetro deverá estar instanciado com `[]`. Os dois últimos parâmetros são de saída: `NRpieces` terá o número de r-peças formadas e `SteinP` é a lista de pontos de intersecção interiores ao polígono ou a arestas horizontais. Estes pontos são representados por termos `s(X,Y)` e, como vimos, são vértices das r-peças situados no interior do polígono ou de arestas horizontais.

```
vertical_sweep([],[],0,[]).
vertical_sweep([X:e(A,B,Sab)|VEdges],SweepState,NRpieces,SteinP) :-
   vSweep(SweepState,A,B,Sab,X,SteinP,RSteinP,NewState,NRpiecesI),
   vertical_sweep(VEdges,NewState,NRpiecesR,RSteinP),
   NRpieces is NRpiecesI+NRpiecesR.
```

```
vSweep([],A,B,Sab,_,RS,RS,[e(A,B,Sab)],0).   %% 9
vSweep([e(A,B,Sab)|PEs],A,B,Sab,_,RS,RS,PEs,Np) :- !,  %% 8
    length(Si,Npi), Np is Npi+1.
vSweep([e(B,Bi,Si)|PEs],A,B,Sab,X,StP,RStP,[e(A,Bi,Sf)|PEs],Np) :- !, %% 2
    steinpts(Si,X,StP,[s(X,Bi)|RStP]),
    append(Sab,[B|Si],Sf),
    length(Si,Npi), Np is Npi+1.
vSweep([e(A,Bi,Si)|PEs],A,B,_,X,StP,RStP,[e(B,Bi,Sf)|PEs],Np) :- !, %% 6
    split_steinpts(Si,B,Sf,_),
    steinpts(Sf,X,StP,[s(X,Bi)|RStP]),
    length(Si,Npi), Np is Npi+1.
vSweep([e(Ai,A,Si)|PEs],A,B,Sab,X,StP,RStP,PEsf,Np) :- !,
    steinpts(Si,X,StP,[s(X,Ai)|RStI],
    append(Si,[A|Sab],Siab),
    length(Si,Npi),
    ( PEs = [e(B,Bi1,Si1)|RPEs] -> %% 4
     ( steinpts(Si1,X,RStI,[s(X,Bi1)|RStP]),
       append(Siab,[B|Si1],Sf), length(Si1,Npi1),
       Np is Npi+Npi1+2,  PEsf = [e(Ai,Bi1,Sf)|RPEs]);
     ( RStP = RStI, Sf = Siab, Np is Npi+1, %% 3
       PEsf = [e(Ai,B,Sf)|PEs])).
vSweep([e(Ai,B,Si)|PEs],A,B,_,X,StP,RStP,[e(Ai,A,Sf)|PEs],Np) :- !, %% 7
    split_steinpts(Si,A,_,Sf),
    steinpts(Sf,X,StP,[s(X,Ai)|RStP]),
    length(Si,Npi), Np is Npi+1.
vSweep([e(Ai,Bi,Si)|PEs],A,B,_Sab,X,StP,RStP,PEsf,Np) :- Ai < A, Bi > B, !,
    %% 1
    PEsf = [e(Ai,A,S1),e(B,Bi,S2)|PEs],
    split_steinpts(Si,A,RSi,S1),split_steinpts(RSi,B,S2,_),
    steinpts(S1,X,StP,[s(X,Ai)|RStPi]),
    steinpts(S2,X,RStPi,[s(X,Bi)|RStP]),
    length(Si,Npi), Np is Npi+1.
vSweep([e(Ai,Bi,Si)|PEs],A,B,Sab,_,StP,StP,PEsf,0) :- B < Ai, !, %% 5
    PEsf = [e(A,B,Sab),e(Ai,Bi,Si)|PEs].
vSweep([E|PEs],A,B,Sab,X,StP,RStP,[E|PEsf],Np) :-  %% procura (Ai,Bi)
    vSweep(PEs,A,B,Sab,X,StP,RStP,PEsf,Np).
```

O predicado `split_steinpts/4`, na chamada `split_steinpts(L,X,LMenorX,LMaiorX)`, determina as listas `LMenorX` e `LMaiorX` dos elementos de `L` que são menores e maiores que `X`, respectivamente. O predicado `steinpts(Ly,X,SteinP,RSteinP)` constroi a lista de termos `s(X,Y)` para os `Y`'s dados em `Ly`. Essa lista `SteinP` é deixada aberta, sendo `RSteinP` o seu resto.

2.2 Construção da Grelha

O método que acábamos de expor calcula apenas os pontos de Steiner da partição $\Pi_{\text{r-cut}}(\mathcal{P})$. Estes pontos são os vértices das r-peças que não são vértices do polígono $\mathcal{P}$. Contudo, para implementar o método de optimização de vigilantes é necessário conhecer a estrutura da

partição, nomeadamente, a relação de adjacência entre as r-peças, os quatro pontos que definem cada r-peça e as peças de que são vértices os vértices de $\mathcal{P}$. Ou seja, é necessário conhecer a *grelha* que define a partição, como mostra a Fig. 7.

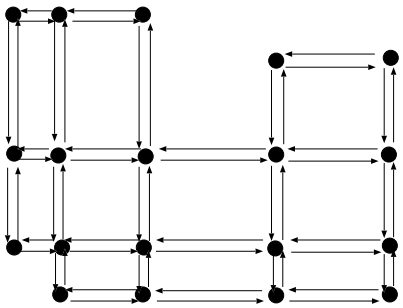

Figura 7: Grelha traduzindo a adjacência entre peças da partição $\Pi_{\text{r-cut}}$. Cada peça pode ser identificada, por exemplo, pelo nó da grelha que é seu canto noroeste.

Vamos ver como definir estruturas de dados para a representar a grelha e o estado de varrimento, que facilitem a sua construção durante o varrimento. Nesta secção apresentamos excertos dum programa (em linguagem C) que constrói a grelha tendo por base o método anterior.

2.2.1 Estruturas de dados

Começamos por definir a estrutura `gridnode` que representará um nó da grelha.

```
typedef struct gridnode{
  int point[2];
  struct piece *idrpiece;
  struct gridnode *adjs[4];
} GNODE;
```

Cada nó contem apontadores para os seus quatro adjacentes (a Norte, Sul, Este e Oeste) e para a r-peça de que é canto noroeste (que será `NULL` se esta não existir). A estrutura `piece` guardará outra informação sobre tal peça, mas não os seus vértices nem dados sobre peças adjacentes. De facto, estes dados podem ser obtidos por análise da grelha.

Antes de introduzirmos estruturas para representar o estado de varrimento, analisemos as Figs. 8 e 9, as quais mostram as alterações que seriam efectuadas na grelha se introduzissemos os pontos de Steiner, à medida que vão sendo criados. A grelha inicial é análoga ao polígono, embora cada nó passe a ser um `gridnode`.

Para representar cada segmento (A_i, B_i) do estado de varrimento usamos a estrutura `segment`, com `start` e `final` a apontarem os nós da grelha que correspondem aos extremos A_i e B_i, os quais são nós activos (c.f., Figs. 8 e 9).

```
typedef struct segment{
  GNODE *start, *final;
  struct segment *nxt, *prev;
} SEGMENT;
```

O estado de varrimento será dado por uma lista de segmentos duplamente ligada. Por isso cada segmento contém, em `nxt` e `prev`, apontadores para os segmentos que o seguem e precedem.

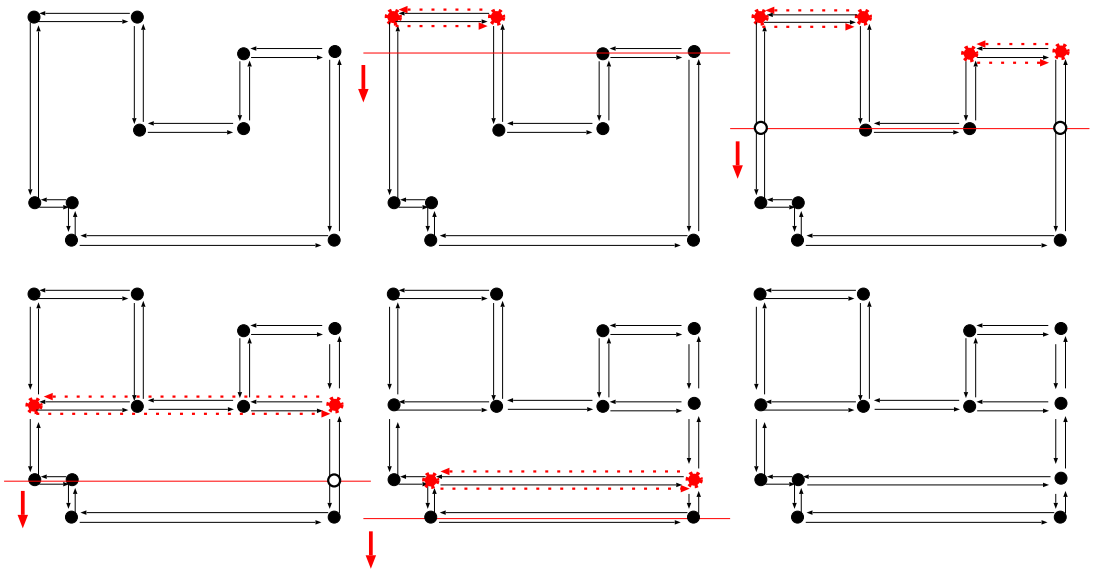

Figura 8: Alteração da grelha inicial durante o varrimento horizontal. Os novos pontos de Steiner são denotados por ○ e os nós activos por ⋆. Cada nó novo é ligado ao nó activo situado na aresta vertical que contem esse novo nó, passando a ser um nó da grelha. A sua ligação à grelha efectua-se por partição das ligações entre esse nó activo e o seu adjacente segundo a direcção do novo nó (que está de facto sempre a Sul), como se pode ver.

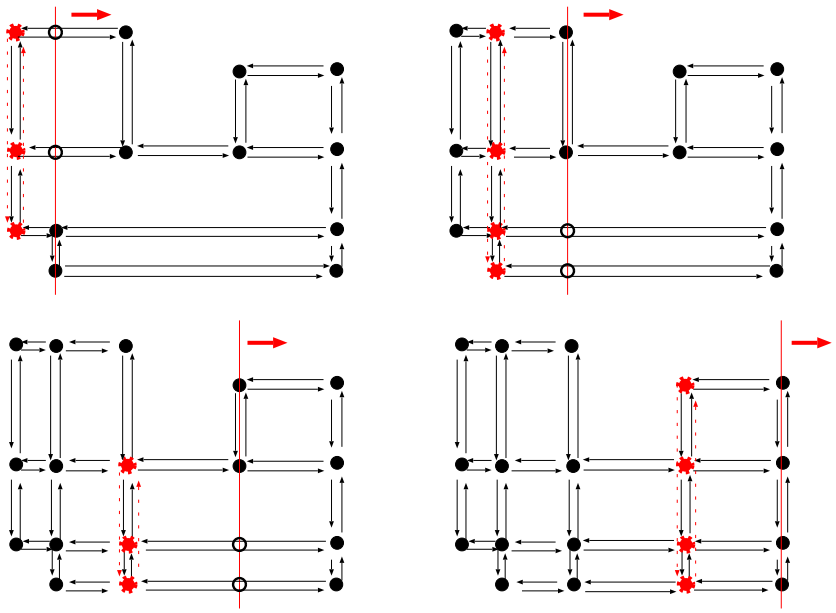

Figura 9: Modificação da grelha resultante da partição horizontal no varrimento vertical. Cada nó novo é ligado ao nó activo situado na corda horizontal a que pertence esse novo nó, partindo as ligações entre esse activo e o seu adjacente a Este. É ligado também ao nó que, no sentido de geração de nós, o precede na corda vertical que prolonga a aresta que determinou o evento. Tal implica a partição das ligações entre esse nó e o que o segue nessa direcção.

Com esta representação para os segmentos é possível aceder de forma natural aos pontos de Steiner que são relevantes num dado evento. De facto, esses pontos podem ser obtidos usando a relação de adjacência entre nós, construída até ao evento. Evita-se assim introduzir uma estrutura auxiliar para representar S_i e simultaneamente fica-se com acesso natural aos nós da grelha que serão relevantes para a ligação dos novos nós.

Note-se que a representação do estado de varrimento por uma estrutura linear não permite que as operações de localização, inserção e remoção de segmentos sejam efectuadas em tempo óptimo. É conhecido que a sua representação por uma árvore bem equilibrada (como, por exemplo, árvore 2-3) permitiria realizar essas operações em tempo logarítmico no número de segmentos [12]. No entanto, esse desempenho óptimo não é essencial para a implementação do método descrito em [14].

Para representar a lista de arestas que define a sequência de eventos, usaremos ainda uma lista de elementos do tipo `SEGMENT`. Recorde-se que cada aresta é definida por dois nós da grelha que são também vértices do polígono. A aresta será dada por um segmento cujas extremidades são os dois nós da grelha que a definem. Deste modo, consegue-se aceder de forma natural aos pontos de Steiner que estão sobre a aresta, evitando-se a utilização da estrutura auxiliar S_{ab}.

Criação de novos nós e de novos segmentos. Apresentamos a seguir o código das funções que efectuam a criação de nós da grelha e de segmentos.

```
GNODE *new_gridnode(int x, int y) {
  ROSE pos;
  GNODE *aux = (GNODE *) malloc(sizeof(GNODE));
  POINT_X(aux) = x; POINT_Y(aux) = y;
  for (pos=0; pos<4; pos++) ADJ(aux,pos) = NULL;
  IDRPIECE(aux) = NULL;
  return aux;
}

SEGMENT *new_segment(GNODE *s, GNODE *f, SEGMENT *prev, SEGMENT *nxt) {
  SEGMENT *aux = (SEGMENT *) malloc(sizeof(SEGMENT));
  FINAL(aux) = f; START(aux) = s;
  PREV(aux) = prev; NXT(aux) = nxt;
  return aux;
}
```

As macros usadas para aceder às estruturas são assim definidas.

```
#define POINT_X(V) ((V)->point[0])
#define POINT_Y(V) ((V)->point[1])
#define START(S) ((S)->start)
#define FINAL(S) ((S)->final)
#define NXT(P) ((P)->nxt)
#define PREV(P) ((P)->prev)
#define IDRPIECE(S) ((S)->idrpiece)
#define ADJ(V,D) ((V)->adjs[(D)])
#define COORD(P,C) ((P)->point[(C)])
```

2.2.2 Alteração da grelha durante varrimento

É possível uniformizar o tratamento do varrimento horizontal e vertical. Ambos serão implementados pelo mesmo procedimento, ainda que naturalmente invocado com argumentos diferentes. Tais argumentos correspondem às diferenças de orientação que obrigam a alguma especificidade de tratamento durante a criação de pontos de Steiner.

Para o varrimento horizontal, a sequência de eventos é definida pela lista de arestas horizontais ordenadas de cima para baixo (isto é, no sentido Norte-Sul). No varrimento vertical, a sequência é definida pelas arestas verticais ordenadas da esquerda para a direita (sentido Oeste-Este). A função principal `GNODE *pi_rcut(VERTEX *polygon, int n)` chamará o procedimento `sweep`, que implementa o varrimento, primeiro com orientação `SOUTH` e depois `EAST`. O seu código em C é o seguinte, com `initgrid` a efectuar a inicialização da grelha.

```
GNODE *pi_rcut(VERTEX *polygon, int n) {
  GNODE *grid = initgrid(VERTEX *polygon, int n);
  sweep(edges(polygon,n,1),SOUTH);
  sweep(edges(polygon,n,0),EAST);
  return grid;
}
```

A função `edges`, declarada como `SEGMENT *edges(VERTEX *polygon, int n, int coord)`, retorna a lista ordenada de arestas horizontais se `coord` for 1 e verticais se for 0. Cada aresta é representada pelos dois nós da grelha que a definem. Para os pontos cardeais e os seus simétricos optou-se pelas seguintes definições,

```
typedef enum{NORTH,SOUTH,EAST,WEST} ROSE;

ROSE OppositeDir[4] = {SOUTH,NORTH,WEST,EAST};
#define OPPOSITEDIR(D) (OppositeDir[(D)])
#define PERPENDIR(D) (((D)==EAST || (D) == WEST)? SOUTH : EAST)
```

importando clarificar que `PERPENDIR(D)` indica o sentido de deslocamento da linha de varrimento. O procedimento `sweep` é dado por

```
void sweep(SEGMENT *edges,ROSE perpd)
{ int event, c=INDEXCOORD(perpd);
  SEGMENT *aibi, *sweepst = new_segment(START(edges),FINAL(edges),NULL,NULL);
  while (!EMPTY((edges = NXT(edges)))) {
    aibi = locate(sweepst,1-c,edges,&event);
    sweepst = update(sweepst,aibi,perpd,event,edges);
  }
}
```

e usa duas novas macros.

```
#define EMPTY(P) ((P)== NULL)
#define INDEXCOORD(D) ((D)==SOUTH?1:0)
```

A última dá o índice da coordenada que define a linha de varrimento: 1 (isto é, y) para o varrimento horizontal e 0 (isto é, x) para o vertical.

Para cada evento, `sweep` começa por localizar o segmento (A_i, B_i) e identificar o tipo de evento (casos `1` a `9`).

```
SEGMENT *locate(SEGMENT *sweepst, int coord, SEGMENT *e, int *event)
{
  int a, b, ai, bi;
  if (EMPTY(sweepst)){
    *event = 5;
    return sweepst;
  }
  a = COORD(START(e),coord); b = COORD(FINAL(e),coord);
  for (; ;) {
    bi = COORD(FINAL(sweepst),coord);
    if (a > bi)
      if (!EMPTY(NXT(sweepst))) sweepst = NXT(sweepst);
      else {
        *event = 9;
         return sweepst; // end sweep-line
      }
    else {
      ai = COORD(START(sweepst),coord);
      if (ai == a) {
        if (bi == b) *event = 8;
        else *event = 6;
      } else if (b == ai) *event = 2;
      else if (bi == a) {
        if (!EMPTY(NXT(sweepst)) && COORD(START(NXT(sweepst)),coord) == b)
          *event = 4;
        else *event = 3;
      } else if (bi == b) *event = 7;
      else if (ai < a && bi > b)
        *event = 1;
      else  *event = 5;
      return  sweepst;
    }
  }
}
```

Recorde-se que se está a considerar um referencial ortonormado em que a coordenada x cresce da esquerda para a direita e y de cima para baixo.

A função `update` actualiza o estado de varrimento e insere na grelha os pontos de Steiner criados pelo evento. Para tal, analisa o valor de `event` e atende também a algumas especificidades dos dois tipos de varrimento. De facto, enquanto que no varrimento horizontal os pontos de Steiner são inseridos na grelha por ligação aos extremos do segmento (A_i, B_i), no varrimento vertical poderão ter que ser ligados a pontos da grelha que estão no interior desse segmento. Pode ser útil observar as Figs. 10 e 11.

Podemos agora apresentar o código da função `update`, no qual `EXTREME` dá o extremo inicial ou final dum segmento, de acordo com o valor duma direcção dada, permitindo uniformizar a implementação dos casos `2` e `3` e dos casos `6` e `7`.

```
#define EXTREME(S,D) (((D)==NORTH || (D)==WEST)? START((S)):FINAL((S)))
```

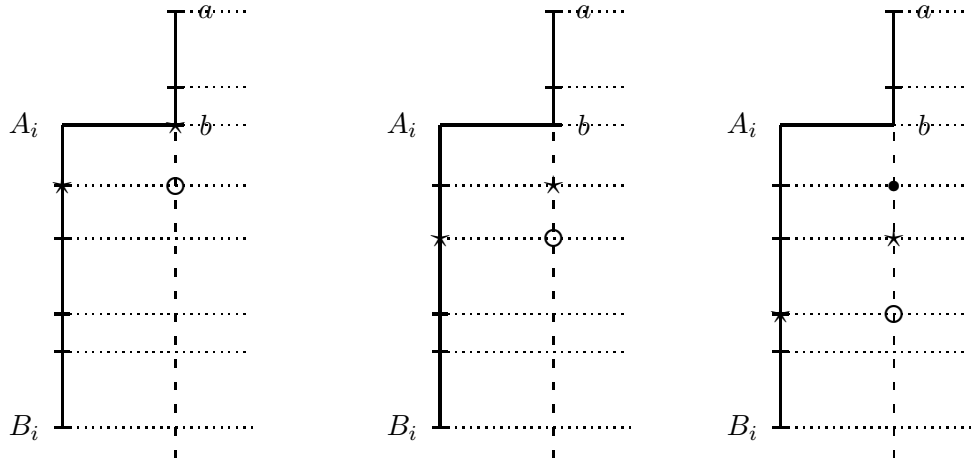

Figura 10: Geração de pontos de Steiner durante o varrimento vertical e sua ligação à grelha. O símbolo $\circ$ marca o ponto criado e $\star$ marca os que intervêm na sua ligação. Esses pontos definem as coordenadas do novo ponto de Steiner.

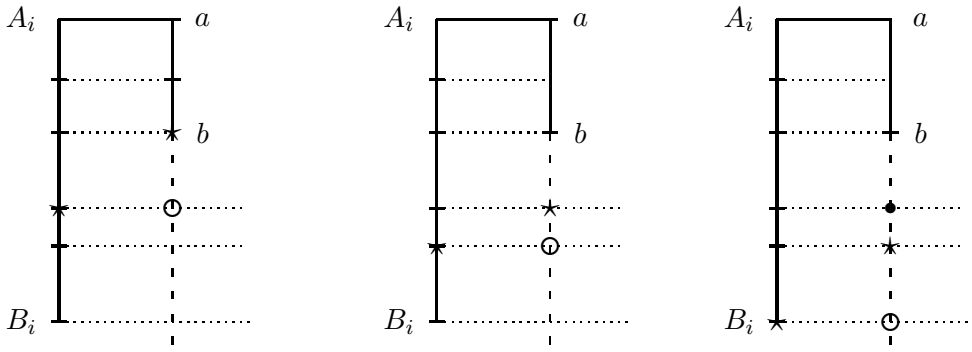

Figura 11: Geração no caso `6` durante varrimento vertical. O primeiro ponto activo sobre (A_i, B_i) é o adjacente Sul do adjacente Oeste de b (ver também `case 6` e `case 7` de `update`).

```
SEGMENT *update(SEGMENT *sweepst,SEGMENT *aibi,ROSE perpd,int event,SEGMENT *e){
  SEGMENT *newsg;  GNODE *gnode;
  int c = INDEXCOORD(perpd);  ROSE d = PERPENDIR(perpd);
  switch(event) {
  case 1:
    NXT(aibi) = new_segment(START(aibi),FINAL(aibi),aibi,NXT(aibi));
    if (!EMPTY(NXT(NXT(aibi)))) PREV(NXT(NXT(aibi))) = NXT(aibi);
    if (perpd == EAST) FINAL(aibi) = ADJ(START(e),WEST);
    gnode = break_edge(perpd,aibi,c,e,OPPOSITEDIR(d));
    if (perpd == EAST) START(NXT(aibi)) = ADJ(FINAL(e),WEST);
    FINAL(NXT(aibi)) = break_edge(perpd,NXT(aibi),c,e,d);
    START(aibi) = gnode;  FINAL(aibi) = START(e);
    START(NXT(aibi)) = FINAL(e);  return sweepst;
  case 3:
    d = OPPOSITEDIR(d);
  case 2:
    gnode = break_edge(perpd,aibi,c,e,d);
    EXTREME(aibi,OPPOSITEDIR(d)) = EXTREME(e,OPPOSITEDIR(d));
    EXTREME(aibi,d) = gnode;  return sweepst;
  case 4:
    gnode = break_edge(perpd,aibi,c,e,OPPOSITEDIR(d));
    START(aibi) = gnode;
    aibi = NXT(aibi);
    gnode = break_edge(perpd,aibi,c,e,d);
    FINAL(PREV(aibi)) = gnode;
    NXT(PREV(aibi)) = NXT(aibi);
    if (!EMPTY(NXT(aibi))) PREV(NXT(aibi))=PREV(aibi);
    free(aibi);  return sweepst;
  case 5:
    if (EMPTY(aibi)) return new_segment(START(e),FINAL(e),NULL,NULL);
    newsg = new_segment(START(e),FINAL(e),PREV(aibi),aibi);
    if (!EMPTY(PREV(aibi))) NXT(PREV(newsg)) = newsg;
    PREV(aibi) = newsg;  return sweepst;
  case 7:
    d = OPPOSITEDIR(d);
  case 6:
    if (perpd == EAST) EXTREME(aibi,OPPOSITEDIR(d))=ADJ(EXTREME(e,d),WEST);
    gnode = break_edge(perpd,aibi,c,e,d);
    EXTREME(aibi,d) = gnode;  EXTREME(aibi,OPPOSITEDIR(d)) = EXTREME(e,d);
    return sweepst;
  case 8:
    if (!EMPTY(PREV(aibi))) NXT(PREV(aibi)) = NXT(aibi);
    else sweepst = NXT(aibi);
    if (!EMPTY(NXT(aibi))) PREV(NXT(aibi)) = PREV(aibi);
    free(aibi);  return sweepst;
  case 9:
    NXT(aibi) = new_segment(START(e),FINAL(e),aibi,NULL);
    PREV(NXT(aibi)) = aibi;  return sweepst; } }
```

Para determinar os pontos de Steiner criados pelo evento na direcção `d` e liga-los à grelha, é chamada a função `break_edge`, a qual retorna o último nó gerado.

```
GNODE *break_edge(ROSE perpd,SEGMENT *aibi,int coord,SEGMENT *e,ROSE d)
{ int x, y;
  ROSE symd = OPPOSITEDIR(d),symperpd;
  GNODE *gnode, *start, *final, *cordnode;

  symperpd = OPPOSITEDIR(perpd),
  final = EXTREME(aibi,d);
  if (perpd!=SOUTH) start = EXTREME(aibi,symd);
  else start = ADJ(final,symd);
  cordnode = EXTREME(e,d);
  do{
    start = ADJ(start,d);
    coords(start,cordnode,coord,&x,&y);
    gnode = new_gridnode(x,y);
    link_gridnode(gnode,cordnode,d,symd);
    link_gridnode(gnode,start,perpd,symperpd);
    cordnode = gnode;
  }  while(start != final);
  return gnode;
}
```

A variável `cordnode` indica o último nó já criado sobre a corda que prolonga a aresta e `start` indica o nó activo sobre (A_i, B_i). O procedimento `coords` determina as coordenadas do novo ponto de Steiner, usando as coordenadas de `cordnode` e `start`.

```
void coords(GNODE *node,GNODE *cordnode,int coord, int *x, int *y)
{
  if (coord) { *x = POINT_X(node);  *y = POINT_Y(cordnode);
  } else { *x = POINT_X(cordnode); *y = POINT_Y(node);  }
}
```

O procedimento `link_node` liga um novo ponto de Steiner a um nó da grelha dado.

```
void link_gridnode(GNODE *newgnode, GNODE *node, ROSE d, ROSE symd)
{
  ADJ(newgnode,d) = ADJ(node,d);
  if (!EMPTY(ADJ(node,d)))  ADJ(ADJ(node,d),symd) = newgnode;
  ADJ(node,d) = newgnode; ADJ(newgnode,symd) = node;
}
```

Finalmente, observamos que o polígono pode ser dado por uma estrutura circular, em que cada elemento define um vértice e contem um apontador para o nó da grelha que corresponde a esse vértice.

```
typedef struct vertex {
  int label, point[2];
  GNODE *posgrid;
  struct vertex *nxt, *prev;
} VERTEX;
```

Estabelece-se assim a ligação dos vértices do polígono aos nós da grelha que lhes correspondem, e consequentemente, às r-peças de que são vértices. Para aceder a tais peças pode ser necessário analisar localmente a relação de adjacência dos nós da grelha.

Passamos ao problema da determinação da região que cada vértice vê, para cada r-peça.

3 Determinação da Região de Visibilidade para cada Vértice

São conhecidos algoritmos que determinam a região de visibilidade dum ponto em tempo e espaço linear no número de vértices do polígono [6, 9, 10]. No âmbito deste trabalho interessa-nos determinar a localização de cada peça duma dada partição relativamente a cada uma dessas regiões. Como os métodos acima referidos não respondem directamente a esta questão, propomos a adaptação dum método baseado na propagação de cones de visibilidade, inspirando-nos num trabalho de Aronov et al. [1]. A ideia é sugerida pela Fig. 12.

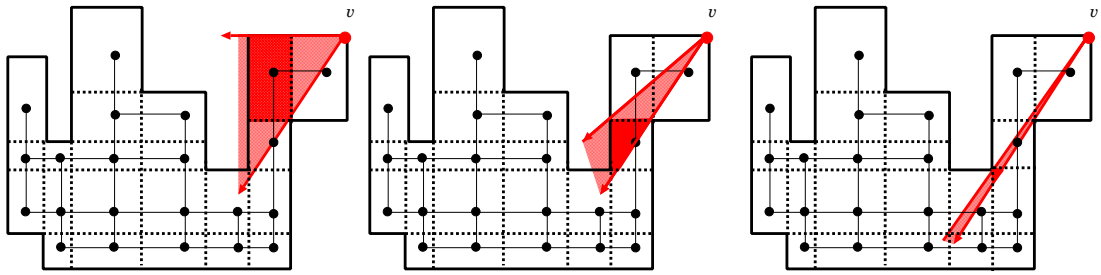

Figura 12: Determinação de visibilidade de v por propagação de cones de visibilidade.

Começamos por analisar o caso da partição $\Pi_{\text{r-cut}}$. A visibilidade é propagada pelas peças que têm vértice v às suas adjacentes e sucessivamente. O cone de visibilidade que atinge uma dada peça p é determinado por v e pela secção visível das arestas partilhadas entre p e as suas adjacentes já analisadas. O grafo dual da partição (isto é, o grafo que traduz a relação de adjacência entre peças) será explorado em largura a partir das peças adjacentes a v. Como apenas peças total ou parcialmente visíveis podem propagar visibilidade, nem todas as peças serão visitadas.

Uma peça é totalmente visível de v quando v vê todos os seus pontos e é parcialmente visível quando v vê apenas uma secção da peça sendo o interior dessa secção não vazio. Uma peça é visível de v se for totalmente ou parcialmente visível de v.

No caso de v ser um vértice reflexo, v é vértice de três r-peças. Em [11], F. Marques mostra que os três quadrantes definidos por v e as suas peças adjacentes podem ser explorados em simultâneo. No entanto, para provar que o método é completo é mais simples supor que os três quadrantes serão analisados separadamente, como na Fig. 13.

Seja $p_{0,0}$ a única r-peça adjacente a v (isto é, que tem v como vértice) se v for convexo ou uma das três peças que têm v como vértice se for reflexo. Seja (d_1, d_2) o par de direcções que define a posição do vértice de $p_{0,0}$ diametralmente oposto a v relativamente a v, com $d_1 \in \{Este, Oeste\}$ e $d_2 \in \{Norte, Sul\}$. O quadrante que o método explora é definido por $(p_{0,0}, d_1, d_2)$, Se v for reflexo, serão explorados os três quadrantes $(p_{0,0}, d_1, d_2)$ cada um definido por uma peça $p_{0,0}$ adjacente a v.

Como nos métodos de pesquisa em largura, as peças a visitar serão colocadas numa fila que obedecerá a uma disciplina FIFO (*first in first out*). Classifica-se $p_{0,0}$ como totalmente

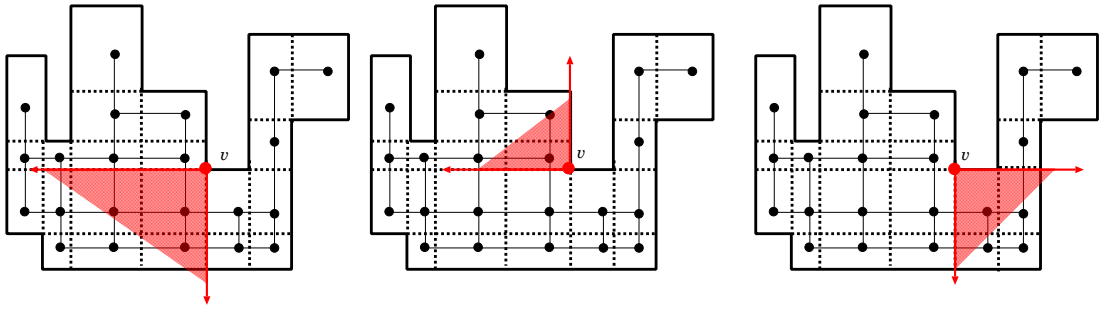

Figura 13: Determinação de visibilidade para vértices reflexos: exploração por quadrantes, neste caso (Oeste,Sul), (Oeste,Norte) e (Este,Sul).

visível de v, e coloca-se na fila as adjacentes a $p_{0,0}$ segundo as direcções d_1 e d_2, isto é, as peças $p_{1,0}$ e $p_{0,1}$, se existirem. Esta designação para as peças decorre da ordem pela qual são visitadas na pesquisa em largura, como se vê na Fig. 14.

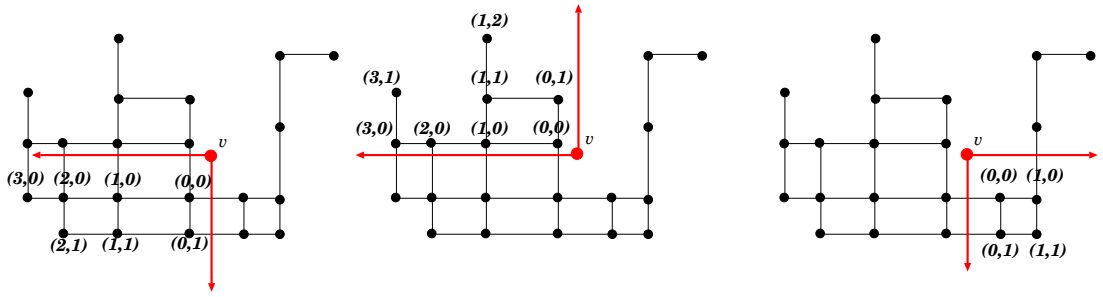

Figura 14: Numeração das peças por pesquisa em largura a partir de $p_{0,0}$ segundo d_1 e d_2.

Em cada passo, retira-se a peça $p_{i,j}$ que está à cabeça da fila. Determina-se qual a secção dessa peça que v vê. Para isso, considera-se o cone definido por v e pelas secções visíveis nas arestas que $p_{i,j}$ partilha com as suas adjacentes segundo $-d_1$ e $-d_2$. Note-se que, como $p_{i,j}$ foi colocada na fila, existe pelo menos uma das peças $p_{i-1,j}$ e $p_{i,j-1}$. Depois de determinada a secção de $p_{i,j}$ que é visível, coloca-se na fila cada uma das suas adjacentes (segundo d_1 e d_2, isto é, $p_{i+1,j}$ e $p_{i,j+1}$), desde que exista, não esteja já na fila e a aresta que $p_{i,j}$ partilha com ela seja visível de v. Uma aresta é visível de v se pelo menos dois dos seus pontos interiores forem visíveis de v. Isto quer dizer que, o segmento que une v e cada um desses pontos não contem pontos exteriores ao polígono. O método termina quando a fila ficar vazia.

Proposition 1 *Todas as r-peças visíveis dum vértice v são analisadas pelo método descrito, qualquer que seja v.*

Proof. Seja v um vértice de $\mathcal{P}$ e $p_{0,0}$ uma ou a r-peça adjacente a v. Vamos mostrar que todas as r-peças $p_{i,j}$ do quadrante $(p_{0,0}, d_1, d_2)$ que são visíveis de v são analisadas pelo método. A prova será por indução sobre $i+j$, isto é sobre a distância da peça $p_{i,j}$ à peça $p_{0,0}$ no grafo.

Para $i + j = 0$, a peça é $p_{0,0}$, que o método classifica como totalmente visível. Seja agora $p_{i,j}$ uma qualquer peça visível de v, com $i, j \geq 0$ e $i + j > 0$. Suponhamos como hipótese de indução que todas as peças $p_{i',j'}$ visíveis de v com $i', j' \geq 0$ e $i' + j' < i + j$ foram analisadas.

Para que a peça $p_{i,j}$ seja visível de v terá que ocorrer uma das situações esquematizadas na Fig. 15, a menos de simetria ou seja, de troca de d_1 com d_2.

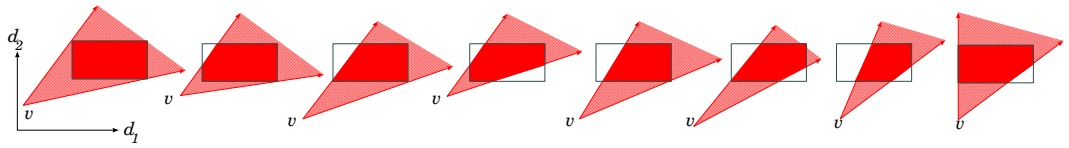

Figura 15: Definição da secção visível duma peça como intersecção dum cone de visibilidade com a peça: todos os casos para peças visíveis, a menos de troca de d_1 com d_2.

Pode concluir-se que se $p_{i,j}$ for visível então a sua aresta na direcção $-d_1$ ou a sua aresta na direcção $-d_2$ é visível. Está a considerar-se a posição das arestas relativamente a um ponto no interior da peça. É importante notar que, a definição de $\Pi_{\text{r-cut}}$ garante que tais arestas serão também arestas das peças adjacentes a $p_{i,j}$ segundo essas direcções, se estas existirem. Caso contrário, são arestas ou partes de arestas do polígono. Como o cone de visibilidade não contem pontos exteriores ao polígono, pode concluir-se, por análise de casos, que se a aresta na direcção $-d_1$ for visível então a peça $p_{i-1,j}$ existe e é visível ou $p_{i-1,j}$ não existe mas existe $p_{i,j-1}$ e é visível. Do mesmo modo, se a aresta $-d_2$ for visível, a peça $p_{i,j-1}$ existe e é visível ou $p_{i,j-1}$ não existe mas existe $p_{i-1,j}$ e é visível. Assim, pelo menos uma das peças $p_{i-1,j}$ e $p_{i,j-1}$ existe e é visível e, portanto, por hipótese de indução, foi analisada. Consequentemente, colocou $p_{i,j}$ na fila. Logo, o método analisa $p_{i,j}$. □

O número máximo de r-peças é $(3r^2+6r+4)/4$ se r for par e $3(r+1)^2/4$ se r for ímpar, sendo $r=(n-4)/2$ o número de vértices reflexos do polígono ortogonal dado [2]. O método pode ser adaptado para outro tipo de partições, como por exemplo triangulações ou para outras mais simples como a partição determinada por extensão das arestas horizontais (que se vê na Fig. 16) ou das arestas verticais, as quais têm no máximo $r+1$ peças. Para facilitar

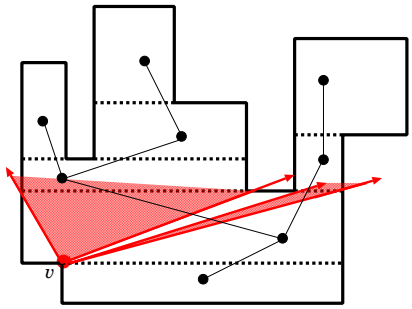

Figura 16: Determinação de visibilidade por propagação de cones de visibilidade.

a análise da secção visível é importante que as peças sejam convexas. Como se pode ver na Fig. 16 pode acontecer que as arestas das peças estejam contidas ou contenham arestas das suas adjacentes. De qualquer modo, a propagação de visibilidade às peças adjacentes será efectuada através das *janelas* definidas pela parte comum. A análise terá início nas peças que contêm v na fronteira, as quais são totalmente visíveis de v, se as peças da partição forem convexas, como para $\Pi_{\text{r-cut}}$.

Vamos agora tratar o terceiro problema referido.

4 Determinação da Partição Induzida pelas Secções Visíveis

Suponhamos dada uma r-peça R e o conjunto $\{V_R^s(v) \mid v \in G_R^s\}$ das suas secções visíveis. Vamos descrever a ideia dum método para construir a partição de R induzida por essas secções (a qual representamos por $\mathcal{Z}_R$) e simultaneamente determinar os vértices que vêem cada peça dessa partição. A Fig. 17 apresenta resumidamente a ideia geral.

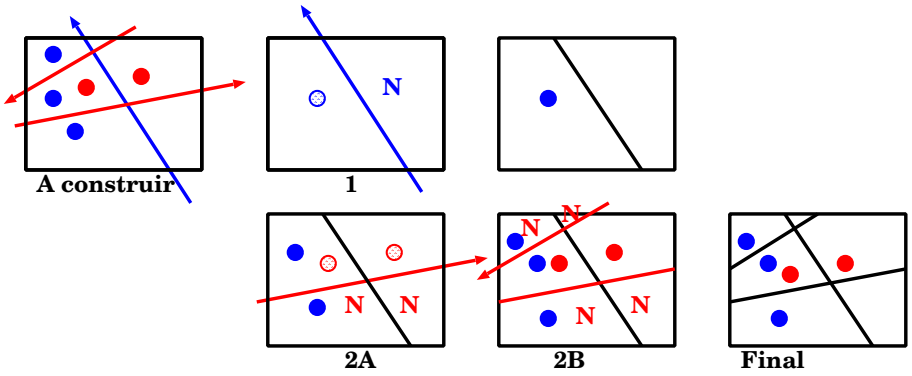

Figura 17: Ideia para construção da partição induzida por secções de visibilidade.

As peças de $\mathcal{Z}_R$ são faces do arranjo de segmentos de recta constituído pelas arestas de R e pelas arestas das suas secções visíveis. Cada secção de visibilidade é um polígono simples. Supomos que é dada a sequência de vértices da secção, ordenados no sentido anti-horário.

Lemma 1 *Cada secção de visibilidade é definida por uma ou duas arestas oblíquas.*

Proof. A secção da r-peça R que é visível do vértice v, representada por $V_R^s(v)$, é a intersecção de R com um cone (de visibilidade) com vértice em v, tendo assim, no máximo, duas arestas oblíquas. As arestas horizontais e/ou verticais das secções visíveis são redundantes para a construção de $\mathcal{Z}_R$, pois, por definição de $\Pi_{\text{r-cut}}$, têm que estar contidas nas arestas de R. Dada a sequência de vértices de $V_R^s(v)$, ordenados no sentido anti-horário, $V_R^s(v)$ está situada à esquerda das suas arestas, e em particular à esquerda das suas arestas oblíquas. □

Como consequência deste resultado, teremos na implementação que manter informação sobre a, ou as duas, arestas oblíquas que definem cada secção. Basta essa informação para caracterizar a secção, admitindo que as arestas estão orientadas de forma que a região visível se situe à sua esquerda. Na Fig. 17 as secções encontram-se representadas por rectas de suporte dessas arestas, as quais estão orientadas de acordo com a convenção referida.

O próximo resultado é importante para a caracterização dos vértices que verão cada face do arranjo. Foi já usado e/ou demonstrado em trabalhos anteriores, por exemplo, de Ghosh [7], Bosé et al. [4], Guibas et al. [8]

Lemma 2 *Quaisquer dois pontos no interior duma dada face $\mathcal{C}$ de $\mathcal{Z}_R$ vêem exactamente os mesmos vértices do polígono P. O interior de $\mathcal{C}$ é ou totalmente visível ou totalmente invisível de cada vértice de P.*

Pelo Lema 2, qualquer ponto interior de $\mathcal{C}$ pode ser usado para verificar se essa face é totalmente visível dum vértice dado. A um tal ponto chamaremos *ponto testemunho*. Como as faces são conjuntos convexos [7], é fácil encontrar pontos no interior de cada face. Por

exemplo, o *centróide* da face é um ponto interior, o qual é dado por $\frac{1}{n_{\mathcal{C}}} \sum_{i=1}^{n_{\mathcal{C}}} w_i$, sendo $w_1, \ldots, w_{n_{\mathcal{C}}}$ os vértices de $\mathcal{C}$. Pelo clássico Teorema de Carathéodory [5, 13] podemos em alternativa considerar um ponto no interior do triângulo definido por três vértices de $\mathcal{C}$, por exemplo, $\frac{1}{3}(w_1 + w_2 + w_3)$. Será assim este o ponto testemunho da face $\mathcal{C}$.

4.1 Determinação de $\mathcal{Z}_R$

O algoritmo que propomos para determinar o conjunto de vértices que vêem cada face de $\mathcal{Z}_R$, baseia-se num algoritmo para construção de arranjos, descrito por Berg et al. em [3].

O arranjo é construído incrementalmente, a partir do arranjo constituído pelas arestas do rectângulo R. Em cada iteração, é considerada uma nova secção visível e são inseridas as suas arestas oblíquas no arranjo anterior, obtendo-se um novo arranjo. Pelo Lema 1, um ou dois segmentos serão inseridos por secção.

Quando se insere um novo segmento, algumas faces do arranjo são divididas. Essas faces são visitadas segundo uma certa ordem, que depende da orientação que é dada ao segmento de recta. O método usado neste passo corresponde ao descrito em [3] para construção de arranjos, sendo apenas adaptado para guardar informação relevante para a caracterização da visibilidade. A ideia é esquematizada na Fig. 18.

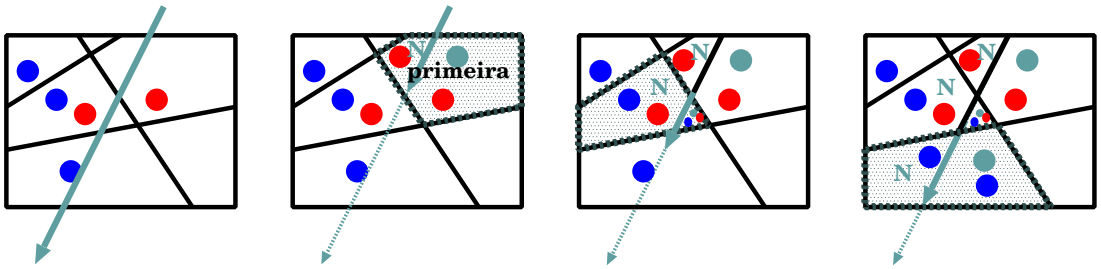

Figura 18: Corte de faces por inserção dum segmento de recta. As faces marcadas com "N" são não visíveis.

Exactamente uma das duas faces do arranjo intermédio que resultaram da divisão da primeira face visitada intersecta a secção visível. O mesmo acontece com as restantes faces que vão sendo partidas em dois pelo segmento que está a ser inserido.

Quando a secção é definida por dois segmentos oblíquos, a construção do novo arranjo é efectuada em dois passos, sendo inserido um segmento em cada passo, conforme se exemplifica na Fig. 19.

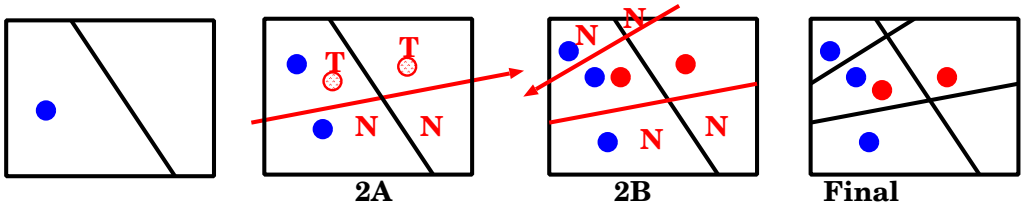

Figura 19: Corte por secção definida por dois segmentos de recta oblíquos. Aqui "T" significa que a face poderá estar dentro da secção visível. As faces marcadas com "N" são não visíveis.

Depois de ter inserido o primeiro segmento, algumas faces ficaram marcadas com "T", significando que *"talvez"* estejam dentro da secção visível. Quando se inserir o segundo

segmento poder-se-á decidir se são totalmente visíveis ou apenas parcialmente. No entanto, não será necessário, nem desejável, efectuar explicitamente esta marcação intermédia.

De facto, suponhamos que é mantida informação sobre as faces criadas pela inserção dos dois novos segmentos. Podemos simplesmente determinar um ponto no interior de cada uma delas (pelo processo descrito acima) e verificar qual é a posição desse ponto relativamente à região visível. É fácil verificar se está dentro ou fora dessa região uma vez que esta é definida por dois segmentos cuja expressão analítica se conhece.

Note-se que, caso a secção seja definida por um único segmento oblíquo, será ainda mais simples analisar o posicionamento desses pontos interiores, a que chamámos pontos testemunho. Marcamos então esses pontos como visíveis ou não visíveis conforme se situam dentro ou fora da região visível.

Partindo agora duma qualquer face dentro da região visível e que se sabe já ser visível (por se ter marcado o ponto testemunho), propagamos a visibilidade às faces adjacentes e depois, sucessivamente, às adjacentes de adjacente. A Fig. 20 ilustra esta ideia.

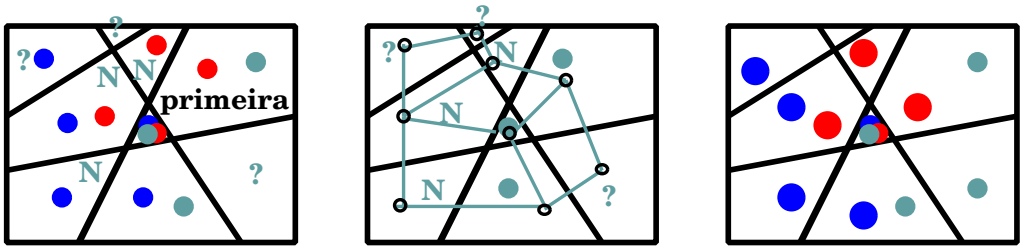

Figura 20: Propagação de visibilidade. Faces marcadas com "?"são visíveis sse alguma das suas adjacentes o for. As faces marcadas com "N" são não visíveis.

Mais formalmente, efectuaremos uma visita (implícita) do grafo dual do arranjo. Este é o grafo que traduz a adjacência das faces do arranjo. É um grafo conexo, pelo que poderemos aceder de qualquer face a qualquer outra. Mais importante ainda é o facto da secção de visibilidade ser também conexa, e consequentemente, é possível aceder de uma qualquer face visível a outra qualquer face visível, visitando nesse caminho apenas faces visíveis.

Iremos usar esta propriedade para detectar as faces que se encontram na região visível, conseguindo assim localizar a pesquisa apenas numa parte do arranjo. As faces adjacentes de faces marcadas como visíveis são visíveis se e só se não tiverem sido marcadas (na primeira fase) como não visíveis. Assim, quando se marcar uma face como visível, todas as suas adjacentes não visitadas serão marcadas como visíveis. Se durante a visita se chegar a uma face marcada como não visível, então não se prosseguirá dessa face para as suas adjacentes.

A Fig. 21 apresenta um exemplo maior. Pretende mostrar que com este método de propagação se pode evitar analisar faces que não estão próximas da região visível e não estão por isso nela incluídas. Assim, permite melhorar a eficiência média do algoritmo.

5 Conclusão

Apresentaram-se algoritmos que resolvem de forma eficiente os três problemas abordados. Poder-se-ia ainda procurar usar estruturas de dados que garantissem um desempenho óptimo. Não foi essa a abordagem que se seguiu, dada a aplicação em vista ser a resolução dum problema que é NP-hard. Não se prevê em geral que a não utilização de algoritmos óptimos se reflicta significativamente num pior desempenho, uma vez que cada um dos sub-problemas

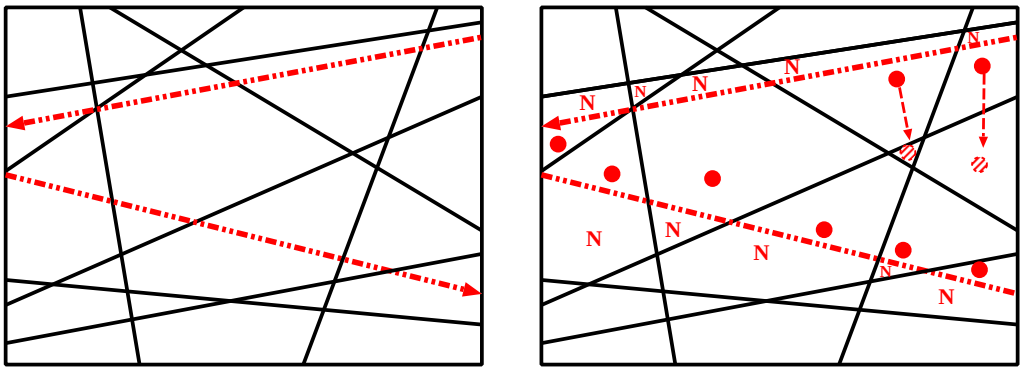

Figura 21: O método de propagação de visibilidade permite restringir a análise a um subconjunto relevante de faces do arranjo.

abordados, é resolvido uma única vez. O segundo e terceiro problemas são resolvidos uma vez para cada vértice e peça, mas, o terceiro poderá não o ser para todas as peças, uma vez que o método descrito em [14] permite empregar *lazy evaluation*. Mais detalhes de implementação podem ser encontrados em [11].

Referências

[1] Aronov B., Guibas L., Teichmann M., Zang L.: Visibility queries and maintenance in simple polygons. Discrete and Computational Geometry, 27 (2002) 461-483.

[2] Bajuelos A.L., Tomás A.P., Marques F.: Partitioning orthogonal polygons by extension of all edges incident to reflex vertices: lower and upper bounds on the number of pieces. In António Laganà et al. (Eds) Proceedings of Computational Science and its Applications (ICCSA 2004), LNCS 3045, Springer Verlag (2004) 127–136.

[3] De Berg M., Van Kreveld M., Overmars M., and Schwarzkopf O.: Computational geometry (Algorithms and Applications), Springer-Verlag (1997).

[4] Bosé P., Lubiw A., and Munro J. I.: Efficient visibility queries in simple polygons. Proc. 4th Canad. Conf. Comput. Geom. (1992) 23–28.

[5] Carathéodory C.: Über den Variabilitätsbereich der Fourierschen Konstanten von positiven harmonischen Funktionen, Rendiconto del Circolo Matematico di Palermo 32 (1911) 193-217.

[6] ElGindy H., Avis D.: A Linear Algorithm for Computing the Visibility Polygon from a Point, J. Algorithms 2 (1981) 186–197.

[7] Ghosh S. K.: Approximation algorithms for art gallery problems. Proc. Canadian Information Processing Society Congress. (1987) 429–434.

[8] Guibas L. J., Motwani R., and Raghavan P.: The robot localization problem in two dimensions. Proc. 3rd ACM–SIAM Symp. Discrete Algorithms (1992) 259–268.

[9] Joe B., Simpson R.B.: Corrections to Lee's Visibility Polygon Algorithm. BIT 27 (4) (1987) 458–473.

[10] Lee D. T.: Visibility of a simple polygon. Computer Vision, Graphics, and Image Processing 22 (1983) 207–221.

[11] Marques F.: Minimização do Número de Vigilantes em Galerias de Arte por Aproximações Sucessivas. Tese de mestrado submetida à Faculdade de Ciências da Universidade do Porto (Junho 2004).

[12] Preparata F. P. , Shamos M. I. : *Computational Geometry – An Introduction*, Springer-Verlag (1985).

[13] Schrijver A.: *Theory of Linear and Integer Programming*, Wiley-Interscience (1986).

[14] Tomás A. P., Bajuelos A. L., Marques F.: Approximation algorithms to minimum vertex cover problems on polygons and terrains. In P.M.A Sloot et al. (Eds): Proc. of ICCS 2003, LNCS 2657, Springer-Verlag (2003) 869-878.