Técnicas de Busca Aplicadas a Deteção de Contornos

Maria do Socorro Alves Taumaturgo de Farias

Dissertação de Mestrado

Instituto de Computação
Universidade Estadual de Campinas

Técnicas de Busca Aplicadas a Deteção de Contornos

Maria do Socorro Alves Taumaturgo de Farias[1]

Julho de 1997

Banca Examinadora:

- Marcus Vinicius Soledade Poggi de Aragão - DI/PUC/RJ (Orientador)
- Ricardo Dahab - Instituto de Computação - IC/Unicamp (Co-Orientador)
- Roberto de Alencar Lotufo - Faculdade de Engenharia Elétrica e de Computação - FEEC/Unicamp
- Jorge Stolfi - Instituto de Computação - IC/Unicamp
- João Carlos Setubal - Instituto de Computação - IC/Unicamp (Suplente)

[1]Este trabalho recebeu apoio financeiro do Conselho Nacional de Desenvolvimento Científico e Tecnológico — CNPq, e é atualmente financiado pela Fundação de Amparo à Pesquisa do Estado de São Paulo — FAPESP (Proc. No. 95/9029-1).

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA DO IMECC DA UNICAMP

Farias, Maria do Socorro Alves Taumaturgo de

F225t Técnicas de busca aplicadas a deteção de contornos / Maria do Socorro Alves Taumaturgo de -- Campinas, [S.P. :s.n.], 1997.

Orientador : Marcus Vinicius Soledade Poggi de Aragão

Dissertação (mestrado) - Universidade Estadual de Campinas, Instituto de Computação.

1. Detecção de sinais. 2. Otimização combinatória. 3. Programação heurística. 4. Algoritmo genético. I. Poggi de Aragão, Marcus Vinicius Soledade. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

Técnicas de Busca Aplicadas a Deteção de Contornos

Este exemplar corresponde à redação final da Dissertação devidamente corrigida e defendida por Maria do Socorro Alves Taumaturgo de Farias e aprovada pela Banca Examinadora.

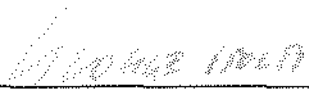
Campinas, 11 de julho de 1997.

Marcus Vinicius Soledade Poggi de Aragão
DI/PUC/RJ
(Orientador)

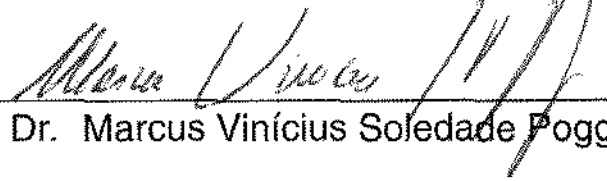
Dissertação apresentada ao Instituto de Computação, UNICAMP, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

Tese de Mestrado defendida e aprovada em 20 de junho de 1997 pela Banca Examinadora composta pelos Professores Doutores

Prof. Dr. Roberto de Alencar Lotufo



Prof. Dr. Jorge Stolfi



Prof. Dr. Marcus Vinícius Soledade Poggi de Aragão

Agradecimentos

Ao meu orientador Marcus Vinicius Soledade Poggi de Aragão, pessoa de uma vitalidade e conhecimentos notáveis, qualidades incentivadoras sobre o meu trabalho.

Aos professores Cid Carvalho de Souza, Neucimar Jerônimo Leite e Ricardo Dahab, pelos conselhos e pelo auxílio durante todo o curso.

Ao Conselho Nacional de Desenvolvimento Científico e Tecnológico - CNPq e à Fundação de Amparo à Pesquisa do Estado de São Paulo - FAPESP pelos apoios financeiros.

Ao meu pai José (*in memoriam*), que me ensinou em proporções que eu só compreendi quando eu não podia lhe dar mais nada além da minha imensa saudade.

À minha mãe Francisca, parâmetro de todo o meu caráter.

Ao Aminadab, pelo seu amor e carinho, e por tornar cada dia aparentemente normal em um dia especial.

Aos meus amigos distantes, sem os quais eu certamente seria outra pessoa e provavelmente de valores menores.

Aos meus amigos e companheiros de mestrado Cereja, Cláudia Nalon, Cláudio Nogueira, Cristiane Reis, Delano Beder, Eduardo Uchoa, Elder, Jerônimo, Marcos André, Nivando, Solange Sobral e muitos outros que não posso enumerar, mas que tornaram a convivência no instituto por demais alegre.

Ao meu irmão Jocimar e aos meus tios Neuda e Milton, pelos incentivos e carinho.

Conteúdo

Lista de Tabelas

Lista de Figuras

Lista de Algoritmos

Resumo

Deteção de contornos é uma importante tarefa em processamento de imagens. Desempenha um papel crucial em reconhecimento de padrões e sistemas de tratamento de imagens, o que explica o grande número de trabalhos que têm sido aplicados a detectar contornos de regiões em imagens obtidas em diferentes contextos. Apesar disto, a deteção automática de contornos é ainda um grande desafio para a tecnologia de hoje. Uma maneira de atacar o problema é através da definição de uma função de custo capaz de capturar o conceito de um contorno ou borda. Neste trabalho é usada a função de custo proposta por *Tan, Gelfand* e *Delp*, que é uma combinação linear de fatores tais como continuidade de uma linha de borda, dissimilaridade entre regiões limitadas por bordas e espessura de bordas. Esta função é usada para propor um novo algoritmo para minimização de custos. Trabalhos anteriores, que minimizam a mesma função de custos, foram propostos por *Tan et al.*, que realizou experiências com algoritmos baseados em busca local e em *Simulated Annealing*, além de *Bhandarkar et al.*, que desenvolveu um algoritmo genético para deteção de contornos. O algoritmo aqui apresentado é baseado na **busca em vizinhanças variáveis**, proposta por *Hansen e Mladenovic*, que é uma busca local que inteligentemente trata com diferentes vizinhanças. A experiência computacional mostrou resultados favoráveis à abordagem proposta em relação às anteriores, no que diz respeito ao tempo de CPU e aos valores encontrados para a função de custo.

Abstract

Edge detection is an important task in image processing. It plays a crucial role in object recognition and image understanding systems, what explains the great deal of research that has been dedicated to detect region edges in images obtained from different contexts. Nevertheless, the automatic edge detection is still a great challenge to today's technology. One way to tackle this problem is by the definition of a cost function able to capture the concept of an edge. We use the cost function proposed by *Tan, Gelfand and Delp*, which is a linear combination of factors such as continuity of an edge, region dissimilarity and edge thickness, to propose a new algorithm for the minization of the resulting function. Previous works that minimize equivalent cost function has been proposed by *Tan et al.*, who experimented with local search and Simulated Annealing algorithms, and *Bhandarkar et al.*, who developed a Genetic Algorithm to accomplish this task. Our algorithm is based on *Hansen and Mladenovic*'s Variable Neighborhood Search (VNS), which is a local search capable of playing smartly with different neighborhoods. The computational experience showed that our algorithm compares favorably with the previous ones with respect to cpu time and minimum cost function value found.

Capítulo 1

Introdução

O problema de deteção de bordas é muito estudado [24, 29, 20], tratando-se de uma importante tarefa em processamento de imagens. Detectar bordas em uma imagem consiste em encontrar pontos que caracterizam regiões de transição. Tais pontos dividem regiões dissimilares, as quais podem representar objetos distintos que figuram na imagem. Portanto, bordas em uma imagem correspondem a fronteiras, as quais dividem regiões distintas segundo um dado conjunto de características considerado.

Deteção de bordas geralmente envolve duas fases. A primeira corresponde a operações baseadas em cálculos diferenciais na função de níveis de cinza da imagem, através do uso de gradientes, operadores laplacianos ou ainda laplacianos da gaussiana. Os resultados desta primeira fase correspondem a um *mapa de candidatos a pontos de borda*, os quais serão combinados na segunda fase, a **binarização**, a fim de formar contornos que correspondem a pontos com valor 1 na imagem binarizada, enquanto os 0's são pontos que não representam contornos. Para a binarização, vários processos podem ser utilizados, tais como *deteção de contornos* [24], *concatenação de arestas* [29] e *agrupamento de arestas locais* [20], além de outros métodos [26]. A figura 1.1 mostra um exemplo de um resultado de um operador diferencial (a) e de uma binarização (c).

A maioria dos métodos propostos para realizar a fase de binarização estão totalmente condicionados aos resultados dos operadores diferenciais, os quais têm um conceito restrito do que é uma borda. Tais operadores decidem sobre a existência de uma borda em um certo ponto da imagem sem considerar as bordas detectadas nos pontos vizinhos. Isto é muito comum em algoritmos não-sequenciais, os quais aplicam operadores diferenciais em uma imagem e obtém resultados iguais independentemente da ordem pela qual os pontos da imagem foram percorridos.

Métodos de busca trabalham sobre um espaço de soluções de um problema à procura

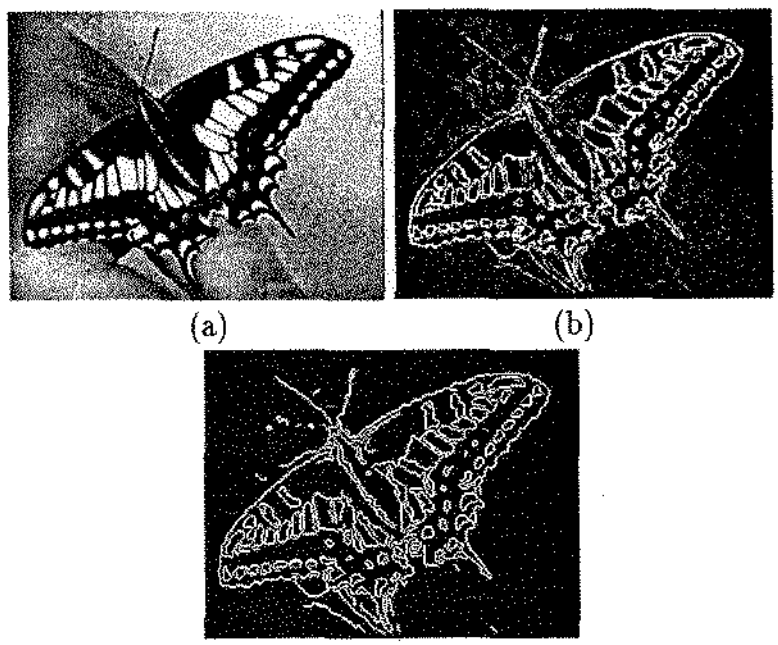


Figura 1.1: Exemplo de saídas das etapas para uma deteção de contornos: (a) Imagem de entrada; (b) Resultado de um operador diferencial para deteção de bordas; (c) Resultado da binarização

de soluções de "qualidade". Tal qualidade geralmente é avaliada por uma função de custo, que é formulada de maneira a primar por certas características que são desejáveis em uma boa solução. Tratam-se de procedimentos heurísticos, que não garantem a obtenção da melhor solução possível, mas tentam buscá-la de maneira inteligente. Seja $\mathcal{S}$ um espaço de soluções para um problema P e F uma função que avalia os custos das soluções de $\mathcal{S}$. A associação de F com $\mathcal{S}$ define uma métrica da qualidade das soluções de $\mathcal{S}$. De posse de F, o objetivo dos métodos de busca é encontrar a solução de maior qualidade S^*, a qual apresenta o menor custo possível: $F(S^*) = \min_{S \in \mathcal{S}} F(S)$. Os problemas para os quais não se conhece nenhum método analítico que obtenha S^* são geralmente solucionados através de métodos heurísticos. Tais problemas pertencem à classe dos **problemas de otimização** e a solução de menor custo possível S^* é conhecida como **ótimo global** ou solução ótima. São alguns exemplos de métodos heurísticos para problemas de otimização: a busca local, a *simulated annealing* [5, 14], os algoritmos genéticos [12], a busca *tabu* [6], a GRASP (*greedy randomized adaptive search procedures*) [10], etc.

Em deteção de contornos, a fase de binarização pode ser vista como uma busca pela configuração ótima dos pontos de borda. *Tan et al.* [27] propõem a formulação do problema como um problema de otimização por minimização de custos através de uma busca local. A função utilizada para avaliar os custos das soluções é baseada numa

definição de bordas que é geral o suficiente para incluir a maioria dos tipos de contorno (vide cap. 4). Uma das vantagens desta definição é que avalia a condição de borda de um ponto p valendo-se das configurações de borda nos pontos bem próximos de p, além de considerar a dissimilaridade em p e em seus vizinhos (onde a dissimilaridade é uma quantização semelhante aos resultados dos operadores diferenciais para deteção de bordas - vide cap. 4). Ou seja, a definição de bordas faz uso dinamicamente de resultados parciais, como o estado dos pontos de contorno próximos, bem como de resultados estáticos, como os valores de gradiente calculados por operadores de deteção de bordas. A abordagem de *Tan et al.* [27] utiliza-se de uma busca local, que é o esquema mais simples dentre os métodos de busca. O algoritmo compara duas soluções para a instância investigada do problema, soluções estas que diferem em um único ponto, e então escolhe a melhor em relação aos custos avaliados.

Tan et al. [28] também propõe uma abordagem para deteção de contornos, baseada em outro método de busca conhecido como *simulated annealing*. Este também utiliza a função de custos da abordagem anterior, assim como o faz a proposta de um algoritmo genético criada por *Bhandarkar et al.* [3].

Na dissertação aqui apresentada, são descritos o trabalho realizado e as estratégias adotadas para conduzir a implementação da *simulated annealing* de *Tan et al.* e do *algoritmo genético* de *Bhandarkar et al.* O esforço empreendido visa principalmente fazer uma reprodução o mais fiel possível dos algoritmos originais, a fim de que se possa fazer um estudo crítico baseado em resultados confiáveis. No capítulo 5 são descritos os métodos de *Tan et al.* e de *Bhandarkar et al.*, além de algumas propostas de melhoramento para o algoritmo genético. No capítulo 7 são apresentados os resultados computacionais que comprovam a qualidade das modificações inseridas nos algoritmos originais.

Os resultados mais importantes da pesquisa, no entanto, referem-se à utilização de um método de busca recente, denominado *busca em vizinhanças variáveis* - VNS [19]. O método é semelhante a uma busca local, sendo que se utiliza de estratégias variadas para gerar novas soluções, assim tentando ser dinâmico na exploração do espaço de soluções. A abordagem proposta para o problema de deteção de contornos pode ser avaliado no capítulo 6 e os resultados computacionais, que fazem uma comparação entre as implementações da *simulated annealing*, do algoritmo genético e da VNS proposta, estão apresentados no capítulo 7.

Capítulo 2

Conceitos e métodos para deteção de contornos

A compreensão ou a descrição de uma cena é o objetivo da Análise de Imagens. Seu primeiro passo é a segmentação, que corresponde a uma divisão da imagem de entrada em regiões, cada uma com certas propriedades que as diferenciam entre si. Segmentação de imagens é uma das aplicações de deteção de contornos. Depois de detectados os contornos, eles são então conectados a fim de formarem regiões fechadas, que correspondam a objetos na imagem de entrada.

As aplicações de análise de imagens são inúmeras: reconhecimento aeroespacial, descrição do cariótipo de microfotografias de células biológicas, caracterização de fotos de circuitos integrados, controle de qualidade de peças manufaturadas, análise do movimento de corpos rígidos e não rígidos, etc. Todos estes processos envolvem um passo inicial de segmentação.

Um exemplo de análise de imagens é a leitura automática de um texto escrito a máquina ou a mão. A entrada consiste de uma imagem do texto e a saída é um arquivo com o texto original transcrito. A princípio, é necessário isolar os caracteres na imagem original, ou seja, segmentar a figura em partes individuais. Só então a análise pode prosseguir com a identificação de cada caractere. Outro exemplo é a deteção de células cancerosas em microfotografias biomédicas. Primeiramente a imagem deve ser dividida em partes, isolando cada célula contida na figura. Então pode-se seguir com o reconhecimento e deteção das células anômalas.

A maioria dos operadores de deteção de bordas (veja seção 2.2.1) são baseados no cálculo da derivada primeira, ou da derivada segunda ou ainda do Laplaciano da Gaussiana [16] sobre a função de níveis de cinza da imagem a ser analisada. Buscar os pontos de

máximo da derivada primeira corresponde a identificar rápidas mudanças da função de níveis de cinza, que corresponderiam a contornos de objetos. Semelhantemente calculam-se os pontos de zero da derivada segunda, a fim de detectar bordas na imagem. Os resultados de tais operadores correspondem a um *mapa de candidatos a pontos de borda* ou *mapa de gradiente*, os quais são combinados em uma segunda fase de deteção de contornos, denominada **binarização**, a fim de se obter contornos que correspondem aos pontos com valor 1 na imagem binarizada, enquanto os 0's são pontos não pertencentes a bordas. O objetivo desta fase é identificar imparcialmente contornos. Para tanto, pode-se selecionar os pontos com derivadas primeiras acima de um certo limiar, ou ainda os pontos com derivadas segundas próximos de zero por no máximo um dado valor. Essas operações com limiares (*thresholding*) são frequentemente suscetíveis a ruídos que produzem bordas fragmentadas. Outros esquemas de deteção de bordas, baseados em *filtros ótimos* [4, 8, 16], *surface fitting* [13] e *sequencial contour tracing* [18, 7], apresentam métodos matemáticos sofisticados. Apesar disso ainda é difícil encontrar bordas reais em uma imagem. Parte da dificuldade consiste em tecer uma definição razoável do que seja uma borda. A maioria dos algoritmos têm uma visão restrita de bordas, e muitas vezes levam em consideração apenas os resultados dos operadores de deteção (filtros) para produzirem o resultado final. Ou seja, para compor o resultado de deteção de bordas em um ponto p, não são levados em consideração os resultados parciais em pontos próximos a p, e assim, a interdependência entre pontos vizinhos é relevada.

O objetivo deste capítulo é introduzir alguns métodos utilizados para deteção de contornos em análise de imagens, mais especificamente sobre o cálculo do mapa de gradiente e sobre a binarização, que são as etapas investigadas e implementadas por este trabalho.

2.1 Definições

Seja I uma imagem constituída de pontos referenciáveis através de pares ordenados (x, y), que correspondem às coordenadas do ponto. Portanto I consiste de uma matriz de pontos:

$$G = \{f(x,y); 1 \leq x \leq W \;\; 1 \leq y \leq H\},$$

onde W e H são as dimensões da imagem, respectivamente nas direções do eixo X e Y. f é conhecido como *função imagem*. O valor de cada ponto $f(x, y)$ é um nível de cinza no intervalo $0 \leq f(x, y) \leq 255$, onde 255 corresponde ao *branco*, que é o valor máximo de brilho. 0(zero) corresponde à ausência de brilho, ou seja, o *preto*. Os valores intermediários são mais "claros" quanto mais próximos de 255 e mais "escuros"mais próximos de zero.

Dado um ponto p pertencente a uma imagem S, dá-se o nome de janela $m \times m$ centrada

em p, ou sub-imagem $S(p, m \times m)$, a um conjunto de pontos pertencentes a um quadrado de lado m, cujo centro p corresponde às coordenadas $(m + 1)/2$ e $(m + 1)/2$, quando m é ímpar, e $m/2$ e $m/2$ para m par.

Métodos de pré-processamento usam uma pequena vizinhança de um ponto em uma imagem de entrada e produzem novos valores da função de níveis de cinza em uma imagem de saída. Tais métodos também são conhecidos como *filtragem*, na terminologia de processamento de sinais. De acordo com o objetivo do pré-processamento, as duas categorias mais relevantes para este trabalho são: **smoothing** ou *suavização*, cujo objetivo é suprimir ruídos e pequenas flutuações na imagem; e *operadores de gradiente*, que são baseados em derivadas locais da função imagem, derivadas estas que respondem mais intensamente em locais que apresentam mudanças rápidas da função, ou seja, locais que estão associados com contornos de objetos da cena capturada.

Outra classificação dos métodos de pré-processamento local baseia-se na propriedades das operações realizadas: **lineares** e **não-lineares**. As operações lineares fazem uma combinação linear dos valores da função da imagem de entrada $f(x, y)$ em uma determinada vizinhança $\mathcal{O}$, e os valores resultantes da operação são depositados em uma imagem de saída $g(x, y)$. A contribuição de cada ponto da vizinhança $\mathcal{O}$ é ponderada pelos coeficientes h:

$$g(x, y) = \sum_{(m,n)} \sum_{\in \mathcal{O}} h(i - m, j - n)\, f(m, n), \tag{2.1}$$

A operação da equação 2.1 descreve o que é denominado como uma **convolução** de f por h, onde h é denominado **máscara de convolução**. A notação correspondente é dada por: $h * f = f * h$. A figura 2.1 apresenta um exemplo de uma convolução de uma imagem I por um padrão F. Os pontos com intensidade igual a 0 (zero) foram omitidos. Operações baseadas em convolução (filtros) podem ser utilizadas tanto para suavizações como para operadores de gradiente.

2.2 Deteção de bordas

Deteção de bordas é uma técnica de análise de imagens que procura descontinuidades locais [24, 26, 11]. Estas são detectadas primeiro, e depois conectadas a fim de formar fronteiras unívocas e maximais entre as regiões. Uma descontinuidade é um lugar em que há uma mudança sensível em termos de níveis de cinza, caracterizando bordas ou contornos na imagem de entrada. As operações dessa abordagem são essencialmente locais, ou seja, em geral não se utilizam de conhecimento global da imagem para concluir

Imagem I

				1	1	1		
1	1	1		1	1	1		
				1	1	1		
	1			1	1	1		
	1							
	1			1	1	1		

Padrão F

0	0	0
1	1	1
0	0	0

Convolução de I por F

					1	2	3	2	1
1	2	3	2	1	1	2	3	2	1
					1	2	3	2	1
	1	1	1		1	2	3	2	1
	1	1	1		1	1	2	1	1
	1	1	1		1	2	3	2	1

Figura 2.1: Exemplo de convolução

se um dado ponto é ou não um ponto de borda característico.

Deteção de bordas pode ser dividida em dois passos: o primeiro consiste na utilização de um operador diferencial ou de gradiente, o qual responde a mudanças na função de níveis de cinza; o segundo passo utiliza os resultados do primeiro passo a fim de dar forma aos contornos. É sabido que o resultado dos operadores diferenciais são valores não binários, sendo maiores onde os níveis de cinza mudam mais radicalmente. A imagem resultante da aplicação de um operador diferencial corresponde a um mapa de "candidatos a pontos de borda", fornecendo, assim, graus de "aptidão a borda" dos pontos da imagem. Tal resultado não pode ainda ser utilizado como imagem de contornos final. O segundo passo se justifica no sentido de que é necessária uma combinação dos valores de gradiente detectados no primeiro passo a fim de se obter valores binários. Nas próxiams seções serão descritos alguns operadores de gradiente e os algoritmos mais conhecidos para binarização.

2.2.1 Operadores para deteção de bordas

O gradiente

O operador de gradiente, para funções contínuas f, é definido como

$$\nabla f(x,y) = \frac{\partial f}{\partial x}\vec{i} + \frac{\partial f}{\partial y}\vec{j}$$

A magnitude do gradiente é dada por:

$$|\nabla f(x,y)| = \left(\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2\right)^{\frac{1}{2}}$$

e a direção do gradiente é

$$\tan^{-1}\frac{(\partial f/\partial y)}{(\partial f/\partial x)}$$

onde $\vec{i}, \vec{j}$ são os vetores unitários nas direções positivas de x e y, respectivamente.

As funções de imagens em níveis de cinza são funções discretas e os operadores digitais de gradiente baseiam-se em aproximações, as quais não se utilizam da direção de $\nabla f(x,y)$ para obter respostas às bordas contidas em uma imagem. Em vez disso, os operadores discretos de gradiente inspiram-se simplesmente na magnitude do gradiente, que indica mudanças em níveis de cinza na direção da borda mais acentuada.

O operador de *Roberts* [22] responde a mudanças de f nas diagonais 45^o e 135^o:

$$(\triangle_+ f)(x,y) \equiv f(x+1,y+1) - f(x,y)$$
$$(\triangle_- f)(x,y) \equiv f(x,y+1) - f(x+1,y)$$

Este operador é sensível a mudanças significativas em valores de f entre dois pontos adjacentes. Assim o operador detectará somente bordas bem salientes e com alto contraste entre as superfícies que a formam. Bordas que se caracterizam por uma mudança gradual em níveis de cinza não serão detectadas. Além disso, o operador de *Roberts* [22], que é baseado em uma janela 2×2, é mais suscetível a ruídos que operadores como *Sobel* [9], *Kirsch* [15] e *Prewitt* [21], que são operadores de gradiente baseados em janelas 3×3.

A principal diferença entre esses operadores são os pesos associados aos elementos das janelas 3×3. O operador de *Sobel*, por exemplo, dá maior peso aos pontos mais próximos do ponto central:

$$\begin{aligned}(\triangle_{sx})(x,y) \equiv 1/4[&f(x+1,y-1) + 2f(x+1,y) + f(x+1,y+1)\\ &-f(x-1,y-1) - 2f(x-1,y) - f(x-1,y+1)]\\ (\triangle_{sy})(x,y) \equiv 1/4[&f(x-1,y-1) + 2f(x,y-1) + f(x+1,y-1)\\ &-f(x-1,y+1) - 2f(x,y+1) - f(x+1,y+1)]\end{aligned}$$

equivalendo a convolução de f por

$$\frac{1}{4}\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad e \quad \frac{1}{4}\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}.$$

Os pares de operadores diferenciais $\triangle_-$ e $\triangle_+$ ou $\triangle_{sx}$ e $\triangle_{sy}$, cada um correspondendo a uma direção, devem ser combinados em um único valor que represente a "magnitude do gradiente digital". Seja um par de operadores diferenciais genéricos, $\triangle_x$ e $\triangle_y$. Para combiná-los pode-se simplesmente tirar a média: $(\triangle_x + \triangle_y)/2$, mas isto é pouco usual. Em geral, utiliza-se o máximo entre as magnitudes nas duas direções.

O laplaciano

O laplaciano digital [24], ao contrário do gradiente, é um operador independente de orientação, pois trata-se de um operador diferencial de segunda ordem. O laplaciano $|\nabla^2 f| \equiv \partial^2 f/\partial^2 x + \partial^2 f/\partial^2 y$ tem seu análogo para imagens digitais dado por:

$$(\nabla^2 f)(x,y) \equiv 1/4[f(x+1,y) + f(x-1,y) + f(x,y+1) + f(x,y-1) - 4f(x,y)]$$

correspondendo a convolução de f por:

$$\frac{1}{4}\begin{bmatrix} & 1 & \\ 1 & -4 & 1 \\ & 1 & \end{bmatrix}$$

O laplaciano responde a bordas com valores positivos ou negativos. É comum utilizar-se o valor absoluto $|\nabla^2 f|$ ou apenas os valores positivos $(\nabla^2 f)^+ (\equiv \nabla^2 f$ se $\nabla^2 f \geq 0; \equiv 0$ se $\nabla^2 f < 0)$. O laplaciano é sensível a ruídos, por isso é comum o uso de variações do operador que são um pouco mais robustas: um desses "pseudo-laplacianos" é o máximo entre os valores absolutos das derivadas segundas nas direções x e y. Outra versão é o valor absoluto da diferença entre a média e a mediana computadas sobre a mesma vizinhança do ponto avaliado.

Diferença de Médias

Operadores que se utilizam das **diferenças de médias** objetivam a redução dos efeitos do ruído, que são comuns nas respostas de operadores como o laplaciano [24]. Seja o operador que fornece a média em uma janela 2×2:

$$\overline{f}_4 \equiv 1/4[f(x,y) + f(x+1,y) + f(x,y+1) + f(x+1,y+1)].$$

Para se utilizar tal média na composição de um operador de deteção de bordas, deve-se tomar diferenças entre médias $\overline{f}_4$ cujas janelas não se interceptem, como é o caso de $(\overline{\Delta}_{4x} f)(x,y) \equiv \overline{f}_4(x+1,y) - \overline{f}_4(x-1,y)$, o que equivale à convolução de f por:

$$\frac{1}{4}\begin{bmatrix} -1 & -1 & 1 & 1 \\ -1 & -1 & 1 & 1 \end{bmatrix}$$

Um operador baseado em diferença de médias dá respostas em pontos correspondentes a bordas, mas também responde nas periferias destas, em locais que não são bordas mas que causam respostas devido ao "tamanho" do operador. Veja o exemplo de resposta de $|\overline{\Delta}_{4x}|$ para um dado padrão de imagem na figura 2.2.

A operação de *supressão de pontos não-maximais*, a qual será denominada simplesmente por supressão não-maximal, visa eliminar essas respostas indesejáveis nas periferias de uma borda. As respostas em um ponto (x,y) são setadas em zero se existe uma resposta mais "forte" suficientemente próxima a (x,y) na direção perpendicular à borda a ser detectada. A borda objetivo do operador $|\overline{\Delta}_{4x}|$ tem direção vertical, portanto os pontos

$$\begin{array}{ccccccc} & . & . & . & . & & \\ . & 0 & 0 & 0 & 1 & 1 & 1 \\ . & 0 & 0 & 0 & 1 & 1 & 1 \\ & . & . & . & . & & \end{array} \quad \xrightarrow{\overline{\Delta}_{4x}} \quad \begin{array}{ccccc} 0 & 1/2 & 1 & 1/2 & 0 \\ 0 & 1/2 & 1 & 1/2 & 0 \end{array}$$

Figura 2.2: Exemplo de aplicação do operador $|\overline{\Delta}_{4x}|$

próximos a (x, y) a serem testados na busca de uma resposta mais "forte" são os pontos $(x-1, y)$ e $(x+1, y)$. Se uma média em uma janela 3×3 fosse utilizada em vez de uma 2×2, então a resposta ao padrão exemplificado na figura 2.2 seria 0 1/3 2/3 1 2/3 1/3 0, logo os pontos a serem testados pela *supressão não-maximal* seriam $(x-2, y)$, $(x-1, y)$, $(x+1, y)$ e $(x+2, y)$ a fim de eliminar as respostas 1/3 e 2/3. Portanto, a *supressão não-maximal* estende-se pela vizinhança de (x, y) proporcionalmente ao tamanho da janela da média utilizada.

Um operador baseado em uma janela $n \times n$ deve ser seguido de uma *supressão não-maximal* que testa os pontos que distam $n-1$ do ponto (x, y) na direção perpendicular à direção da borda representada pelo operador. Logo, o operador $(\overline{\Delta}_{2x} f)(x, y) \equiv 1/2[f(x, y) + f(x, y+1) - f(x-1, y) - f(x-1, y+1)]$ não necessita de *supressão não-maximal*, pois ele não responde indesejavelmente nas periferias de uma borda da imagem. Isso se dá devido às dimensões restritas do operador. Portanto, um operador baseado em uma janela $n \times n$ não deve ser utilizado para detectar regiões mais estreitas que n pontos, a fim de evitar competições dentro da operação de *supressão não-maximal*, que provavelmente se darão entre as bordas simetricamente opostas de uma região de largura menor ou igual a n.

Os operadores devem ser combinados a fim de se formar um operador de gradiente digital insensível a orientação de seus operadores componentes. Para tanto, geralmente se combinam dois operadores em direções perpendiculares, como nos exemplos a seguir:

$$\overline{\Delta}_{4x} \text{ e } \overline{\Delta}_{4y}: \quad \frac{1}{4}\begin{bmatrix} -1 & -1 & 1 & 1 \\ -1 & -1 & 1 & 1 \end{bmatrix} \quad e \quad \frac{1}{4}\begin{bmatrix} 1 & 1 \\ 1 & 1 \\ -1 & -1 \\ -1 & -1 \end{bmatrix}$$

$$\overline{\Delta}_{3x} \text{ e } \overline{\Delta}_{3y}: \quad \frac{1}{3}\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \quad e \quad \frac{1}{3}\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$

$$\overline{\Delta}_{2x} \text{ e } \overline{\Delta}_{2y}: \qquad \frac{1}{2}\begin{bmatrix} -1 & 1 \\ -1 & 1 \end{bmatrix} \quad e \quad \frac{1}{2}\begin{bmatrix} 1 & 1 \\ -1 & -1 \end{bmatrix}$$

Para combinar tais operadores pode-se utilizar a média de seus valores absolutos, ou o máximo entre os valores obtidos pelos dois operadores componentes, como é o caso do operador de *Prewitt* [21], que corresponde a $\overline{\Delta}_3 = max(\overline{\Delta}_{3x}, \overline{\Delta}_{3y})$.

Máscaras direcionais

O método que emprega *máscaras direcionais* é uma abordagem para deteção de bordas baseada em padrões de contorno [24]. Tais padrões, que serão denominados por **modelos de borda**, concorrem entre si em cada ponto da imagem. O padrão que melhor se adapta a um dado ponto dará a orientação e a medida da "aptidão a borda" daquele ponto.

Qualquer um dos operadores digitais diferenciais ou de diferença de médias pode ser utilizado para definir conjuntos de modelos de borda para deteção de contornos por *máscaras direcionais*. Tais modelos são, então, usados na convolução da função de níveis de cinza de uma imagem. O modelo que obtém o maior valor de convolução em um dado ponto determina a orientação da borda naquele ponto e o valor obtido expressará a magnitude da borda, valor este correspondente ao contraste do contorno.

Por exemplo, o operador de *Roberts* serve de inspiração para os seguintes modelos de borda:

$$\begin{matrix} 0 & 1 \\ -1 & 0 \end{matrix}, \quad \begin{matrix} 0 & -1 \\ 1 & 0 \end{matrix}, \quad \begin{matrix} 1 & 0 \\ 0 & -1 \end{matrix} \quad e \quad \begin{matrix} -1 & 0 \\ 0 & 1 \end{matrix}$$

Seguem exemplos de modelos de borda inspiradas em alguns operadores:

Prewitt $\overline{\Delta}_3$:

$$\begin{matrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{matrix}, \quad \begin{matrix} 1 & 1 & 0 \\ 1 & 0 & -1 \\ 0 & -1 & -1 \end{matrix}, \quad \begin{matrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{matrix}, \quad \begin{matrix} 0 & -1 & -1 \\ 1 & 0 & -1 \\ 1 & 1 & 0 \end{matrix},$$

$$\begin{matrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{matrix}, \quad \begin{matrix} -1 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 1 \end{matrix}, \quad \begin{matrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{matrix} \quad e \quad \begin{matrix} 0 & 1 & 1 \\ -1 & 0 & 1 \\ -1 & -1 & 0 \end{matrix}$$

Sobel :

$$\begin{matrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{matrix}, \quad \begin{matrix} 2 & 1 & 0 \\ 1 & 0 & -1 \\ 0 & -1 & -2 \end{matrix}, \quad \begin{matrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{matrix}, \quad \begin{matrix} 0 & -1 & -2 \\ 1 & 0 & -1 \\ 2 & 1 & 0 \end{matrix},$$

$$\begin{matrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{matrix}, \quad \begin{matrix} -2 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 2 \end{matrix}, \quad \begin{matrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{matrix} \quad e \quad \begin{matrix} 0 & 1 & 2 \\ -1 & 0 & 1 \\ -2 & -1 & 0 \end{matrix}$$

Prewitt define, ainda, algumas máscaras interessantes que dão peso positivo para cada cinco pontos consecutivos vizinhos do ponto central, e dá pesos negativos para os demais, com exceção do ponto central, que tem peso diferenciado para balancear o modelo de borda:

$$\begin{matrix} 1 & 1 & 1 \\ 1 & -2 & 1 \\ -1 & -1 & -1 \end{matrix}, \quad \begin{matrix} 1 & 1 & 1 \\ 1 & -2 & -1 \\ 1 & -1 & -1 \end{matrix}, \quad \begin{matrix} 1 & 1 & -1 \\ 1 & -2 & -1 \\ 1 & 1 & -1 \end{matrix}, \quad \begin{matrix} 1 & -1 & -1 \\ 1 & -2 & -1 \\ 1 & 1 & 1 \end{matrix},$$

$$\begin{matrix} -1 & -1 & -1 \\ 1 & -2 & 1 \\ 1 & 1 & 1 \end{matrix}, \quad \begin{matrix} -1 & -1 & 1 \\ -1 & -2 & 1 \\ 1 & 1 & 1 \end{matrix}, \quad \begin{matrix} -1 & 1 & 1 \\ -1 & -2 & 1 \\ -1 & 1 & 1 \end{matrix} \quad e \quad \begin{matrix} 1 & 1 & 1 \\ -1 & -2 & 1 \\ -1 & -1 & 1 \end{matrix}$$

Raízes da derivada segunda

Um técnica de deteção de bordas, baseada nas raízes da derivada segunda, foi proposta por *Marr-Hildreth*[16]. Um contorno na imagem, o qual corresponde a uma mudança abrupta na função da imagem f, responde com "picos" na derivada primeira de f, e consequentemente com zeros na derivada segunda. O filtro (operador) de *Marr-Hildreth* primeiro realiza um suavização (*smoothing*) na imagem original, em seguida o cálculo digital das derivadas segundas é feito. O operador de **gaussiana** para suavização $G(x, y)$ (também conhecido como filtro gaussiano ou simplesmente gaussiana) é dado por:

$$G(x, y) = \exp^{-\frac{x^2+y^2}{2\sigma^2}}$$

onde x, y são as coordenadas da imagem e σ é o desvio padrão da distribuição de probabilidades associada. σ é um parâmetro do filtro, que deve ser proporcional à escala da imagem tratada. As máscaras de convolução se tornam mais largas à medida que se deseje

maior precisão, aumentando-se para isso os valores de σ. Por exemplo, $\sigma = 4$ precisa de uma máscara de 40 pontos de largura. Depois de obtido o resultado da gaussiana, este é submetido a um filtro de derivada segunda: o operador laplaciano ∇^2. A operação total corresponde ao **laplaciano da gaussiana** de uma imagem $f(x, y)$, ou simplesmente **LoG** de $f(x, y)$, que é dado pela seguinte fórmula:

$$\nabla^2(G(x, y, \sigma) * f(x, y)) = (\nabla^2 G(x, y, \sigma)) * f(x, y).$$

O filtro $\nabla^2 G$ é independente da imagem em consideração, portanto pode ser pré-computado analiticamente, reduzindo, assim, a complexidade da operação de composição. Para tanto, sem perda de generalidade, a máscara de convolução correspondente ao **LoG** pode ser dada por :

$$h(x, y) = c(\frac{x^2 + y^2}{\sigma^2} - 1)\exp^{-\frac{x^2+y^2}{\sigma^2}}$$

onde c normaliza a soma dos elementos da máscara para a unidade. Uma das desvantagens do filtro **LoG** é que ele suaviza a imagem (2D gaussiana), podendo causar perda de bordas nesse processo.

O operador de Canny

O operador de *Canny* é uma grande contribuição para o cálculo das bordas ótimas de uma imagem. Diferentemente do operador de *Marr-Hildreth*, o operador de *Canny* [4] usa informação sobre a direção da borda. Seja n a variável que dá a direção perpendicular à borda, e seja G a gaussiana 2D. Seja G_n o seguinte operador:

$$G_n = \frac{\partial G}{\partial n} = n \cdot \nabla G$$

onde n pode ser dado por uma estimativa: $n = \nabla(G * f)/|\nabla(G * f)|$. Logo, a localização das bordas são dadas pelos máximos locais da convolução de f pelo operador G_n, na direção de n:

$$\frac{\partial}{\partial n} G_n * f = 0$$

Esta equação mostra como encontrar máximos locais na direção estimada por n, ou seja, na direção perpendicular às das bordas da imagem. Tal operação é equivalente à *supressão não-maximal*. Depois de encontrados esses máximos, calcula-se a magnitude das bordas de f através de:

$$|G_n * f| = |\nabla(G * f)|$$

Respostas dúbias nas periferias dos contornos são muito comuns em resultados de operadores para deteção de bordas. *Canny* propõe a utilização de um *limiar com hysterese* para eliminar respostas indesejadas. Quanto à escolha do desvio padrão σ da gaussiana G, a qual determina a escala do operador, *Canny* propõe a utilização de vários valores σ, cujos resultados sobre f devem ser agregadas para obter as respostas finais através de uma operação denominada **feature synthesis**. Todas as bordas significativas obtidas pelo operador com menor escala são marcadas primeiro, e as bordas de um *hipotético* operador de σ maior são sintetizadas através de uma previsão baseada nos valores do menor e do maior σ utilizado. As respostas sintetizadas são, então,comparadas com as respostas reais do operador com σ maior. Bordas adicionais só serão acrescentadas se elas tiverem uma resposta significativamente mais forte que a resposta prevista pelo operador sintético.

2.2.2 Deteção de bordas - Binarização

A binarização deve converter os valores da matriz de gradiente, que variam entre 0 e o máximo nível de cinza da imagem, para valores binários (0 representando não-borda e 1, borda). Esta conversão deve ser cuidadosa a fim de que a imagem binária de bordas seja composta de linhas contínuas e finas, além de que deve realmente corresponder, da melhor maneira possível, à matriz de gradiente. Tem-se, portanto, que a binarização consiste em detectar padrões na matriz de gradiente, tais como linhas, curvas ou quinas. Seguem vários métodos para binarização.

Thresholding

Um limiar pode ser utilizado para binarizar a matriz de gradiente, ou seja, valores da matriz que estejam abaixo do limiar serão setados em 0 e valores acima do limiar serão setados em 1. O objetivo é remover as bordas que apresentam respostas "fracas" na matriz de gradiente. Tais bordas poderiam corresponder a mudanças de níveis de cinza não significativas, talvez resultantes de ruído, ou de pequenas irregularidades de iluminação. Somente os pontos de borda mais expressivos, ou seja, aqueles com magnitudes da matriz de gradiente superiores ao limiar, seriam mantidos na imagem binarizada final, como contornos detectados. As desvantagens do *Thresholding* são seus resultados fortemente afetados por ruído e frequentemente com partes dos contornos ausentes.

Relaxação

Tal método, proposto por *Rosenfeld et al.*[23], utiliza uma técnica de relaxação para binarizar a matriz de gradiente. A relaxação visa utilizar iterativamente o contexto da vizinhança dos pontos visando definir pontos de borda na imagem final. Uma borda "fraca", ou seja, um ponto que apresente baixos valores na matriz de gradiente, mas que esteja posicionado entre duas bordas "fortes" dá um exemplo do uso de contexto, pois é altamente provável que essa borda "fraca" inter-posicionada seja parte de um contorno. Se, por outro lado, uma borda "forte" esteja cercada de "fracas", é provável que não seja parte de nenhum contorno. Essas idéias são utilizadas pela relaxação para iterativamente atualizar o grau de convicção $c^{(i)}(p)$, onde p é um ponto da matriz de gradiente e i é a iteração corrente.

Os valores iniciais $c^{(1)}(p)$ são indicados por uma normalização da matriz de gradiente previamente calculada, onde $0 \leq c^{(1)}(p) \leq 1$. Valores de $c^{(1)}(p)$, próximos a zero, são reflexos de baixo gradiente normalizado e o contrário para valores próximos a 1. Iterativamente os valores de $c^{(i)}(p)$ são atualizados e podem crescer ou decrescer dependendo do estado dos vizinhos e dos critérios adotados pela relaxação. Estes critérios visam a conexidade dos pontos de um contorno, e se um ponto p se caracteriza por conectar dois pontos vizinhos com alto grau de convicção, então o valor de $c^{(i)}(p)$ na relaxação provavelmente tenderá a 1. Um fator que é considerado para influenciar negativamente o grau de convicção, fazendo $c^{(i)}(p)$ tender a zero, é a fragmentação, ou seja, a isolação de um ponto de borda que liga dois pontos de convicção fraca. A convergência de uma relaxação para deteção de bordas está condicionada aos valores da matriz de convicção C. O algorimto só pára quando todos os valores de C forem iguais a 0 ou 1.

É importante notar que a relaxação para deteção de bordas utiliza informação direta da imagem apenas no início do processo, quando a matriz de gradiente é consultada para inicializar C. Nos passos seguintes os valores $c^{(i)}(p)$ dependem apenas do contexto da vizinhança, e nem a matriz de gradiente nem a imagem original são consultadas.

Busca heurística em grafos

Busca heurística é uma técnica que utiliza métodos de busca no espaço de soluções para o problema. Informação heurística é utilizada para limitar o espaço a ser investigado. A aplicação de busca em grafos para deteção de bordas foi primeiramente publicada por *Martelli* [18]. A abordagem formula o problema de deteção de bordas como um problema de busca heurística pelo menor caminho em um grafo. As arestas do grafo são elementos de borda definidos por dois pontos vizinhos na imagem: p e q ($|p_i - q_i| + |p_j - q_j| \leq 1$). Os pontos p e q definem o elemento de borda direcionado $\overline{pq}$. A orientação de um elemento

$\overline{pq}$ é a mesma direção da borda detectada previamente em $\overline{pq}$ pelos operadores de deteção de bordas. A intensidade de $\overline{pq}$ é também fornecida pela matriz de gradiente. Com isso pode-se definir o grafo G, onde os nós são os pontos da imagem, as arestas correspondem aos elementos de borda $\overline{pq}$ e os pesos das arestas são as intensidades correspondentes na matriz de gradiente.

Martelli [18] convenciona que um contorno é uma sequência de elementos de borda consecutivos que começa em um ponto inicial x_A e termina em x_B, não contendo ciclos e nem elementos cuja direção seja no sentido $x_B \rightarrow x_A$. Equivalentemente, um contorno seria um caminho no grafo e o problema de encontrar o melhor contorno que liga x_A a x_B reduz-se ao problema de encontrar o caminho ótimo no grafo que une tais pontos tal que o caminho seja estritamente orientado de x_A para x_B. O algoritmo consiste em explorar os caminhos do grafo G que saem de x_A e dirigem-se a x_B. Uma lista L de nós já visitados é mantida e os custos associados com os sub-caminhos de x_A até esses nós servem de conhecimento heurístico para decidir quais os próximos nós a serem visitados, ou seja, se os nós sucessores do nó x_c são escolhidos para serem visitados, então x_c é substituído na lista L pelos sucessores de x_c (se estes já não estiverem na lista). O custo dos sub-caminhos de x_A até os novos nós são calculados. Essa operação é denominada de "expansão". A busca acaba quando x_B é atingido por uma expansão. Esse algoritmo é conhecido como *A-algorithm*. Para implementar tal método uma função de custo deve ser definida tal que seja uma estimativa do custo dos caminhos de x_A a x_B. Alguns fatores de custo são comumente utilizados na composição de uma função de custo para o problema de deteção de bordas, tais como:

- *magnitude da matriz de gradiente dos pontos que compõem o caminho.* Quanto maior a soma dessas magnitudes, mais robusta é a borda e menor é o seu custo;
- *curvatura da borda.* Não são desejáveis bordas que possuem seus pontos formando estruturas retorcidas de pequenas dimensões. A função de custo deve penalizar tais curvaturas que formem angulos inferiores a 90°;
- *estimativas da distância ao ponto final* x_B.

Para usar busca em grafos para deteção de bordas, deve-se primeiramente definir algumas estratégias para as operações de expansão em grafos orientados com pesos nas arestas. Tais estratégias devem proibir a formação de caminhos com "loops" e devem ser cuidadosas com a expansão dos nós. Em geral, algoritmos de busca para o menor caminho em grafos mantêm uma lista dos nós que estão expandidos (abertos) no momento. Tal lista pode se tornar extremamente numerosa devido a natureza do problema, que lida com objetos densos como imagens. Esta lista representa a parte do grafo já percorrida, e cada

nó x_j da lista tem um custo associado com o sub-caminho do ponto inicial x_A até x_j. Tais custos podem ser utilizados para "podar" caminhos que não sejam promissores, impedindo assim uma explosão da lista de nós expandidos. Esse tipo de estratégia exemplifica uma utilização de conhecimento heurístico [25].

Programação dinâmica

Programação dinâmica (PD) aplicada a deteção de bordas utiliza o mesmo grafo G definido para o método de busca heurística em grafos, e necessita igualmente de uma função de custo que avalie os caminhos em G. As principais diferenças são que PD necessita da construção total de G, não requerendo um conhecimento dos pontos iniciais e finais do contorno a ser detectado, o que é uma exigência do *A-algorithm*.

PD é um método de otimização baseado no **princípio da otimalidade**. O método procura pelo ótimo de uma função sem ter que interrelacionar todas as variáveis simultaneamente, ou seja, não avalia todas as variáveis do problema ao mesmo tempo. Isso é de grande valor para problemas que têm um espaço de soluções vasto, como é o caso daqueles que envolvem imagens. A principal idéia do princípio de otimalidade em relação ao problema de caminho ótimo em grafos é que qualquer que seja o caminho do nó inicial x_A a um nó qualquer x, existe um caminho ótimo de x até o nó final x_B. Ou seja, se o caminho ótimo de x_A para x_B passa por x, então os dois sub-caminhos $x_A \rightarrow x$ e $x \rightarrow x_B$ são também ótimos. Problemas que apresentam essas características seriais podem ser resolvidos por processos de otimização multi-estágios como PD.

Foi mostrado que o método para deteção por bordas baseados no *A-algorithm* pode ser mais eficiente que programação dinâmica quando se deseja encontrar o melhor contorno entre dois pontos pré-estabelecidos [18]. Além disso, ao contrário de programação dinâmica, busca heurística não exige construção completa do grafo G. No entanto, programação dinâmica tem a vantagem de buscar pelo caminho ótimo simultaneamente com vários pontos iniciais e finais. A questão de qual das abordagens é a mais eficiente também depende da função de custo utilizada e da qualidade das heurísticas do *A-algorithm*.

Capítulo 3

Metaheurísticas

Métodos Heurísticos sempre estiverem presentes na vida do homem indicando como resolver seus problemas diários, seja apresentando maneiras práticas de obter soluções, seja produzindo soluções de qualidade. Versões matemáticas vêm crescendo no seu escopo aplicativo na medida da quantidade de problemas difíceis que surgem e que precisam ser solucionados de maneira eficiente e satisfatória. Novas tecnologias heurísticas estão criando habilidades para resolver problemas que eram muito grandes ou complexos para algoritmos das gerações anteriores.

Um método heurístico, também conhecido como procedimento inexato, ou simplesmente heurística, é um conjunto de passos bem definidos para obter em tempo hábil soluções satisfatórias para uma instância de um dado problema, explorando de maneira inteligente o espaço de soluções à procura da melhor solução existente, mesmo não garantindo que esta solução seja visitada. Para tanto a heurística geralmente percorre um subconjunto bastante reduzido do espaço de soluções do problema, ganhando assim em velocidade. Mesmo não garantindo a obtenção da melhor solução possível, há métodos heurísticos avançados que garantem para alguns problemas difíceis boas aproximações em relação ao custo da melhor solução (*algoritmos aproximativos*). A aplicabilidade de métodos heurísticos justifica-se não só pela necessidade de se reduzir os tempos computacionais, mas também pela estrutura de certos problemas, que em alguns casos apresentam características que dificultam até mesmo a busca de uma solução viável para o problema.

Há também aqueles problemas que não são bem definidos (*ill-posed problems*) e para os quais nenhuma formulação conhecida se adequa totalmente. Nestes problemas geralmente é difícil identificar o que caracteriza soluções satisfatórias, além de não ser claro o que faz uma solução ser melhor do que outra, sendo ambas viáveis. Tais dificuldades certamente estão relacionadas com os objetivos, que podem ser múltiplos e conflitantes para problemas *ill-posed*. O tipo de soluções que será preterido e os critérios de comparação entre duas

soluções dependem exclusivamente de decisões tomadas por quem faz a modelagem desses problemas com características pouco claras. Esse tipo de atitude visa resolver o problema na prática, através de opções adotadas dentro da modelagem, que visam restringir as ambiguidades e os conflitos entre os objetivos. Um exemplo desse tipo de problemas *fuzzy* é o de detectar bordas em uma imagem. Experiências com métodos heurísticos na resolução de problemas *ill-posed* para imagens têm revelado resultados positivos [17].

O objetivo deste capítulo é apresentar e discutir os métodos heurísticos mais conhecidos e utilizados, assim dando suporte aos capítulos seguintes.

3.1 Conceitos

Métodos de busca que utilizam conhecimento heurístico trabalham sobre um espaço de soluções de um problema a procura de soluções de "qualidade". Tal qualidade geralmente é avaliada por uma função de custo, que é formulada de maneira a primar por certas características que são desejáveis em uma boa solução. Essas características geralmente são convertidas em fatores para a função de custo. Tais fatores são variáveis que assumem valores proporcionais à intensidade da característica detectada em uma solução.

Tome-se como exemplo genérico um dado problema P onde é desejável em suas soluções uma certa característica $\mathcal{C}_i$, que é medida pelo fator-variável c_i, tal que $0 \leq c_i(S) \leq 1$, com $c_i(S) \in \mathbb{R}$ para $S \in \mathcal{S}$, onde $\mathcal{S}$ é o espaço de soluções para P. Os valores de $c_i(S)$ são diretamente proporcionais ao grau de manifestação de $\mathcal{C}_i$ na solução S, ou seja, $c_i(S) = 0$ se $\mathcal{C}_i$ não se verifica em S, caso contrário $c_i(S) = 1$, e $0 < c_i(S) < 1$ para manifestações parciais de $\mathcal{C}_i$.

Seja uma função de custo F que avalia as soluções de $\mathcal{S}$ com base nas variáveis c_i. A intensidade da influência de cada c_i sobre o valor de F é determinada não só pelo valor que c_i assume para cada solução S, mas também pelo peso w_i, que depende do grau de influência que se deseja de c_i nas avaliações. O valor de w_i geralmente é arbitrado de acordo com uma ponderação entre todas as variáveis c_j, onde $1 \leq j \leq (\mathsf{n} = \mathsf{Total\text{-}de\text{-}variáveis})$, de maneira que a conjugação desses fatores faça F avaliar as soluções de $\mathcal{S}$ da maneira desejada. Em geral o valor de F corresponde a um somatório ponderado das variáveis c_i avaliadas sobre uma solução S:

$$F(S) = \sum_{j=1}^{\mathsf{n}} w_i c_i(S).$$

Quando $w_i < 0$ tem-se que a característica $\mathcal{C}_i$ é desejável em soluções de $\mathcal{S}$. Logo quando $\mathcal{C}_i$ é detectada em uma solução S, seu custo $F(S)$ será negativamente influenciado

pelo termo $w_i c_i(S)$, o que indica que S é uma solução de qualidade positiva em relação a $\mathcal{C}_i$. Já se $w_i > 0$ tem-se que $\mathcal{C}_i$ não é desejável no conjunto $\mathcal{S}$, pois se a característica $\mathcal{C}_i$ é detectada em uma solução S então o custo $F(S)$ cresce positivamente no termo $w_i c_i(S)$.

As variáveis também podem ser controladas a fim de se respeitar certas condições impostas ao problema. Tais relações restritivas que são estabelecidas entre as variáveis c_i são denominadas de **restrições** de um problema. Para o exemplo do problema P pode-se impedir predominâncias de certa característica $\mathcal{C}_k$ em detrimento de outra $\mathcal{C}_j$ através da seguinte restrição: $c_j - c_k > 0$. As restrições limitam o espaço $\mathcal{S}$, eliminado certas soluções que não podem atender às relações estabelecidas entre as características C_i.

Otimalidade

Seja $\mathcal{S}$ um espaço de soluções para um problema P e F uma função que avalia os custos das soluções de $\mathcal{S}$. A associação de F com $\mathcal{S}$ define uma medida da qualidade das soluções de $\mathcal{S}$. De posse de F, o objetivo dos métodos heurísticos é encontrar a solução de maior qualidade S^*, a qual apresenta o menor custo possível: $F(S^*) = \min_{S \in \mathcal{S}} F(S)$. Em geral os problemas solucionados por métodos heurísticos são aqueles para os quais não se conhece nenhum método analítico que obtenha S^* num tempo aceitável. Tais problemas são conhecidos como **problemas de otimização** e a solução de menor custo possível S^* é conhecida como **ótimo global** ou solução ótima.

Métodos heurísticos têm sido sugeridos para a resolução de problemas de otimização como uma maneira natural de explorar os conjuntos de soluções $\mathcal{S}$, que para alguns problemas pode chegar a ser infinito. Tratam-se de procedimentos iterativos que em sua maioria se movem de uma solução $S \in \mathcal{S}$ para outra S', repetindo esse passo enquanto parecer necessário. Geralmente os movimentos de S em S' são restritos a uma classe de modificações aceitáveis sobre S. Seja $S' = S \oplus m$ a notação para a operação onde se obtém S' a partir de uma modificação m sobre S. Defina-se M como o conjunto de todas modificações aceitáveis e $M_k \subset M$ um subconjunto de modificações. Com isso pode-se definir uma vizinhança $\mathcal{N}(S)$ como:

$$\mathcal{N}(S) = \{S' \mid \exists m \in M_k : S' = S \oplus m\}$$

Portanto, uma vizinhança $\mathcal{N}(S)$ relativa a uma solução S corresponde a um conjunto de soluções que pode ser obtido a partir de S através de um conjunto de modificações M_k. Pode-se fazer uma associação de um grafo $G = (\mathcal{S}, U)$ com os problemas de otimização. Cada solução $S \in \mathcal{S}$ corresponde a um nó de G. A aresta (S, S') pertence a U se e somente se $S' \in \mathcal{N}(S)$. Portanto as soluções visitadas consecutivamente em um método iterativo

de busca formará um caminho orientado em G. Pode-se notar que G pode ter um número infinito de nós, logo métodos heurísticos nunca chegam a construir G explicitamente, pois G é apenas uma maneira conveniente de representar o espaço de soluções do problema.

3.2 Busca local - BL

Minimização de uma função de custo através de uma busca local pertence a uma classe de heurísticas conhecidas como **métodos descendentes**, que são métodos que fazem decrescer o custo de suas soluções até quando for possível, baseando-se numa vizinhança $\mathcal{N}(S)$. Em cada passo de um algoritmo de busca local compara-se a solução corrente S com um conjunto V^*, constituído de uma ou mais soluções pertencentes $\mathcal{N}(S)$. Existem várias maneiras de se gerar um conjunto de soluções "candidatas" V^*. Uma delas é fazer V^* conter todas as soluções pertencentes a $\mathcal{N}(S)$. Tal estratégia só é possível quando $\mathcal{N}(S)$ é relativamente pequena e enumerável. Uma maneira mais simples de se construir V^* é simplesmente gerar uma solução aleatória pertencente a $\mathcal{N}(S)$, mas dessa maneira não se garante que $\mathcal{N}(S)$ será percorrido de maneira ótima. As buscas locais ou fazem $V^* \leftarrow \mathcal{N}(S)$, ou fazem V^* receber um subconjunto de $\mathcal{N}(S)$ tal que seja garantido que o subconjunto contenha a melhor solução de $\mathcal{N}(S)$. Gerado o conjunto V^*, a melhor solução é então escolhida entre S e as soluções de V^*, baseando-se em uma função de custos F. Se outro critério de parada não for estabelecido, a busca local pára somente quando melhores soluções não possam ser atingidas a partir de S (veja o algoritmo 3.1), ou seja, quando a solução corrente S é um **ótimo local** em relação a vizinhança $\mathcal{N}(S)$. Embora S não seja necessariamente a melhor solução possível, ao final do "loop" do algoritmo 3.1, ela será localmente ótima de modo que nenhum de seus vizinhos em $\mathcal{N}(S)$ tenha custo inferior ao de S.

Algumas vezes pode não ser uma questão trivial perfazer o passo 2b no algoritmo 3.1, que consiste em encontrar a melhor solução pertencente a V^*, o que poderá implicar na resolução do problema de otimização local $\min\{F(S) \mid S \in V^* \subseteq \mathcal{N}(S)\}$ através de um procedimento heurístico. Neste sentido o procedimento de busca geral pode ser entendido como uma **metaheurística**, que são heurísticas que usam algum outro procedimento heurístico a cada passo para encontrar um boa solução para um problema de otimização local em V^*. Esta heurística mais interna pode ser determinística (quando $V^* \leftarrow \mathcal{N}(S)$, a melhor solução de $\mathcal{N}(S)$ certamente será encontrada) ou não determinística (quando V^* recebe uma ou mais soluções aleatórias de $\mathcal{N}(S)$, assim não garantindo uma "descida" ao ótimo local).

O desempenho de uma busca local é determinado pela vizinhança $\mathcal{N}(S)$, pelos métodos de otimização local e pela estrutura do problema. Vizinhanças muito grandes podem

1. Obtenha uma solução inicial S.
2. Enquanto existe uma solução em $\mathcal{N}(S)$ não testada prossiga:
   (a) Seja $V^* \subseteq \mathcal{N}(S)$ um conjunto de soluções ainda não testadas.
   (b) Encontre a melhor solução $S' \in V^*$.
   (c) Se $F(S') < F(S)$, faça $S \leftarrow S'$.
3. retorne S.

Algoritmo 3.1: Esboço de uma de busca local.

prejudicar o desempenho de um algoritmo de busca, pois tais vizinhanças podem conter um vasto número de soluções que de alguma forma serão avaliadas a fim de se procurar por uma solução melhor que a solução corrente S. Por outro lado, grandes vizinhanças podem alcançar melhores soluções que estão "distantes" de S.

3.3 Simulated annealing - SA

Simulated Annealing é uma técnica de otimização que se utiliza de princípios mecânico-estatísticos para encontrar um mínimo ou máximo global de uma função de custo associada a um problema. O algoritmo originalmente descrito era voltado para simular o processo físico de resfriamento de sólidos [1, 14]. A técnica de *Simulated Annealing* consiste em iterativamente passar de soluções iniciais instáveis *(sólidos em altas temperaturas)* para soluções estáveis *(baixas temperaturas)*, onde estas seriam ótimos locais do problema. Para tanto, uma certa vizinhança das soluções é visitada segundo critérios probabilísticos avaliados sob a influência de um parâmetro T, que corresponderia ao grau de instabilidade das soluções *(Temperatura)*. Os valores de T decrescem durante a busca, o que provoca uma queda na instabilidade das soluções, o que corresponderia a uma redução das degradações sobre a solução corrente, as quais são provocadas pelas altas temperaturas. Tem-se, portanto, que no princípio de uma *simulated annealing* a busca apresenta passos descendentes e ascendentes, onde os ascendentes correspondem a degradações provocadas pelos altos valores de T. A medida que T tende a zero, a busca tende a ser descendente, sendo que em valores de T próximos ou iguais a 0 (zero) a busca passa a ser totalmente descendente e o algoritmo pode chegar a um ótimo local.

Considere o problema de minimizar a função $F(S) = \sum_i w_i c_i(S)$. A idéia básica de

simulated annealing consiste em tratar o sistema a ser otimizado como se fosse um sistema físico, cuja a energia é dada por $F(S)$. Isso significa que minimizar o sistema corresponde a encontrar seu estado mínimo de energia. O processo para encontrar tal estado consiste em lentamente "resfriar" o sistema, a partir de uma temperatura alta que decrescerá gradativamente até atingir o zero. O resfriamento deve ser lento o suficiente a fim de que o sistema não fique preso em um estado que corresponda a soluções de custo bem maiores que o da melhor solução, para em vez disso se encaminhar a um estado que tenha um custo bem "próximo" ao custo do ótimo global, ou mesmo a um estado que corresponda à própria solução ótima. Esse processo de resfriamento é simulado utilizando o método de *Metrópolis* via uma distribuição de *Boltzmann* [5, 28], onde $f = e^{-F/T}$ é a função que propriamente descreve o estado de equilíbrio termo-dinâmico para uma dada temperatura T.

A *simulated annealing* começa com a construção de uma solução aleatória S. Em seguida aplica-se em S certa modificação randômica $m \in M_k$, onde M_k é um conjunto de modificações aceitáveis que definem a vizinhança $\mathcal{N}(S)$ investigada pelo algoritmo. A variação de energia $\Delta F(S) = F(S \oplus m) - F(S)$ é então calculada. Se $\Delta F(S) < 0$, então a modificação m é validada, pois essa queda de energia é favorável ao resfriamento do sistema. Caso $\Delta F(S) \geq 0$, m é aceita com uma probabilidade dada por $e^{\Delta F(S)/T}$. Caso contrário a modificação não é validada.

A grande dificuldade de métodos de otimização local é que não existe maneira de "sair" de certos ótimos locais não desejáveis, pois buscas locais estritamente descendentes nunca se movem para novas soluções a não ser que seja na direção de soluções com menores valores de custo. Já a *simulated annealing* permite degradações em momentos ocasionais, tentando assim não se ver presa em algum ótimo local bem distante da solução ótima.

3.4 Busca tabu - BT

Uma desvantagem, anteriormente citada dos métodos descendentes, é a possibilidade de a busca ser detida em um mínimo local S^{Δ} em relação à função de custo F, onde o valor $F(S^{\Delta})$ seja bem superior ao custo $F(S^*)$ do ótimo global S^*. Para superar tal situação, algumas vezes é conveniente aceitar uma modificação m que transforme a solução corrente S em uma solução de custo superior, ou seja, aceitar uma solução S' vizinha de S tal que $F(S) < F(S' = S \oplus m)$, o que representa uma degradação na função objetivo. Como foi visto, em *simulated annealing* as degradações são aceitas probabilisticamente influenciadas pelos próprios valores de $F(S)$ e pelo valor de instabilidade do sistema dado pela "temperatura" T. Em busca tabu não existe nada equivalente ao parâmetro de temperatura, e além disso, a melhor solução $S' \in \mathcal{N}(S)$ (ou $S' \in V^* \subseteq \mathcal{N}(S)$) é

normalmente aceita, mesmo se $F(S') > F(S)$, a menos que outras condições proíbam a substituição. Tais condições são implementadas em busca tabu a fim de impedir, ou pelo menos reduzir ciclos na busca. Quando se aceita uma solução S' pior que S, eventualmente a busca pode retornar a S através de um movimento descendente que restaure o custo da solução corrente. A proibição de tais retornos indesejáveis pode ser realizada mantendo uma lista L_t de soluções já visitadas, conhecida como **lista tabu**. A figura 3.1 apresenta um exemplo de busca com ciclo, representado pelo caminho tracejado. Esse caminho tem ciclo pois depois de visitar a solução S_1, a busca não proíbe S_1 em uma lista tabu, logo, ao chegar em S_2 a busca volta novamente para S_1, pois $F(S_1) < F(S_2)$ e $S_1 \in \mathcal{N}(S_2)$. Mas se S_1 ficar *tabu*, então a busca prosseguirá de S_2 para S_3 e daí para S_4, que é uma solução de menor custo que o ótimo local S_1, além de S_4 ser um ótimo global para a função objetivo F traçada no gráfico da figura 3.1.

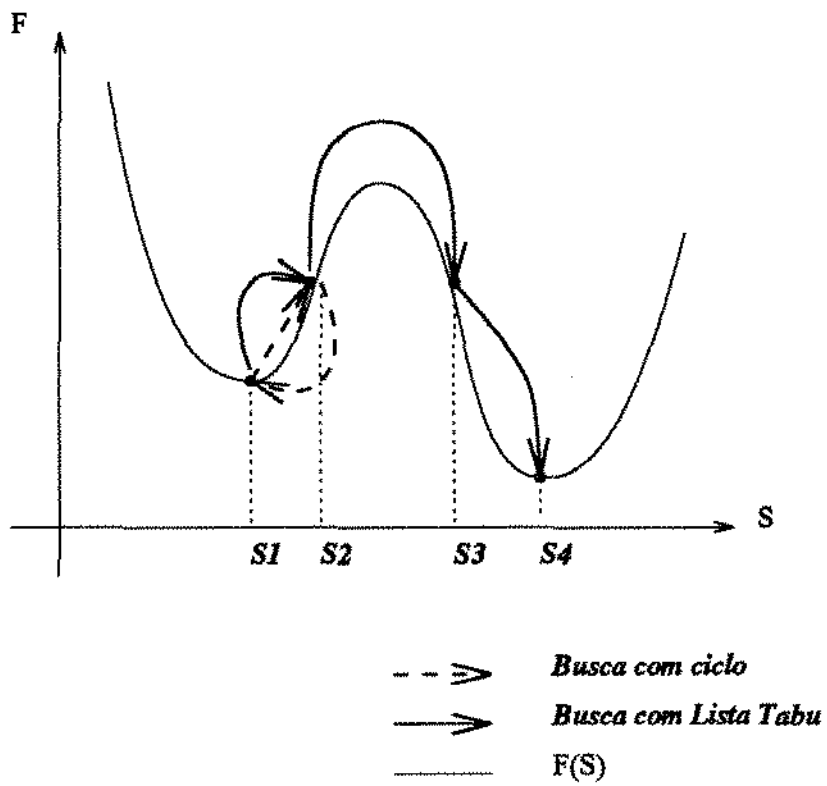


Figura 3.1: Exemplo de busca com ciclo e de *Busca Tabu* sem ciclo.

Quando a busca tabu encontra-se em uma solução S e uma lista de soluções $V^* \subseteq \mathcal{N}(S)$ é gerada, todas as soluções candidatas a membro do conjunto V^* são checadas a fim de garantir que não estão proibidas na lista L_t. Assim fica garantido que nenhuma solução será visitada duas vezes e, portanto, não ocorrerão ciclos na busca. Um dos inconvenientes de se manter L_t é o tamanho que tal lista pode chegar a ter em instantes avançados da busca, alcançando um número excessivo de elementos que exijam métodos de acesso

1. Escolha uma solução inicial $S \in \mathcal{S}$
2. $S^* \leftarrow S$; $i \leftarrow bestiter \leftarrow 0$; $L_t \leftarrow \emptyset$
3. fim $\leftarrow$ falso
4. Enquanto fim = falso faça
   (a) Se $(i - bestiter > k_{max})$
   (b) então fim $\leftarrow$ verdadeiro
   (c) senão
      i. $i \leftarrow i + 1$
      ii. Gere $V^* \subseteq \mathcal{N}(S) - L_t$
      iii. Escolha a melhor solução $S' \in V^*$
      iv. $S \leftarrow S'$
      v. Atualize L_t: $L_t \leftarrow L_t -$ (solução mais antiga deL_t) $\cup \{S\}$
      vi. Se $F(S') < F(S^*)$ então $S^* \leftarrow S'$ e $bestiter \leftarrow i$

Algoritmo 3.2: Esboço de uma *busca tabu.*

eficientes. Além disso, existem certas instâncias onde pode ser conveniente retornar a soluções já visitadas, de onde a busca pode continuar em direção diferente daquela tomada anteriormente. Por essas razões é comum se restringir a lista tabu L_t às t últimas soluções visitadas. Cada vez que uma solução é introduzida em L_t, ela toma o lugar da mais antiga inserida em L_t.

Apesar de lista tabu reduzir o ocorrimento de ciclos na busca, estes ainda são possíveis, pois uma lista de tamanho t impede ciclos de no máximo tamanho t. Portanto outros critérios de parada devem ser introduzidos. O procedimento pode cessar se não ocorrer melhoria na solução depois de k_{max} iterações. O valor de t também pode ser variado no decorrer do procedimento a fim de tornar a busca mais dinâmica. O algoritmo 3.2 traz um esboço do funcionamento de uma busca tabu.

Na prática não se mantém uma lista de soluções tabu, pois isso pode consumir muito espaço. Em vez disso, dada uma solução corrente S ou uma modificação m que será inserida na lista de proibição, dá-se o *status* tabu somente a alguns constituintes l_i de S ou m, onde os constituintes podem ser elementos ou partes ou algum subconjunto de características de S ou m. Em geral, tem-se uma coleção de condições: $l_i(S \text{ ou } m) \in L_i(i)$ $(i = 1, \ldots, \mathcal{I})$ que são conhecidas por **condições tabu**. Portanto diz-se que uma

solução S ou um movimento m estão proibidos se ele satisfaz uma das condições tabu.

Assim sendo, as condições tabu não somente proíbem o retorno a certas soluções já visitadas, mas também impedem alcançar soluções que se "assemelham" às soluções tabu, pois o que na verdade encontra-se proibido são movimentos ou partes de soluções, que não indicam univocamente a solução que em verdade está proibida. Assim soluções semelhantes a uma certa solução tabu serão ao mesmo tempo proibidas pelas mesmas condições tabu. Isso pode ser interessante, mas no entanto pode chegar a impedir a obtenção de certas soluções melhores, que equivocamente estejam tabu. Critérios de aspiração são instrumentos que permitem em algumas circunstâncias retirar o *status* tabu de um determinado movimento incluído na lista L_t, ou seja, às vezes pode ser vantajoso utilizar uma solução, mesmo ela sendo tabu. Tais critérios podem ser classificados em algumas classes. Considere L_t a lista tabu, S a solução corrente e S^* a melhor solução encontrada até o momento:

- **Aspiração por default**: Quando todos os movimentos são tabus e nenhum outro critério de aspiração adotado está satisfeito, pode-se selecionar o movimento menos tabu, se existir indícios de melhoria da solução atual.

- **Aspiração por objetivo**: Este é o critério básico de aspiração. Se um movimento $m \in L_t$, mas a atualização $S \leftarrow S^* \oplus m$ é tal que $F(S) < F(S^*)$ então m pode deixar a lista L_t.

- **Aspiração pela direção de busca**: Neste caso, a idéia é retirar o *status* tabu de um movimento que atualiza a solução numa direção que nas iterações anteriores tem produzido melhorias na função objetivo.

- **Aspiração por influência**: Este critério normalmente está ligado a noção de distância. Exemplo: um movimento de objeto entre duas caixas distantes uma da outra pode ser visto como um movimento de grande influência na solução associada, ao contrário desse mesmo movimento entre duas caixas próximas uma da outra. O que provoca grandes influências na solução deve ser usado em ótimos locais, enquanto aquele de pequenas influências deve ser usado quando a solução atual já foi considerada uma boa solução.

3.5 Busca em vizinhanças variáveis - *Variable neighborhood search* - VNS

O método de *Busca em Vizinhanças Variáveis - VNS* foi primeiramente proposto por *Mladenovic* e *Hansen* [19] como uma metaheurística simples e efetiva para a otimização de problemas combinatórios, como o do *Caixeiro Viajante*. A principal idéia da VNS é a utilização de vizinhanças que variam sistematicamente dentro de um algoritmo de busca local. Com isso é possível "superar" uma solução que seja um ótimo local para uma dada vizinhança $\mathcal{N}_k(S)$ sem necessariamente realizar movimentos ascendentes, simplesmente prosseguindo com a busca através de uma outra vizinhança $\mathcal{N}_l(S)$, a qual é distinta e não equivalente a $\mathcal{N}_k(S)$. Tal estratégia é razoável, pois sabe-se que se uma solução é um ótimo local para uma dada vizinhança $\mathcal{N}_k(S)$, então não necessariamente também será ótimo local para outra vizinhança $\mathcal{N}_l(S)$.

Veja o exemplo da figura 3.2, onde são apresentadas duas buscas: a que percorre uma vizinhança $\mathcal{N}_1$, representada pela linha contínua orientada, e a busca que explora $\mathcal{N}_2$, representada pela linha tracejada orientada. Pelo uso da vizinhança $\mathcal{N}_1$, a busca só consegue "enxergar" as soluções $S_1, S_2 \ldots S_8$, e portanto, usando $\mathcal{N}_1$ chegou-se ao ótimo local S_5. Mas se ao chegar a S_5, a busca prosseguir usando a vizinhança $\mathcal{N}_2$, então é possível chegar a uma solução de custo inferior ao de S_5, pois com $\mathcal{N}_2$ é possível chegar a S_{11} a partir de S_5.

Generalizando: Seja um vizinhança $\mathcal{N}_k$ que define relações de vizinhança em um espaço de soluções $\mathcal{S}$. $\mathcal{N}_k$ é baseada em um conjunto M_k de movimentos aceitáveis em $\mathcal{S}$: $\mathcal{N}(S) = \{S' \mid \exists m \in M_k : S' = S \oplus m\}$. Assim se M_k não define movimento que leva a solução S_1 na solução S_2, onde S_1 e $S_2 \in \mathcal{S}$, então S_1 e S_2 não são vizinhas para $\mathcal{N}_k$. Portanto, dependendo de M_k pode-se ter relações de vizinhança distintas entre as soluções de $\mathcal{S}$.

Seja uma função objetivo F que avalia as soluções de um espaço $\mathcal{S}$ e seja $\mathcal{N}_k$ uma vizinhança definida para $\mathcal{S}$. Seja S_{best} a solução de menor custo que é $\mathcal{N}_k$-vizinha de uma outra solução S, tal que $F(S) > F(S_{best})$. Seja $\mathcal{N}_l$ uma vizinhança equivalente a $\mathcal{N}_k$, com a exceção de que $\mathcal{N}_l$ não define movimento que leve S em S_{best}. Portanto S e S_{best} não são $\mathcal{N}_k$-vizinhos, e a solução $\mathcal{N}_l$-vizinha de S com menor custo inferior a $F(S)$, ou não existe ou tem custo maior que $F(S_{best})$. Pensando indutivamente: a solução obtida a partir de uma solução S_0, escolhendo-se sempre a solução $\mathcal{N}_k$-vizinha de menor custo inferior ao custo da solução corrente é denominada $\mathcal{N}_k$-(ótimo local) a partir de S_0. Ou seja, o movimento de busca estritamente descendente que parte de uma solução inicial S_0 pode terminar em ótimos locais distintos para vizinhanças $\mathcal{N}_k$ não equivalentes.

Portanto a vizinhança determina o comportamento da busca e o desempenho desta é

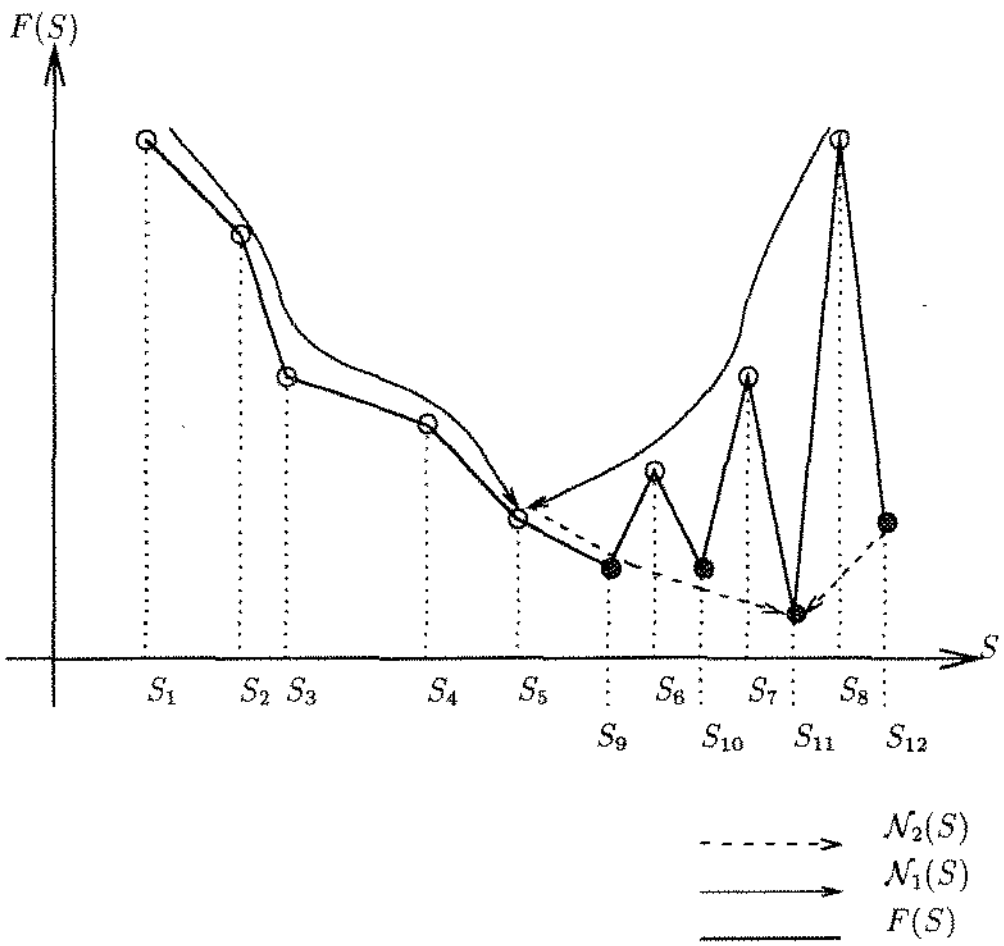


Figura 3.2: Exemplo de uma execução de uma VNS

1. Selecione o conjunto de estruturas de vizinhança $\mathcal{N}_k$, onde $1 \leq k \leq k_{max}$, os quais serão utilizados na busca
2. Encontre uma solução inicial S (randomicamente ou através de uma busca local)
3. $k \leftarrow 1$
4. Faça até $k = k_{max}$
   (a) Gere um ponto S' randomicamente na k-ésima vizinhança de S ($S' \in \mathcal{N}_k(S)$)
   (b) Aplique algum método de busca local onde S' é a solução inicial e armazene o resultado em S'
   (c) Se S' é melhor que S então $S \leftarrow S'$ e continue a busca com $\mathcal{N}_1(k \leftarrow 1)$, se não $k \leftarrow k + 1$

Algoritmo 3.3: Esquema de uma *busca em vizinhanças variáveis* - VNS.

determinado pela boa combinação entre a vizinhança adotada e a função objetivo. Isto se reflete na capacidade do algoritmo para alcançar boas soluções independentemente da solução inicial.

Algoritmo VNS Suponha que se dispõe de várias vizinhanças $\mathcal{N}_k$, onde $1 \leq k \leq k_{max}$. Temos que para uma dada solução S, $\mathcal{N}_k(S)$ corresponderia ao conjunto de todas soluções na k-ésima vizinhança de S.

Um simples algoritmo de busca local que se utiliza das vizinhanças $\mathcal{N}_k$, procederia pesquisando uma única $\mathcal{N}_k$, ou uma união de algumas $\mathcal{N}_k$'s. Já uma VNS procuraria utilizar as vizinhanças $\mathcal{N}_k$ de maneira sistemática, uma a cada vez, a fim de tornar a busca dinâmica e menos suscetível a ótimos locais de baixa qualidade. O algoritmo 3.3 traz um esquema básico de uma VNS.

3.6 Algoritmo genético - AG

Diferentemente das outras técnicas de busca tradicionais, algoritmos genéticos [12] utilizam informação de uma população de soluções para conduzir a busca por melhores soluções e não apenas informação de uma única solução. O algoritmo pode ser entendido como um método probabilístico de busca que é baseado no processo de evolução dos seres vivos. O método tenta imitar o desenvolvimento de novas e melhores populações de seres

entre diferentes espécies através dos tempos.

Durante o curso da evolução, organismos mais adaptados sobrevivem em detrimento de populações menos adaptadas. Esse fenômeno da natureza é conhecido como *seleção natural.* Indivíduos com maior aptidão ao ambiente em que vivem têm maior probabilidade de sobreviver e reproduzir, enquanto os menos adaptados vão sendo gradativamente eliminados. Isso significa que *genes* de indivíduos mais adaptados serão sucessivamente multiplicados a cada nova geração de seres. A combinação de boas características vindas de ancestrais mais adaptados pode produzir populações de indivíduos cada vez mais adaptadas aos seus ambientes.

Algoritmos genéticos simulam o processo de evolução dos seres vivos. A princípio eles tomam uma população inicial de indivíduos, a qual é sucessivamente modificada pela aplicação de operadores genéticos em cada reprodução. Em termos de otimização, cada solução é codificada como uma *string* de *bits*, que é denominada de *cromossomo*, onde cada cromossomo é composto de células denominadas *genes*. Cada cromossomo representa uma possível solução para um dado problema, onde cada uma de suas variáveis geralmente correspondem a um dado *gene* do cromossomo. Os valores de aptidão de um cromossomo são calculados com relação a uma dada função objetivo F_a, que em geral tem valores inversamente proporcionais à função de custo F. Os resultados da avaliação de F_a são então utilizados como parâmetro para medir as probabilidades de reprodução de cada cromossomo. Cromossomos mais adaptados, ou seja, cromossomos com maior avaliação da função objetivo terão mais chance de reproduzir do que os menos valorados.

A reprodução se dá através da troca de informação genética entre cromossomos mais adaptados, num procedimento denominado *crossover*. Este processo produz novas soluções que compartilham características herdadas de ambos os pais. Já a operação mutação, quando ocorre, é geralmente aplicada após o *crossover*, pela alteração de alguns *genes* no cromossomo (veja a figura 3.3). Tal operador, de caráter estocástico, visa a inserção de material genético dentro da população de cromossomos, prevenindo assim uma inadvertida perda de genes úteis causada pela seleção elitista de soluções para o *crossover*. Essa escolha das melhores soluções, as quais transmitem seus genes para as novas gerações, pode induzir uma padronização das próximas populações, que geralmente tendem a ser soluções bem uniformes em estágios avançados do algoritmo genético, apesar do uso da mutação.

As novas gerações podem tanto substituir toda antiga população, como também apenas os cromossomos menos adaptados. O algoritmo 3.4 sumariza o funcionamento de um algoritmo genético e a figura 3.4 apresenta um esquema do ciclo central do método.

Existem uma série pontos que têm forte influência no sucesso de um algoritmo genético:

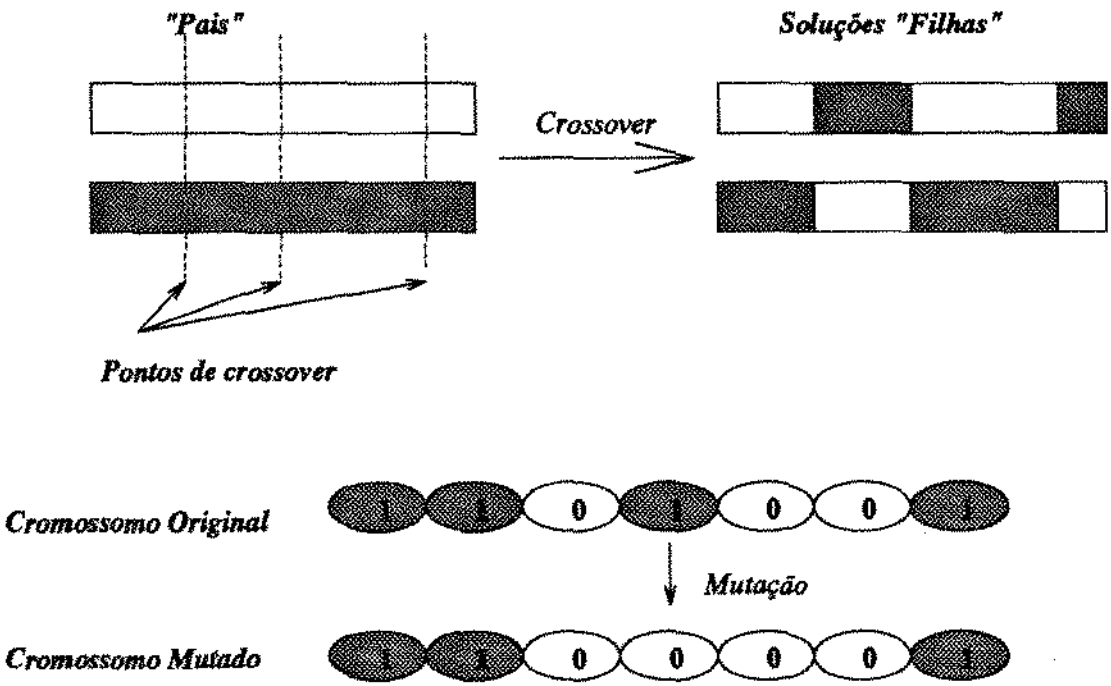


Figura 3.3: Exemplos de operadores genéricos tradicionais: *crossover* e mutação

---

1. Geração de uma população inicial
2. Avaliação da função objetivo de cada indivíduo da população
3. Repita até que uma solução satisfatória seja encontrada
   - ⇒ Selecionar pais a partir da população
   - ⇒ Recombinar pais para produzir filhos
   - ⇒ Avaliar função de aptidão dos filhos
   - ⇒ Substituir alguns ou todos indivíduos da população pelos filhos

---

Algoritmo 3.4: Esboço de um algoritmo genético.

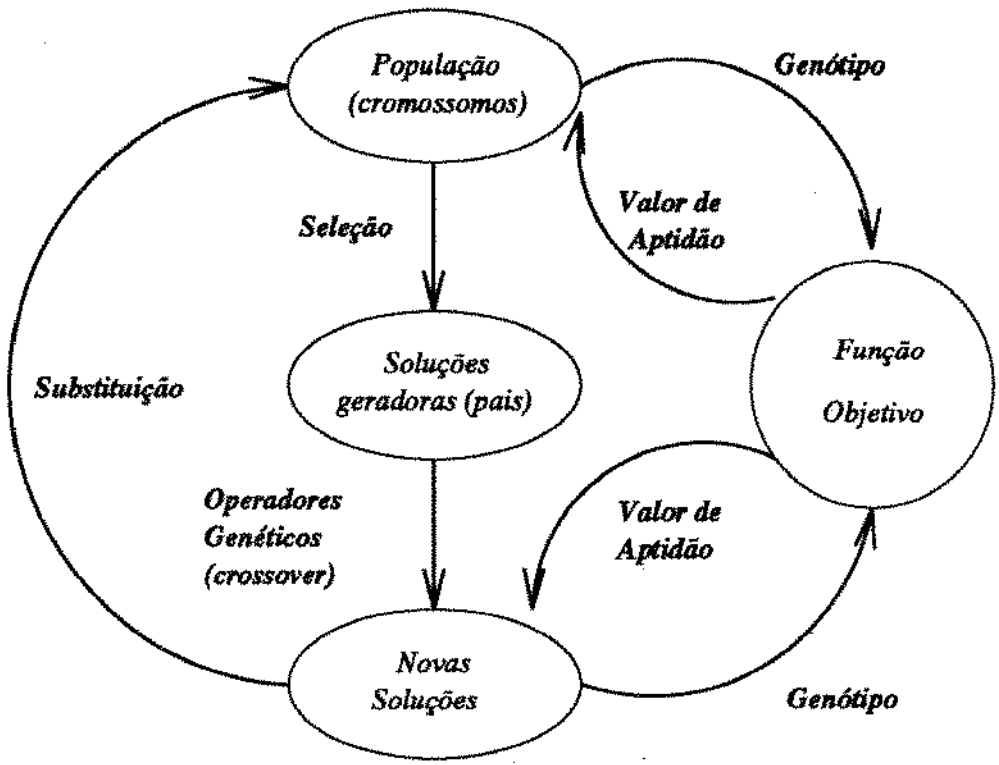


Figura 3.4: Ciclo genérico de um algoritmo genético

- *a representação das soluções*: a estrutura de dados que codifica as soluções deve facilitar o funcionamento dos operadores genéticos;

- *a geração da população inicial*: as soluções iniciais preferencialmente não devem induzir algumas características que posteriormente possam levar a população a ser um conjunto de soluções uniformes e sem chances de prover soluções mais fortes. Por outro lado, inicializar as soluções de forma totalmente randômica pode gerar uma população tão fraca e incapaz de ser conduzida pelos operadores genéticos a soluções com as características desejadas;

- *a composição de indivíduos para reprodução*: as soluções de uma geração que produzirão os indivíduos da próxima geração, devem ser soluções de boa qualidade, ou seja, a escolha dos "pais" das novas soluções devem preferir "pais" de menor custo e maior valor na função objetivo. Entretanto, deve-se ter cuidado para que uma estratégia totalmente elitista não venha sempre a escolher os mesmos "pais" entre as soluções de alta qualidade, pois tal estratégia, embora objetivando a qualidade, pode prejudicar a diversidade "genética" das soluções.

- *os operadores genéticos*: questões de como se deve combinar os "pais" para obter as novas gerações de soluções devem ser consideradas. Algumas estratégias devem

ser cuidadosamente tecidas, de preferência primando por "pinçar" o melhor da informação genética dos "pais" para a formação dos "filhos". O operador de mutação também deve ser utilizado de maneira a introduzir informação nova nas soluções, mas deve ser aplicado com moderação para que não acabe por prejudicar a qualidade das soluções mutadas.

Alguns critérios de parada são geralmente utilizados em algoritmos genéticos tais como: quando um tempo limite é excedido; ao verificar-se uma dominação da população por uns poucos indivíduos (ou seja, quando as soluções encontram-se uniformes em termos de custo ou de algumas características); ou ainda quando o melhor valor da função objetivo manter-se constante por um certo número de iterações.

Capítulo 4

Avaliação de uma deteção de contornos

Antes de confeccionar um algoritmo de minimização de custos para deteção de contornos, é necessário que se tenha uma noção precisa do que se deseja considerar como uma borda. Um ponto $p \in I$ deve apresentar algumas características para que seja considerado um **ponto de borda** de uma solução de deteção de contornos S para a imagem I. Um ponto de borda deve ser fronteira entre duas regiões dissimilares, deve formar uma linha contínua com os demais pontos de borda da solução S e não deve formar uma linha espessa com seus pontos de borda vizinhos.

Cada linha formada pelos pontos de borda de uma solução tem que ser única para uma dada fronteira da imagem, e vice-versa. Não é desejável que uma solução S apresente mais de uma linha de pontos de borda que representem a mesma fronteira. Além disso, os pontos de borda não devem se apresentar isolados na solução S. Cada ponto de borda deve pertencer a uma linha contínua e "fina" que corresponda a uma fronteira na imagem.

Tais considerações são diretrizes básicas para a confecção de uma função de custo. Tal função deve avaliar a qualidade de uma solução S, levando em conta cada uma das diretrizes. Estas, por sua vez, devem ser ponderadas, a fim de que tipos de qualidades aparentemente desejáveis em uma solução não inviabilizem a obtenção de uma boa imagem de bordas. Por exemplo: é desejável que uma solução apresente linhas de borda *não-fragmentadas*, no entanto, todo final de linha é um ponto de borda *fragmentado*, pois trata-se de um ponto de descontinuidade na linha. Seguindo cegamente essa diretriz que rejeita a fragmentação, poder-se-á chegar a uma solução vazia, ou seja, sem nenhum ponto de borda encontrado.

As diretrizes citadas correspondem à definição de bordas proposta por *Tan et al.* [27],

que tenta ser de âmbito geral, restringindo-se às bordas que são evidentes na imagem. Ou seja, tenta caracterizar bordas que separam significativamente regiões dissimilares, não se preocupando com os contornos que podem ser captados por processos cognitivos de alto nível. Neste capítulo é apresentada a função de avaliação baseada na caracterização de uma borda ideal de *Tan et al.*, função esta que é utilizada pelos métodos heurísticos apresentados nos próximos capítulos.

4.1 Função de Custo

Para representar uma solução para o problema de deteção de contornos será utilizada uma imagem com dimensões equivalentes a da imagem original:

$$S = \{s(x,y); 1 \leq (x,y) \leq N\},$$

onde S corresponde a uma matriz binária, com seus elementos $s(x,y)$ valendo 1 se o ponto (x,y) é considerado uma borda. Se $s(x,y) \neq 1$, então o ponto (x,y) não é considerado borda. Os pontos que apresentam $s(x,y) = 1$ serão denominados de pontos de borda. Já os segmentos formados por pontos de borda serão chamados de **linhas de borda**.

Os fatores de custo propostos por [28] são: dissimilaridade (c_d), número de pontos de borda (c_e), fragmentação (c_f), curvatura (c_c) e espessura (c_t). Dada uma solução S, o custo no ponto p é dado pelo seguinte somatório:

$$F(S,p) = \sum_i w_i c_i(S,p),$$

onde w_i são pesos tais que $w_i \geq 0$ e $0 \leq c_i \leq 1$ para $i \in \{c,d,e,f,t\}$. Estendendo-se a notação de custo para regiões: sejam dois pontos p e $q \in S$. Seja $S[p,q]$ uma sub-imagem de S, que é induzida pelos pontos $p = (i_p, j_p)$ e $q = (i_q, j_q)$. Seja um ponto $r = (i_r, j_r) \in S$, então $r \in S[p,q]$ se e somente se $\min(i_p, i_q) \leq i_r < \max(i_p, i_q)$ e $\min(j_p, j_q) \leq j_r < \max(j_p, j_q)$. O cálculo do custo de $S[p,q]$ é importante quando se realizam operações de busca local na região induzida por dois pontos p e q. Para tanto, o custo relativo da região $S[p,q]$ é dado pelo seguinte somatório:

$$F(S[p,q]) = \sum_{p \leq r < q} \sum_i w_i c_i(S,r).$$

Já o custo total de uma solução S é dado por:

$$F(S) = \sum_{0 \leq p < N} F(S, p).$$

onde N é o total de pontos de S. Sejam duas soluções S e S'. O custo incremental $\Delta F(S, S')$ é dado por:

$$\Delta F(S, S') = F(S') - F(S).$$

O custo incremental também pode ser dado pela seguinte equação:

$$\Delta F(S, S') = \sum_i w_i \Delta c_i(S, S'),$$

onde

$$\Delta c_i(S, S') = \sum_{p \in K} [c_i(S', p) - c_i(S, p)],$$

onde K é um conjunto de pontos pertencentes a I, tais que: $\forall k \in K$, tem-se que $s'(k) \neq s(k)$, ou então k é um dos "vizinhos-dependentes" de um ponto r, tal que $s'(r) \neq s(r)$. Entende-se que o ponto $k = (i_k, j_k)$ é "vizinho-dependente" de um ponto $r = (i_r, j_r)$, se $i_k = i_r \pm \delta$ ou $j_k = j_r \pm \delta$, onde δ é a distância mínima entre dois pontos de uma solução que garante a independência entre o estado de um ponto e custo do outro. Esta inter-dependência entre pontos próximos em uma distância de até δ é provocada por certos fatores de custo c_i, que consideram a configuração dos pontos vizinhos de um ponto $p \in S$, ou seja, consideram os valores da janela $S(p, 2\delta + 1 \times 2\delta + 1)$ para calcular o valor do termo de custo de $c_i(S, p)$. Isso pode ser verificado em alguns fatores proposto por *Tan et al.*, como a espessura c_e e a curvatura c_c.

Dissimilaridade

Seja I uma imagem, S uma solução para o problema de deteção de contornos em I, e p um ponto em I. Seja $c_d(p)$ o fator de custo de dissimilaridade para o ponto p. O valor de $c_d(p)$ baseia-se no valor de $s(p)$ e no grau de aptidão de p para ser um ponto de borda. Tal grau de aptidão é dado pela *matriz de gradiente D:*

$$D = \{D(x, y); 1 \leq (x, y) \leq N\},$$

onde $0 \leq D(x,y) \leq 1$. O cálculo de D é semelhante aos dos operadores de gradiente (veja seção 2.2.1), com algumas diferenças: as máscaras de convolução não são todas "quadradas" e a operação realizada em cada ponto (x,y) influencia não só o valor da célula $D(x,y)$, mas também os valores de gradiente em algumas células vizinhas de (x,y). O cálculo do gradiente D é baseado em um conjunto de *estruturas de borda* E_i, onde $1 \leq i \leq 12$. As máscaras E_i são semelhantes à máscaras de convolução. Veja a figura 4.1, onde são apresentadas as 12 estruturas de borda que são utilizadas nas implementações deste texto. As estruturas E_i possuem duas regiões: R_1 e R_2, que definem os dois lados de uma linha de borda. Os círculos correspondem aos pontos que formam a linha de borda sugerida pela estrutura. O ponto central da estrutura está indicado pelo círculo negro. As máscaras de convolução equivalentes às estruturas E_i teriam pesos -1 nas células da região R_1 e pesos 1 nas de R_2, ou vice-versa, pois o gradiente D só tem valores não negativos baseados no módulo da diferença entre a soma dos valores da função da imagem em cada região R_1 e R_2.

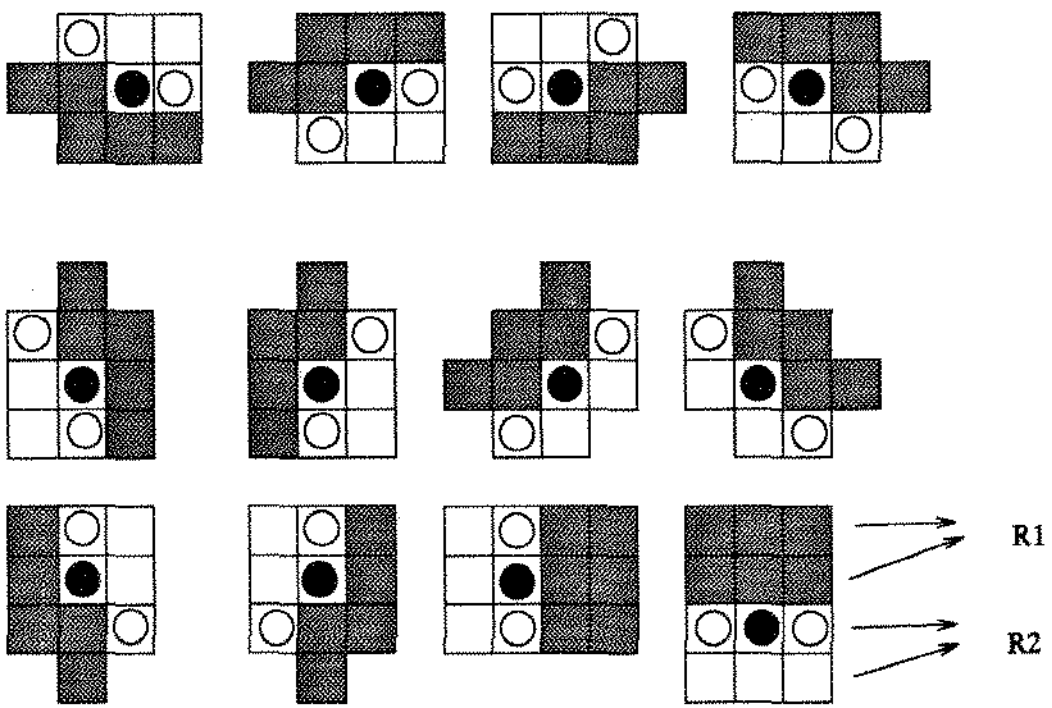


Figura 4.1: Estruturas de Borda.

As 12 estruturas de borda E_i competem entre si pelo "melhor" (maior) gradiente encontrado em cada ponto da imagem. A estrutura vencedora indica a melhor configuração de borda para aquele ponto, ou seja, se existir linha de borda que passa pelo ponto, certamente a borda terá as características do contorno inferido pela estrutura vencedora. O algoritmo 4.1 apresenta os passos para a obtenção da matriz D.

1. Inicialize todos os pontos de D com 0 (*zero*)
2. Para cada ponto p de I faça:
   (a) Para cada *estrutura de borda* E_i, onde $i \in [1, 12]$ faça:
      i. Sobreponha E_i em I, centrando E_i em p
      ii. Some os valores de $f(q)$ (nível de cinza de q), para todo ponto q que corresponda a R_1
      iii. Some os valores de $f(q)$, para todo ponto q que corresponda a R_2
      iv. Faça d_i receber a diferença entre as duas somas
      v. Divida d_i pelo valor do número de pontos de R_1 (= tamanho de R_2)
   (b) Atribua à variável *best* o valor do índice do maior d_i
   (c) Realize a operação *supressão não-maximal* com base na estrutura E_{best} e no ponto p
   (d) Se a *supressão não-maximal* obtiver valor superior ao de d_{best}
      então não há alteração na matriz D
      senão $D(q) \leftarrow D(q) + (d_{best}/3), \forall q \in$ (*linha de borda* definida por E_{best})
3. Mapeie os valores de D para o intervalo $[0, 1]$

Algoritmo 4.1: Procedimento para cálculo da matriz de gradiente.

A operação de *supressão não-maximal* [4] do passo 2c do algoritmo 4.1 corresponde a um algoritmo de busca na vizinhança de um ponto $p \in S$. Dado que se tem posse da estrutura de borda E_{best}, que é a estrutura melhor adequada à região centrada em p (E_{best} é a estrutura que produziu o gradiente de maior valor: d_{best}, veja o algoritmo 4.1), então desloca-se tal estrutura em várias direções: direita, esquerda, acima, abaixo ou diagonal. As direções dos deslocamentos variam de acordo com as características da linha de borda sugerida por E_{best} (veja a figura 4.2). Feitos estes deslocamentos, calcula-se as diferenças entre as somas dos pontos correspondentes às regiões R_1 e $R_2 \in E_{best}$. Se for obtido um valor de "gradiente" maior que o valor de d_{best}, ou seja, se for obtido um valor superior ao que foi obtido quando se sobrepôs a estrutura E_{best} em p, então a possível linha de borda não deve passar por p, e sim por outro ponto em sua vizinhança. Por isso, nesse caso, conservam-se inalterados os valores da matriz D. Em caso contrário, ou seja, quando o valor de d_{best} não é superado por nenhuma soma resultante da *supressão não-maximal*, então o valor de $(d_{best}/3)$ é acrescentado aos valores correntes de $D(q)$, onde q são os pontos correspondentes à *linha de borda* sugerida por E_{best} quando sobreposto em p (*os pontos q correspondem aos círculos indicados nas estruturas da figura 4.1*). Depois de obtida a matriz de gradiente D, o passo 3 do algoritmo 4.1 mapeia os valores de D para o intervalo $[0,1]$. O mapeamento, proposto por [27], é feito com base num valor limiar ξ, que deve ser escolhido de acordo com as características da imagem e com as bordas que se deseja detectar. Tal mapeamento corresponde a uma normalização linear:

Mapeamento da matriz D para $[0,1]$*:*

1. Para cada $p \in D$ faça:

   (a) Se $D(p) \leq 1.8\xi$
   então $D(p) \leftarrow D(p)/(2\xi)$
   senão $D(p) \leftarrow 0.9 + (0.1/(255 - 1.8\xi))(D(p) - 1.8\xi)$

O diagrama da figura 4.3 mostra um esquema de funcionamento de um método genérico para minimização de custos para deteção de contornos. Percebe-se que o cálculo do gradiente é feito no início do método, para em seguida dar-se início a heurística de busca por minimização. O resultado do gradiente é então armazenado na matriz D a fim de que possa ser utilizado pela função de custo nas avaliações das soluções.

Espessura

Em deteção de contornos, geralmente é desejável que uma linha de borda não seja "espessa". O fator de custo relacionado com a espessura das linhas de borda é representado

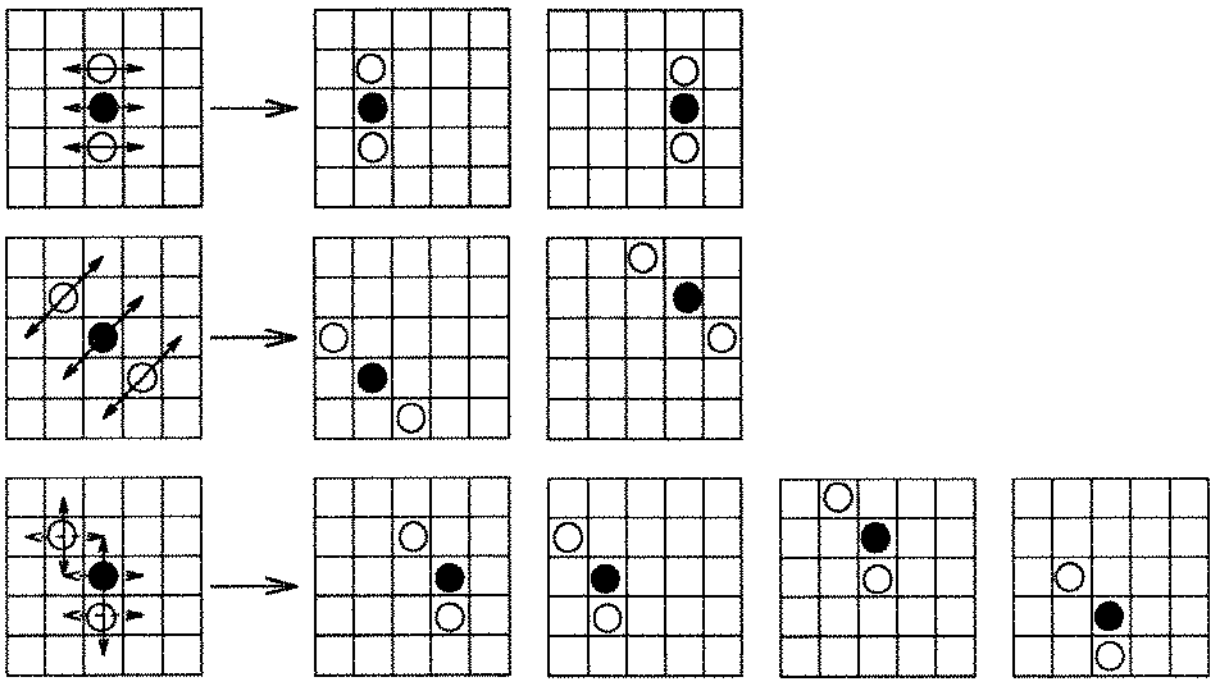


Figura 4.2: Exemplos da operação de *supressão não-maximal*.

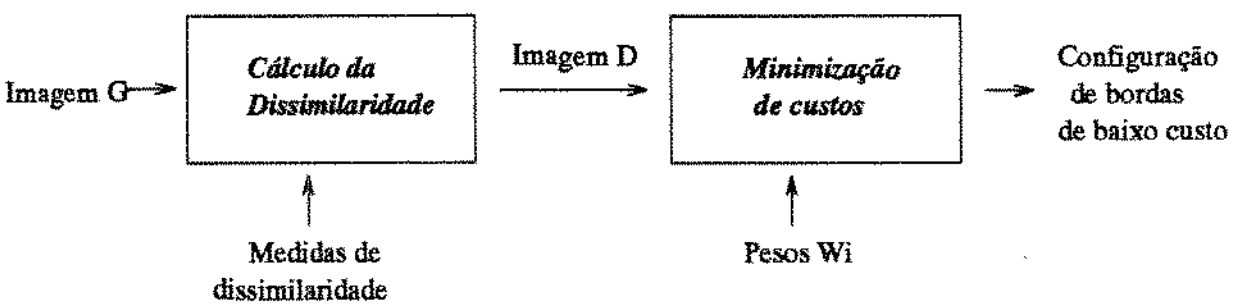


Figura 4.3: Diagrama do funcionamento de uma abordagem de minimização de custos para deteção de contornos

Exemplos dos ângulos que podem ser formados por estruturas de borda são apresentados na figura 4.4:

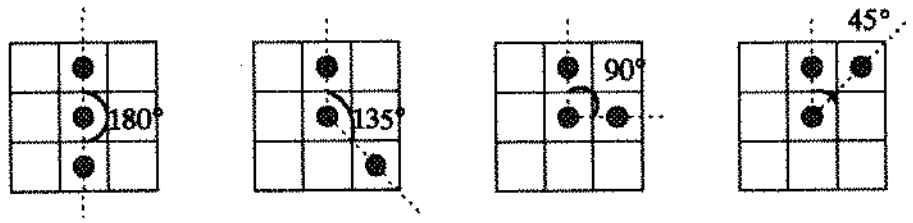


Figura 4.4: Exemplos da formação de ângulos em linhas de borda.

Para investigar as ocorrências dos ângulos 180° e 135°, as máscaras da figura 4.5 podem ser comparadas sobre a região $S(p, 3 \times 3)$:

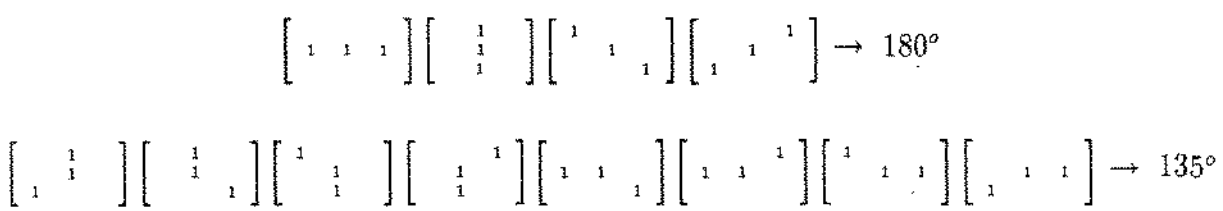


Figura 4.5: Exemplos de máscaras para deteção de linhas de borda que formam ângulos de 180° e 135°.

Os ângulos menores ou iguais a 90° são identificados por "exclusão": se não foram detectados nem ângulos de 180° nem ângulos de 135°, logo é direto que o ângulo formado no ponto é menor ou igual a 90°.

Número de Pontos de Borda

Para impedir um número excessivo de pontos de borda na imagem solução S, adota-se um fator de custo representado por c_e, o qual penaliza a existência de pontos de borda. Dessa maneira, impede-se a formação de linhas de borda pequenas, geralmente relacionadas a ruídos na imagem I. Além disso, o fator de custo para dissimilaridade c_d favorece a deteção de pontos de borda que tenham $D(p)$ diferente de zero. A intenção do fator de custo c_e é exatamente suprimir essa tendência característica de c_d.

$$c_e(p) = \begin{cases} 1, & \text{se } s(p) = 1, \\ 0, & \text{senão.} \end{cases}$$

Pesos e Interdependências entre os fatores de custo

Para as implementações deste texto, temos que os pesos w_i, que ponderam os fatores de custo c_i, têm seus valores iguais aos propostos por *Tan et al.* [28]: $w_d = 2.0$, $w_e = 1.0$, $w_c = 0.5$, $w_f = 2.0$ e $w_t = 4.51$. A computação da função custo é dada pela árvore de decisão da figura 4.6, que apresenta uma maneira eficiente e otimizada para o cálculo do custo de um ponto p.

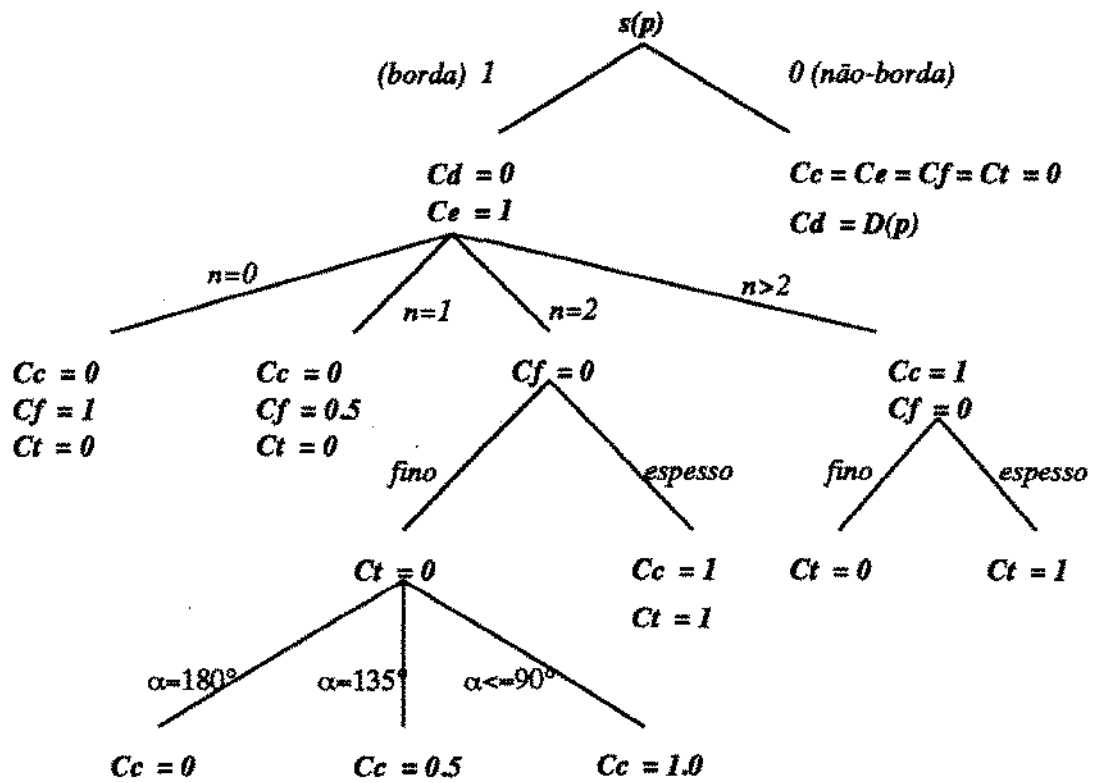


Figura 4.6: árvore de decisão para computação do custo $F(S, p)$

Capítulo 5

Métodos na literatura

No capítulo anterior foi descrita a definição de bordas de *Tan et al.* [28, 27], e sua resultante função de avaliação para soluções do problema. Pode-se notar que o custo de um dado ponto de uma solução é calculado levando vários fatores em consideração, desde ordem geométrica, como a curvatura, como também de ordem estrutural, como a espessura. A maioria dos fatores considera o contexto da vizinhança de um ponto para inferir o custo que "realmente" reflete a subsolução verificada naquela região.

Vários métodos heurísticos foram baseados na função de avaliação de *Tan et al.* [27]: busca local [27], *simulated annealing* [28] e algoritmo genético [3]. *Acton e Bovik* [2] também utiliza um método de minimização de custos para o problema de deteção de contornos baseado em critérios semelhantes aos propostos pela definição de bordas de *Tan et al.* Tais trabalhos tentam superar os problemas dos algoritmos tradicionais através da utilização de uma definição de borda mais robusta, que se propõe a atingir um escopo maior de tipos de contorno.

O objetivo deste capítulo é introduzir dois dos algoritmos heurísticos para deteção de contornos: a *simulated annealing* de *Tan et al.* [28] e o algoritmo genético de *Bhandarkar et al.* [3]. Tais métodos são objeto de implementação e servem de base de comparação para a nova abordagem proposta no capítulo 6. Também há modificações propostas sobre esses métodos, visando trazer melhorias em termos de tempo computacional e qualidade das soluções finais.

5.1 A Simulated Annealing de *Tan et al.*

A abordagem de *simulated annealing - SA* para deteção de contornos de *Tan et al.* [28], que será denotada pela sigla SAt, baseia-se no diagrama de um algoritmo genérico de minimização de custos da figura 4.3. O método apresenta duas fases distintas: a primeira corresponde ao cálculo do gradiente (veja capítulo 4) e a segunda fase é uma busca por minimização da função de custos utilizando a meta-heurística *simulated annealing* (veja capítulo 3 para métodos de busca por minimização e capítulo 4 para a descrição da função de avaliação de *Tan et al.*).

Como descrito na seção 3.3, a SAt parte de uma solução inicial com valores randômicos. A cada iteração, a solução corrente é percorrida num total de $W \times H$ passos (W e H são as dimensões da imagem), onde cada passo corresponde a operações sobre uma janela centrada em um ponto da solução. Ou seja, em uma iteração, cada ponto da imagem é tratado em um dado passo. Cada ponto é tratado uma e somente uma vez a cada iteração. A maneira como se percorrem os pontos pode ser randômica ou sequencial (da esquerda para a direita e de cima para baixo). A cada passo do algoritmo uma nova solução S' é gerada a partir da solução S do passo anterior. S' é construída com base em estratégias de transformação $\Im_j$ aplicadas sobre a solução corrente S. As estratégias $\Im_j$ exploram a vizinhança escolhida pelo método de maneira não-determinística (característica própria de uma SA). Entenda-se aqui, que um procedimento determinístico é um procedimento nem aleatório nem randômico. Dada a solução corrente e o ponto $p \in S$ onde será aplicada a busca, escolhe-se uma estratégia $\Im_v$ entre as cinco estratégias disponíveis para agir em $S(p, 3 \times 3)$ e gera-se a nova solução S', a qual difere de S somente na janela $S'(p, 3 \times 3)$. A maneira como se escolhe a estratégia $\Im_v$ consiste em um sorteio ponderado por certas probabilidades selecionadas empiricamente. Na verdade, as estratégias $\Im_j$ correspondem ao conjunto de movimentos que definem a vizinhança $\mathcal{N}_{\mathrm{SAt}}(S, p)$ adotada pela abordagem, onde p é o ponto referencial para a aplicação das transformações $\Im_j$ sobre S. A união das soluções que podem resultar de cada uma das transformações $\Im_j$, para $1 \leq j \leq 5$, correspondem ao conteúdo do conjunto $\mathcal{N}_{\mathrm{SAt}}(S, p)$. Como a escolha da estratégia é realizada por um sorteio, não se pode garantir uma exploração eficiente de $\mathcal{N}_{\mathrm{SAt}}(S, p)$, e portanto, não se pode garantir que a melhor solução pertencente a $\mathcal{N}_{\mathrm{SAt}}(S, p)$ seja avaliada e comparada com a solução corrente. No entanto, este comportamento da SAt não implica que agindo de maneira diferente faça o método mais eficiente do que é. De fato, experiências foram realizadas no sentido de explorar as soluções de $\mathcal{N}_{\mathrm{SAt}}(S, p)$ de maneira mais completa, ou seja, apanhando-se grande parte de todas as soluções que podem resultar das estratégias $\Im_j$, para $1 \leq j \leq 5$, e escolhendo a de menor custo para compará-la com a solução corrente. Os resultados obtidos mostraram-se bem inferiores aos da implementação da SAt original com sorteio da estratégia $\Im_j$, o que mostra a importância do caráter estocástico

dentro da SAt. Segue uma breve descrição das estratégias $\Im_j$ quando aplicadas sobre uma solução S na região centrada em um ponto p:

- Estratégia $\Im_1$: alternação $(1 \leftrightarrow 0)$ simples do ponto $s(p)$. S' é gerada simplesmente complementando o valor de $s(p)$ na solução S.
- Estratégia $\Im_2$: dupla alternação de pontos. Aqui alterna-se o valor de $s(p)$ e depois escolhe-se randomicamente um ponto q pertencente à janela $S(p, 3 \times 3)$, tal que $q \neq p$, e então também alterna-se o valor de $s(q)$.
- Estratégia $\Im_3$: Um "*shift*" de um ponto. Esta estratégia baseia-se em modificações na região $S(p, 3 \times 3)$, onde a solução resultante S' é idêntica a S a não ser pela região $S'(p, 3 \times 3)$, que têm seus valores indicados pelas transformações da figura 5.1. Se o conteúdo de $S(p, 3 \times 3)$ for igual a uma das estruturas à esquerda das produções da figura 5.1, então o conteúdo da região $S'(p, 3 \times 3)$ é dado pelas estruturas indicadas à direita das produções. Percebe-se que as duas primeiras produções da figura 5.1 têm duas opções para a região resultante $S'(p, 3 \times 3)$, as quais são escolhidas com probabilidades de 50% cada uma. As demais produções são diretas. Caso $S(p, 3 \times 3)$ não seja equivalente a nenhum das estruturas à esquerda das produções, então a estratégia $\Im_3$ falha e a solução S passa inalterada para o próximo passo.
- Estratégia $\Im_4$: Esta estratégia é idêntica a $\Im_3$, a não ser pelas produções, que agora são aquelas indicadas na figura 5.2.
- Estratégia $\Im_5$: Nesta estratégia a configuração da janela $S(p, 3 \times 3)$ é transformada de forma que a configuração resultante $S'(p, 3 \times 3)$ seja randômica.

A escolha da estratégia vencedora $\Im_v$, $v \in \{1, 2, 3, 4, 5\}$, é feita de forma não determinística. A probabilidade de uma estratégia $\Im_j$ ser escolhida é representada por $P(\Im_j)$, onde é recomendado por *Tan et al.* [28] que seus valores sejam: $P(\Im_1) = 200/1024$, $P(\Im_2) = 300/1024$, $P(\Im_3) = 200/1024$, $P(\Im_4) = 200/1024$ e $P(\Im_5) = 124/1024$. O algoritmo 5.1 descreve o procedimento, onde cada ponto da imagem S é submetido a um passo de busca pelas transformações $\Im_j$. A solução resultante S' é escolhida probabilisticamente com base no custo incremental $\triangle F(S) = F(S') - F(S)$ e na instabilidade das soluções dada pelo parâmetro de temperatura T_i (ver seção 3.3). Esta variável só muda de valor ao final de cada varredura total (iteração) da imagem, ou seja, depois que toda imagem S foi percorrida e submetida às operações de transformação $\Im_j$, é que, então, T_i é incrementado de acordo com sua função de "resfriamento", a qual *Tan et al.* [28] recomenda que seja da forma $T_i = c/log(ki + 2)$, onde i é o índice da iteração corrente, c é uma constante dada por $c = \min(w_f, 2w_c, 2w_f + w_d - w_e)$, com w_f, w_c, w_e, w_d sendo os pesos

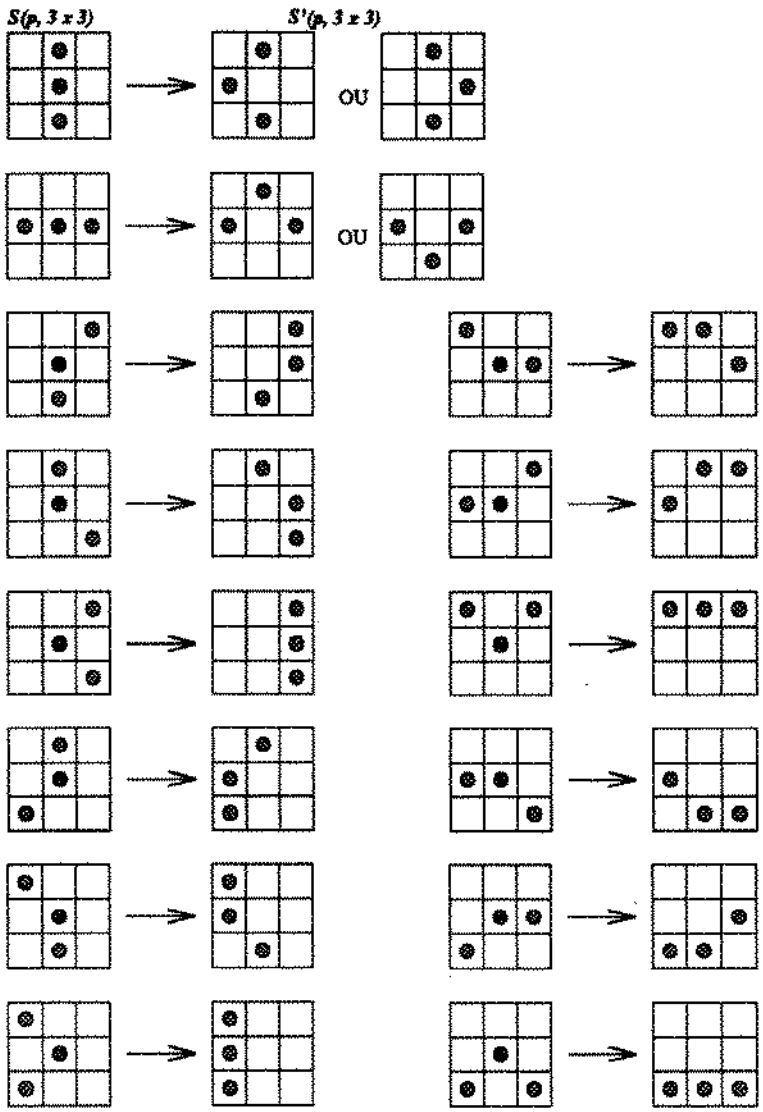


Figura 5.1: Transformações $\Im_3$

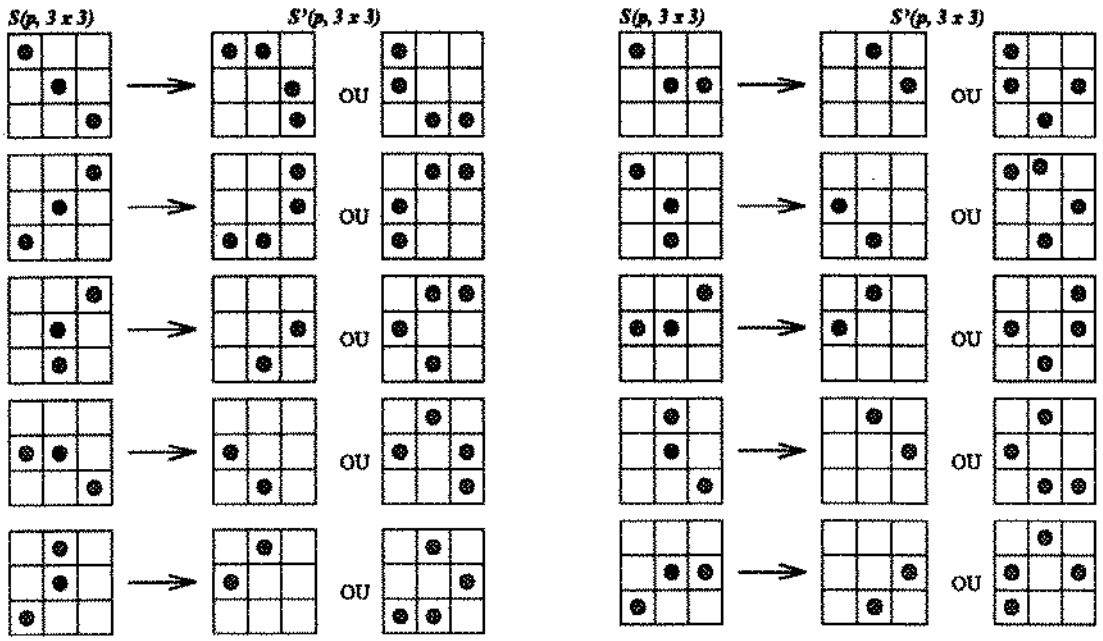


Figura 5.2: Transformações $\Im_4$

da função de custo utilizada pelo algoritmo, que é a função apresentada no capítulo 4. A constante k é escolhida de maneira a fazer com que a temperatura chegue a valores próximos a zero depois da iteração 100. Uma função de resfriamento desse tipo garante a convergência assintótica $\lim_{k \to \infty} T_i = 0$. No entanto a convergência do resfriamento logarítmico é bem lenta. Assim como em *Bhandarkar et al.* [3], constatou-se através de experiências computacionais que uma função geométrica de resfriamento apresenta melhores resultados que a logarítmica para a SAt. A melhor função geométrica avaliada é dada por $T_i = T_0(1-\alpha)^i$, onde $\alpha = 0.01$ e $0.3 \leq T_0 \leq 0.4$. Funções de temperatura que decrescem mais rapidamente, ou que têm temperatura inicial menor que $T_0 = 0.3$, comprometem os resultados finais, obtendo soluções de qualidade inferior.

5.2 O algoritmo genético de *Bhandarkar*

O método de *Bhandarkar et al.* [3], assim como a SAt, também consta das duas fases do diagrama da figura 4.3. A abordagem, que será denotada por AGdb, mantém uma população de soluções binárias, que estão em constante modificação, devido à atuação dos operadores genéticos tradicionais como o *crossover* e a mutação (veja cap. 3), além de outros operadores direcionados ao problema de deteção de contornos, como o *condicionamento randômico*. O funcionamento do algoritmo segue o ciclo da figura 3.4, adaptado às

1. Construa uma solução inicial randômica para a imagem I e armazene em S
2. $i \leftarrow 0$
3. Avalia-se o valor de T_i
4. Para cada ponto $p \in S$, onde p é escolhido randomicamente ou sequencialmente, faça:
   (a) Selecione uma estratégia de transformação $\Im_j$ com probabilidade $P(\Im_j)$, onde $j \in \{1, 2, 3, 4, 5\}$.
   (b) Aplique a estratégia escolhida $\Im_v$ ao ponto p em S, e armazene a solução resultante em S'
   (c) Calcule o custo incremental $\Delta F(S, S')$
   (d) Se $\Delta F(S, S')$ é negativo
      i. então
         Faça $P(S') \leftarrow 1$.
      ii. senão
         Faça $P(S') \leftarrow exp(-\Delta F(S, S')/T_i)$, senão faça $P(S') \leftarrow 0$.
   (e) Faça $S \leftarrow S'$ com probabilidade $P(S')$, senão mantém-se S inalterada
5. Se o critério de parada não foi atingido então
   (a) $i \leftarrow i + 1$
   (b) Volte para o passo3

Algoritmo 5.1: *Simulated annealing* para deteção de bordas.

características do problema atacado.

Cada solução, ou cromossomo, está associado como um custo que é obtido pela aplicação do árvore de decisão da figura 4.6. O cálculo da função de aptidão (*fitness*) $A()$ é feito tomando o pior custo em toda a população da geração corrente:

$$A(S_j) = (F(S_{\text{worst}}) - F(S_j))^k,$$

onde *worst* denota o índice da pior solução e k é um fator que dá a proporção geométrica entre os valores de aptidão e as diferenças de custo entre as soluções, ou seja, quanto maior for k, maior será a diferença de aptidão entre duas soluções que tem custos diferentes. O método propõe que nas primeiras iterações o valor de k deva ser da ordem de 2 e que se torne cada vez maior até que $k = 5$.

A seleção dos cromossomos para reprodução, ou seja, para o *crossover*, é feita através de uma *roulette wheel*, que utiliza os valores de aptidão de cada cromossomo. Uma *roulette wheel* corresponde a um sorteio de elementos S_j, ponderado por probabilidades $P(S_j)$, onde $1 \leq j \leq N$ e $N = \textit{total-de-elementos}$, tais que $\sum_j P(S_j) = 1$. No caso da seleção de cromossomos para reprodução, tem-se que uma dada solução S_k tem a seguinte probabilidade de ser selecionada por uma *roulette wheel*: $P(S_k) = A(S_k)/\sum_{j=1}^{N} A(S_j)$. O algoritmo 5.2 traz um esboço do funcionamento do AGdb.

Seguem as descrições dos operadores propostos por *Bhandarkar et al.* para seu algoritmo AGdb:

Crossover Simples

O operador de *crossover* utilizado pelo AGdb, simplesmente realiza uma troca de porções contínuas das duas soluções geradoras. Dois pontos são escolhidos randomicamente. A janela determinada por esses pontos será a região que será permutada entre as duas soluções. Veja um exemplo genérico do *crossover* simples de *Bhandarkar et al.* na figura 5.3.

Mutação Simples

Tal mutação consiste em simplesmente alternar o valor de $s(p)$, onde p é o ponto escolhido para mutar em uma dada solução S.

1. Gera-se uma população inicial de soluções randômicas S_j, onde $j \in [1, 2, \ldots N]$
2. Repita até que uma solução satisfatória seja encontrada
   (a) Avalia-se $F(S_j)$ para cada S_j
   (b) Para i de 0 até *total-de-crossovers-por-iteração*
      i. *best* $\leftarrow$ índice da solução de menor custo $F(S_j)$, para $j \in [1, 2, \ldots N]$
      ii. *worst* $\leftarrow$ índice da solução de maior custo $F(S_j)$, para $j \in [1, 2, \ldots N]$
      iii. *cond* $\leftarrow$ *best*
      iv. Aplica-se *Condicionamento randômico* em S_{cond}
      v. Recalcula-se $F(S_{best})$
      vi. Para cada S_j, calcule o valor de aptidão $A(S_j)$ (= *fitness*):
      $$A(S_j) \leftarrow (F(S_{worst}) - F(S_j))^2$$
      vii. Seleciona-se soluções geradoras S_1 e S_2, com sorteio ponderado pelos valores de aptidão $A(S_j)$
      viii. Realiza-se *crossover simples* sobre S_1 e S_2, gerando-se os "filhos" S_1' e S_2'
      ix. Realiza-se *mutação inteligente* em S_1' e S_2', segundo uma dada probabilidade P_{Mut}
   (c) Substituir a população corrente pelas novas soluções geradas, conservando na população a solução S_{best}

Algoritmo 5.2: Esboço do AGdb.

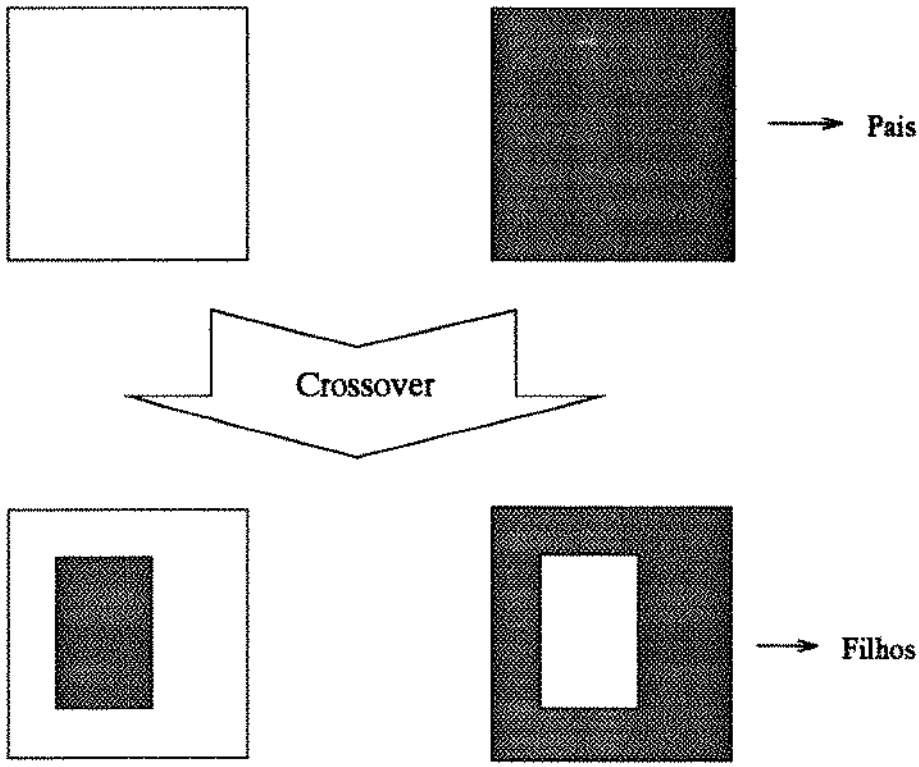


Figura 5.3: *Crossover* Simples

Mutação Inteligente

A mutação inteligente, quando aplicada sobre uma dada solução S, leva em consideração a configuração da janela $S(p, 3 \times 3)$, onde p é o ponto escolhido para mutar.

O operador utiliza certas estruturas de borda E_j, de dimensões 3×3, para as quais há um conjunto estabelecido de mutações: $\mathcal{M}_j = \{M_{j1}, M_{j2}, \ldots M_{jk}\}$. Tais mutações M_{jc} são também estruturas de borda semelhantes a E_j, a não ser por um certo deslocamento da linha de borda descrita por E_j. Veja a figura 5.4, que mostra um exemplo de uma estrutura de borda e seu conjunto de mutações inteligentes, acompanhadas de suas probabilidades.

Dado que será feita mutação inteligente sobre o ponto $p \in S$, primeiramente deve-se verificar se a janela $S(p, 3 \times 3)$ é equivalente a alguma das estruturas E_j. Se isto for verdade para $j = k$, então um sorteio ponderado é feito sobre o conjunto $\mathcal{M}_k$, e uma mutação escolhida M_{ke} é efetivada. A ponderação do sorteio é feita com base em probabilidades pré-estabelecidas. O valor da probabilidade da mutação M_{kc} é calculado com base na estrutura E_k e em quanto a mutação melhora a configuração de $S(p, 3 \times 3)$. Seja o exemplo em que será aplicada mutação inteligente em um ponto $p \in S$. Suponha-se que a região em $S(p, 3 \times 3)$ seja equivalente a estrutura de borda E_k. Portanto, dado que existe E_k tal que $E_k \equiv S(p, 3)$, então existe um conjunto de mutações $\mathcal{M}_k$ aplicáveis em

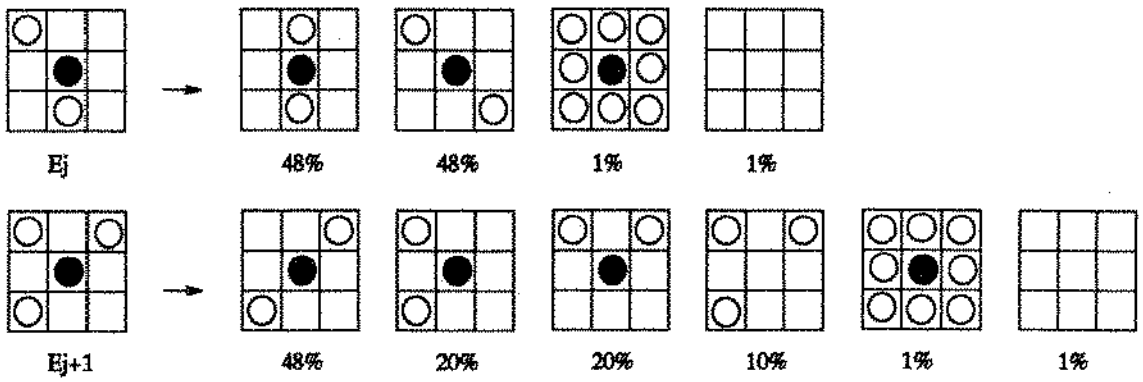


Figura 5.4: Estruturas de borda E_j e E_{j+1} com seus respectivos conjuntos de mutações M_j e M_{j+1}.

$S(p, 3 \times 3)$. As probabilidades de cada mutação $M_{kj} \in \mathcal{M}_k$ são tais que se a mutação M_{kj} gera uma linha de borda não retorcida e a mutação $M_{kj'}$ produz uma linha de borda fragmentada então a probabilidade de M_{kj} ser escolhida é maior que a probabilidade de $M_{kj'}$. Veja as probabilidades das mutações da figura 5.4. Observe que sempre existe uma probabilidade diferente de 0 para as mutações que produzem estruturas vazias ($s(p) \leftarrow 0(zero), \forall p \in S(p, 3 \times 3)$), ou cheias ($s(p) \leftarrow 1, \forall p \in S(p, 3 \times 3)$).

É importante ressaltar que a efetivação de uma mutação M_{je} não é subordinada à avaliação do custo incremental obtido com a mutação, tratando-se assim de um procedimento não determinístico.

Condicionamento Randômico - *Random Engineered Conditioning*

As soluções para deteção de contornos são objetos de grandes dimensões. Notadamente o espaço de soluções de tal problema é imenso, pois cada solução é uma matriz binária de $h \times w$ elementos, totalizando $2^{h \times w}$ soluções viáveis para o problema, onde h e w são as dimensões da imagem de entrada. Ainda mais, o problema de deteção de contornos é um problema *ill-posed* [17], ou seja, sua (*melhor*) solução ou não existe, ou não é única.

Dado que o desempenho de um algoritmo genético é determinado pelo tamanho da população de soluções, tal tipo de abordagem aplicada a deteção de contornos parece tender a resultados insatisfatórios. A limitação de memória das máquinas não permite a manutenção de um largo número de soluções. Além disso, um algoritmo genético que processa tal número de imagens levaria um tempo considerável, dada a velocidade atual das máquinas.

A eficiência do AGdb está fortemente ligada ao operador de *condicionamento randômico EC* (*Engineered Conditioning*). Este é responsável pela aceleração na convergência do

algoritmo, pois a abordagem convencional, sem aplicação do operador *EC*, obtém resultados pobres. O funcionamento do *EC* corresponde a uma espécie de catalizador da convergência.

O operador consiste em condicionar 5% da solução de menor custo, aplicando-lhe um algoritmo de busca local. Isto é feito em conjunto com os demais operadores (*crossover*, mutação). Na implementação deste texto, o *EC* é realizado a cada vez que o operador de *crossover* é executado. Portanto, quando se tem uma população de 100 imagens, e a cada geração 80% das soluções são submetidas ao *crossover*, então haverão 40 chamadas ao *crossover*, e igualmente 40 chamadas ao *EC* por geração. Ou seja, $40 \times 5\% = 200\%$ da melhor solução será condicionada pelo *EC* a cada geração.

O *EC* original funciona como o operador de mutação inteligente, sendo que a modificação proposta pelo operador em uma solução S_j só é efetivada se provoca melhoria no custo $F(S_j)$.

Estratégia Elitista

A estratégia elitista consiste em impedir que a melhor solução de uma população "morra" de uma geração para outra. Garante-se, assim, que o material genético de tal solução, notadamente superior, seja mantido na população por mais tempo, contribuindo para a formação de novas soluções possivelmente ainda melhores.

Adaptação dos operadores genéticos

Tal estratégia consiste em alterar os valores das probabilidades de mutação (P_{Mut}) e de *crossover* (*total-de-crossovers-por-iteração*) durante a execução. O objetivo é adequar suas aplicações de acordo com a evolução do algoritmo. A cada certo número de iterações (no caso da versão implementada neste texto: 20 iterações), os valores das probabilidades de mutação e de *crossover* são alterados. Aumenta-se a probabilidade de mutação, objetivando-se mais inserção de material genético na população. Já a probabilidade de ocorrer *crossover* é reduzida. Segundo as experiências de *Bhandarkar et al.* [3], com o decorrer do algoritmo a capacidade do *crossover simples* de produzir boas soluções fica reduzida. Isso se dá por que a partir de um certo momento do algoritmo as soluções são em sua maioria de boa qualidade. Por isso tem sentido a redução do número de *crossovers* e o aumento da ocorrência de mutações, a fim de que haja uma maior diversificação das soluções correntes.

5.2.1 Novos operadores para o AGdb

Experiências foram realizadas objetivando melhorar o desempenho de alguns operadores utilizados pelo AGdb. Observou-se que o operador de *crossover simples* do AGdb não incrementava a qualidade das soluções por ele produzidas. As experiências relatadas neste texto visaram, sobretudo, confeccionar operadores de *crossover* diferenciados, que utilizassem conhecimento para direcionar o processo. Alguns fatores, tais como o custo dos pontos de uma solução, foram utilizados para ponderar a troca de informação entre as soluções geradoras ("pais"). Também foi investigado o caráter sequencial do *crossover simples* do AGdb, através da implementação de um *crossover* randômico, que permutava pontos aleatórios das soluções geradoras.

Crossover Randômico

O *crossover* randômico é realizado da seguinte forma: sejam S_1 e S_2 duas soluções que foram escolhidas como geradoras, ou seja, que serão os "pais" de duas novas soluções S'_1 e S'_2. Para cada ponto p de S_1 e S_2, é feito um sorteio para saber-se para quem dos dois filhos S'_1 e S'_2 será transmitido o valor de $s_1(p)$. O filho restante herdará o valor de $s_2(p)$. A sequência a seguir esclarece a operação na forma de um algoritmo:

1. Sorteia-se S_1 e S_2
2. Para cada ponto p faça:
   (a) Com probabilidade de 0.5 faça:
   $$s'_1(p) = s_1(p) \text{ e } s'_2(p) = s_2(p)$$
   (b) senão faça:
   $$s'_1(p) = s_2(p) \text{ e } s'_2(p) = s_1(p)$$

Crossover Inteligente

O *crossover* inteligente consiste em fazer duas soluções geradoras transmitirem o melhor de si para um único filho. Formalizando: sejam duas soluções S_1 e S_2 escolhidas para gerarem uma nova solução S'. O valor de $s'(p)$ será igual a $s_1(p)$ ou igual a $s_2(p)$. A escolha é feita deterministicamente, com base nos valores de custo $F(S_1, p)$ e $F(S_2, p)$. Veja a figura 5.5. Tal escolha pode também levar em consideração não só os valores de $F(S_1, p)$ e $F(S_2, p)$, como também dos custos de pontos vizinhos de p. Com base nestas

idéias três modelos de *crossover* inteligente foram propostos: o pontual, o sobre grades e o sobre regiões.

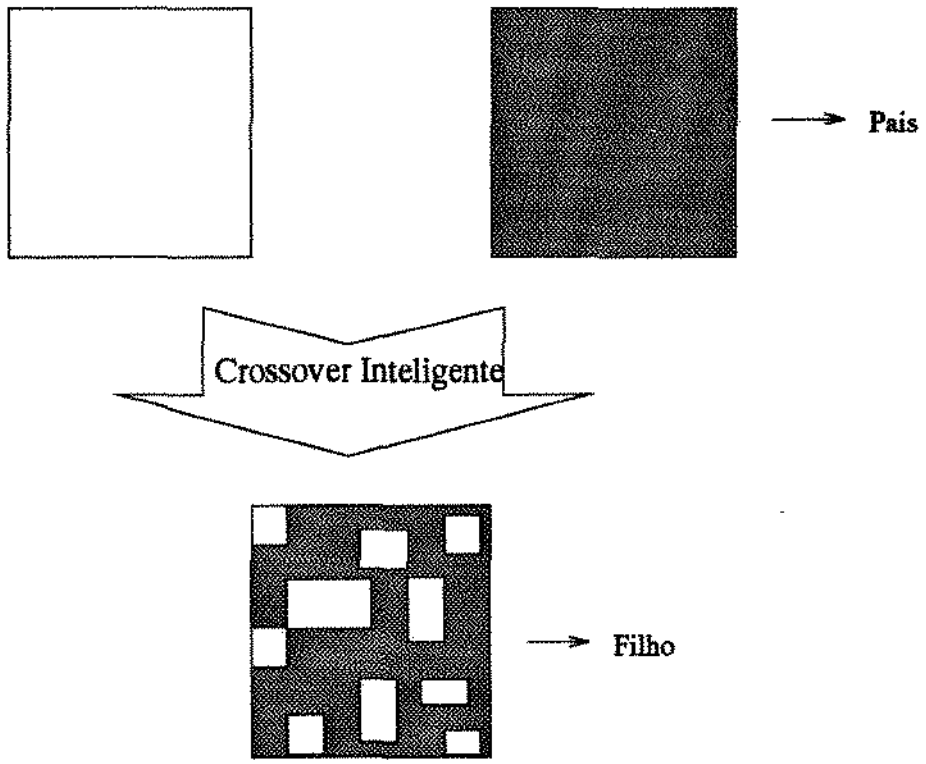


Figura 5.5: Crossover Inteligente

Crossover Inteligente Pontual. Sejam duas soluções geradoras S_1 e S_2. Através do *crossover* inteligente pontual, a solução "filha" herda os pontos de menor custo dos "pais", ou seja, $s'(p) \leftarrow s_1(p)$ se $F(S_1, p) < F(S_2, p)$, caso contrário, $s'(p) \leftarrow s_2(p)$. Tal operador não apresentou grandes resultados, pois o caráter aleatório das trocas provoca uma fragmentação das linhas de borda na solução gerada.

1. Sorteia-se S_1 e S_2
2. Para cada ponto p faça:
   (a) Se $s_1(p) < s_2(p)$:
   então $s'(p) = s_1(p)$
   senão $s'(p) = s_2(p)$

Crossover Inteligente sobre Grades. Seja R um reticulado definindo sobre as soluções geradores S_1 e S_2, a de um total de η pontos randômicos, os quais estão delimitados pelos pontos extremos S_1 e S_2. Portanto R formará $(\eta+1)^2$ regiões retangulares. Sobrepõe-se R às imagens geradoras S_1 e S_2. Formar-se-ão as sub-imagens S_{11}, S_{12}, $\ldots S_{1(\eta+1)^2}$, pertencentes à imagem S_1, e as sub-imagens S_{21}, S_{22}, $\ldots S_{2(\eta+1)^2}$, pertencentes à imagem S_2.

Agora aplica-se o *crossover inteligente sobre grades* em S_1 e S_2, trabalhando da mesma maneira que no *crossover inteligente pontual*, sendo que em vez de avaliar os valores dos custos $F(s_1(p))$ e $F(s_2(p))$, avalia-se os valores dos custos $F(S_{1j})$ e $F(S_{2j})$, referentes às sub-imagens S_{1j} e S_{2j}, para $j \in \{1,2,\ldots(\eta+1)^2\}$.

Sendo S' a imagem gerada, se $F(S_{1j}) < F(S_{2j})$, então $s'(p) \leftarrow s_1(p)$, para $\forall p \in S_{1j}$. Senão $s'(p) \leftarrow s_2(p)$, para $\forall p \in S_{2j}$.

Crossover Inteligente sobre Regiões. Sejam duas soluções geradoras S_1 e S_2. Seja a imagem *diferença* definida da seguinte maneira:

$$S_{1:2} = \{s_{1:2}(p); 1 \leq p < N\},$$

onde N é o total de pontos. Temos que para todo ponto p, tal que $0 \leq p < N$, se $s_1(p) = s_2(p)$ então $s_{1:2}(p) \leftarrow 0$, caso contrário $s_{1:2}(p) \leftarrow 1$.

No *crossover* inteligente sobre regiões, a imagem $S_{1:2}$ é calculada a fim de ser utilizada para determinar regiões *contínuas em valores de 0 (zero)*. Ou seja, regiões contínuas em que $s_{1:2}(p) = 0$, o que, pela definição de $S_{1:2}$, corresponde a encontrar as regiões contínuas de pontos do tipo $s_1(p) \neq s_2(p)$. Tais regiões são representadas por $S_{1:2(k)}$, que correspondem a conjuntos de pontos pertencentes a S_1 e S_2 tais que:

$$S_{1:2} = \bigcup_k S_{1:2(k)} \text{ e } S_{1:2(k')} \bigcap S_{1:2(k'')},$$

onde k é o número de regiões detectadas e $1 \leq k' < k'' \leq k$. $S_{1:2}$ é definida de tal maneira, que um ponto p é tal que $p \in S_{1:2(k)} \leftrightarrow \exists q(q \neq p, (q \in S_{1:2}(p, 3\times 3)), q \in S_{1:2(k)} e s_{1:2}(q) = 0)$. O objetivo de se encontrar as regiões $S_{1:2(k)}$ é obter uma segmentação da imagem diferença, onde as regiões detectados corresponderiam às porções da imagem em que diferem S_1 e S_2. Veja a figura 5.6, onde está representada uma imagem diferença e sua respectiva segmentação em regiões $S_{1:2(k)}$. Nesta, para efeito de visualização, tem-se os pontos da região $S_{1:2(0)}$ representados por ".", em vez do índice próprio de sua região que seria o 0 (zero).

As regiões $S_{1:2(k)}$ terão a mesma funcionalidade que as sub-imagens S_{jk} têm para o *crossover inteligente sobre grades*. Ou seja, as regiões $S_{1:2(k)}$ serão as unidades de comparação para o *crossover inteligente sobre regiões*. Observe que os pontos de S_1 e S_2 pertencentes à região $S_{1:2(0)}$ possuem custos iguais, pois trata-se de uma região idêntica para S_1 e S_2: $s_1(p) = s_2(p)$ para $\forall p \in S_{1:2(0)}$.

$S_{1:2}$													Regiões Segmentadas											
1	1	0	0	1	0	0	0	0	0	0	0	0	.	.	1	1	.	2	2	2	2	2	2	2	2
1	1	1	1	1	1	1	0	0	0	0	0	0	.	.	.	.	.	.	.	2	2	2	2	2	2
0	0	1	1	1	1	1	1	1	0	0	0	0	3	3	.	.	.	.	.	.	.	2	2	2	2
0	0	0	1	1	1	1	1	1	1	1	1	0	3	3	3	.	.	.	.	.	.	.	.	.	2
0	0	1	1	1	1	1	1	1	1	1	1	1	3	3	.	.	.	.	.	.	.	.	.	.	.
0	1	1	1	1	0	0	1	1	1	0	0	0	3	.	.	.	.	4	4	.	.	.	5	5	5
1	1	1	0	0	0	0	1	1	0	0	0	0	.	.	.	4	4	4	4	.	.	5	5	5	5
1	1	1	0	0	0	1	1	1	0	0	0	0	.	.	.	4	4	4	.	.	.	5	5	5	5
1	1	1	1	0	1	1	1	1	1	0	0	0	.	.	.	.	4	.	.	.	.	.	5	5	5
0	0	1	1	0	1	1	1	1	1	1	0	0	6	6	.	.	4	.	.	.	.	.	.	5	5
0	0	1	1	1	1	1	1	1	1	1	1	0	6	6	.	.	.	.	.	.	.	.	.	.	5
1	1	1	0	0	0	1	1	1	1	1	0	0	.	.	.	7	7	7	.	.	.	.	.	5	5
1	1	1	0	0	1	1	1	1	1	1	0	0	.	.	.	7	7	.	.	.	.	.	.	5	5

Figura 5.6: Imagem diferença e as regiões contínuas de zeros obtidas

O algoritmo para calcular as regiões $S_{1:2(k)}$ necessita de apenas uma varredura sobre $S_{1:2}$, feito da esquerda para a direita e de cima para baixo. Segue o algoritmo 5.3 para calcular as regiões $S_{1:2(k)}$.

Condicionamento Ponderado

Experiências foram realizadas sobre o operador *EC*. No Condicionamento do AGdb, a solução que é condicionada sempre é a solução corrente de menor custo. No Condicionamento Ponderado, aqui proposto, a solução que é condicionada a cada chamada do operador *EC* não é necessariamente a solução de menor custo. Em vez desta, a solução a ser condicionada é escolhida dentre todas as soluções da geração corrente, com base em um sorteio ponderado por probabilidades equivalentes aos valores de aptidão (*fitness*) das soluções. Assim, evita-se uma super-valoração de uma só solução, que no caso do AGdb, é a melhor solução corrente.

A escolha da solução a ser condicionada pode ser feita por geração ou por passo. Por geração entende-se que a solução a ser condicionada é escolhida no início de uma iteração

1. $\forall k,\ S_{1:2(k)} \leftarrow \emptyset$
2. $nrótulo \leftarrow 0$
3. Para cada $p \in S_{1:2}$, onde $S_{1:2}$ é percorrido da esquerda para direita e de cima para baixo, faça:
   (a) Se $s_{1:2}((i,j)) = 1$
      i. então $S_{1:2(0)} \leftarrow S_{1:2(0)} \cup \{(i,j)\}$
      ii. senão Se $q \in S_{1:2(k)}$, tal que $k \neq 0$ e $q \in \{(i-1,j),(i,j-1),(i-1,j-1)\}$
         então $S_{1:2(k)} \leftarrow S_{1:2(k)} \cup \{(i,j)\}$
         senão $S_{1:2(nrótulo)} \leftarrow S_{1:2(nrótulo)} \cup \{(i,j)\}$; $nrótulo \leftarrow nrótulo+1$

Algoritmo 5.3: Cálculo das regiões sobre a imagem diferença.

e depois, durante toda a iteração, todas as chamadas ao operador *EC* condicionarão somente a solução selecionada. Já a escolha feita por passo determina que a solução a ser condicionada seja selecionada a cada chamada do operador *EC*.

As diferenças que o Condicionamento Ponderado provocaria no algoritmo 5.2 seriam na maneira como se valora a variável *best*, que passaria a ser por via de um sorteio ponderado pelos valores de aptidão $A(S_j)$. Se a escolha fosse feita por geração, então o sorteio de *best* seria feito entre os passos 2 e 2b, e o passo 2(b)iii seria suprimido do algoritmo. Se a escolha fosse feita por passo, então o sorteio seria realizado em lugar do passo 2(b)iii.

Condicionamento Sequencial

A diferença entre o condicionamento do AGdb e o condicionamento sequencial aqui proposto, é que este é aplicado em uma porção contínua da solução condicionada, enquanto aquele é aplicado em pontos aleatórios da solução.

Estratégia Elitista Total

Diferentemente da estratégia elitista, que conserva apenas a melhor solução de uma geração para outra, a *estratégia elitista total* consiste em escolher entre todas as soluções, geradoras e geradas ("filhos" e "pais"), as que tenham menores custos. Para tanto ordena-se os custos das soluções da geração corrente juntamente com os custos das novas soluções geradas. As *tamanho-da-população* primeiras soluções ordenadas formarão a

próxima geração. Assim garante-se que nenhuma boa solução da geração corrente será desperdiçada em favor de uma solução nova duvidosa. Ou seja, evita-se que soluções arbitrariamente ruins, geradas por *crossovers* de pouco *improvement*, substituam soluções antigas melhor valoradas. Também se impede que soluções de uma geração sobrevivam em detrimento de outras novas, que embora melhores que as anteriores, tiveram que lhes ceder o lugar em favor de uma estratégia elitista equivocada.

Capítulo 6

Método proposto

A SAt e o operador de condicionamento (busca local) do AGdb exploram suas respectivas vizinhanças de forma não determinística, ou seja, os métodos escolhem aleatoriamente uma solução S', tal que $S' \in \mathcal{N}(S)$, onde S é a solução corrente, S' a solução que será investigada pela busca e $\mathcal{N}(S)$ a vizinhança adotada. Portanto, nesses casos, não se garante que a melhor solução pertencente a $\mathcal{N}(S)$ será visitada. Mesmo assim, pode parecer razoável a escolha de se avaliar por vez apenas um em lugar de vários vizinhos de S, pois o problema aqui tratado lida com imagens, que geralmente são objetos de grandes dimensões, e consequentemente precisam ser abordados por métodos que sejam rápidos e menos onerosos.

Apesar disso, optou-se por realizar algumas experiências na exploração de novas vizinhanças de forma **determinística**. Dada uma solução S para o problema de deteção de bordas, e um ponto $p \in S$, tentou-se investigar todas (ou quase todas) as configurações possíveis para o ponto p e seus (m^2-1)-vizinhos, ou seja, para a janela $S(p, m \times m)$. A melhor entre todas as configurações investigadas seria candidata a substituir a região indicada por $S(p, m \times m)$. O objetivo é comparar S com todas as soluções que diferem de S somente pela região $S(p, m \times m)$. O conjunto destas soluções corresponde a uma vizinhança de S com um número finito de elementos, a qual será denominada por $\mathcal{N}_{m \times m}(S, p)$. Neste capítulo são relatadas as razões para a escolha de quais seriam as configurações "candidatas" para a região $S(p, m \times m)$, configurações estas que indicam em quê as soluções que compõem a vizinhança $\mathcal{N}_{m \times m}(S, p)$ diferem de S com relação à janela $S(p, m \times m)$. Além disso, o método de busca escolhido para explorar as novas vizinhanças será descrito com detalhes. Trata-se da aplicação de uma *busca em vizinhanças variáveis*-VNS (vide seção 3.5), que trabalha sobre as novas vizinhanças $\mathcal{N}_{m \times m}$.

6.1 Novas estratégias de transformação

Da vizinhança $\mathcal{N}_{\mathrm{SAt}}(S,p)$, definida pelas estratégias de transformação $\Im_i$ da SAt, escolhe-se uma única solução $S' \in \mathcal{N}_{\mathrm{SAt}}(S,p)$ a cada passo do método. Então S' é avaliada e comparada com a solução corrente, ou seja, a busca procura não avaliar mais do que uma configuração de pontos de borda S' por passo. O objetivo parece ser não tornar o algoritmo lento em relação ao número de soluções vizinhas de S que são investigadas, pois a cada vez que uma solução candidata $S' \in \mathcal{N}_{\mathrm{SA}}$ é gerada a partir de S, tem-se que avaliar os custos tanto de S como de S', em relação à região em que diferem: a janela centrada no ponto referencial p. A diferença entre os custos são então necessárias para decidir se a solução transformada substituirá ou não a solução corrente.

Teve-se o interesse de verificar qual o comportamento de uma busca local que avaliasse várias transformações de S a cada vez que o algoritmo é aplicado sobre um ponto p. Para tanto, várias modificações seriam realizadas em $S(p, m \times m)$, e consequentemente, várias soluções S'_i, distintas entre si seriam obtidas a partir de S, cada uma das quais diferindo de S somente na janela $S(p, m \times m)$. A união dessas soluções candidatas corresponde a vizinhança denotada por $\mathcal{N}_{m\times m}(S,p)$, onde p é o ponto referencial para as transformações.

As novas estratégias de transformação foram definidas de maneira que todas (ou quase todas) as configurações de pontos de borda resultantes para a região $S'_i(p, m \times m)$ fossem "finas", "não curvadas" e "não espessas". Tais configurações foram escolhidas com base nas janelas 1×1, 2×2, 3×3 e 4×4, ou seja, para $m \in \{1,2,3,4\}$. A intenção é realizar uma busca exaustiva pela melhor configuração de contornos, pesquisando entre grande parte das combinações possíveis para uma dada janela $m \times m$. Percebe-se que as transformações propostas não dependem dos valores contidos em $S(p, m\times m)$ para gerarem os valores das configurações de $S'_i(p, m \times m)$, pois estas apresentam sempre os mesmos valores independentemente de S e p, correspondendo a um conjunto de configurações para uma janela $m \times m$ especialmente construídas de maneira a formarem estruturas de borda ideais. As novas transformações serão denominadas por $\Im(m \times m)$ e seu esquema do funcionamento pode ser observado na figura 6.1.

Seja S a solução corrente e p o ponto de S onde se encontra a busca. As estratégias $\Im(m \times m)$ são descritas a seguir:

- $\Im(1 \times 1)$: gera soluções com as configurações possíveis para a janela 1×1 de S centrada em p, que são em um total de 2.
- $\Im(2 \times 2, k)$: gera as soluções S'_i, tais que as regiões $S'_i(p, m \times m)$ apresentem todas as configurações para uma janela 2×2 com k pontos de borda. A estratégia $\Im(2 \times 2)$ ficou definida como a que corresponde a união das $\Im(2 \times 2, k)$, para $k \in \{0, 2\}$,

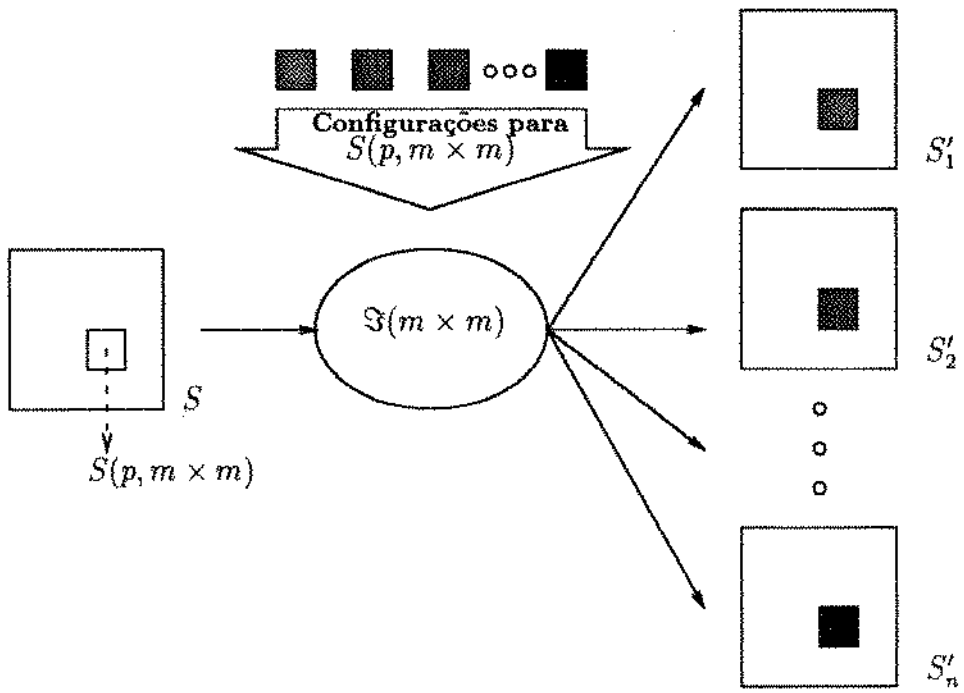


Figura 6.1: Funcionamento das transformações $\Im(m \times m)$.

pois as configurações correspondentes aos valores de k iguais a 1,3 e 4 representam estruturas de bordas com características não desejáveis, como por exemplo espessura excessiva, o que é reprovado pela função de custo. O número de combinações avaliáveis para $\Im(2 \times 2)$ correspondem à soma do total de transformações para as duas estratégias: $1 + 6 = 7$.

- $\Im(3 \times 3, k)$: gera as soluções S'_i com os conteúdos de $S'_i(p, m \times m)$ apresentando todas (ou quase todas) as configurações em uma janela 3×3 com k pontos de borda, onde $k \in \{0, 1, 2, 3\}$. A união destas estratégias ficou definida como a estratégia $\Im(3 \times 3)$. O total de configurações de $\Im(3 \times 3)$ é dado pela soma do número de combinações para valores de $k \in \{0, 1, 2, 3\}$: $\sum_{k=0}^{3} |\Im(3 \times 3, k)| = 1 + 8 + 20 + 16 = 45$. Outra estratégia que apresentou bons resultados é a que foi denominada por $\Im(3 \times 3, (03))$. Esta corresponde a um subconjunto de $\Im(3 \times 3)$ que contém apenas as transformações com $k \in \{0, 3\}$. O número de configurações de $\Im(3 \times 3, (03))$ é igual a $1_0 + 16_3 = 17$.

- $\Im(4 \times 4, k)$: gera as soluções S'_i, com $S'_i(p, m \times m)$ apresentando todas (ou quase todas) as configurações em uma janela 4×4 com k pontos de borda, onde $k \in \{0, 1, 2, 3, 4\}$. Novamente não foram consideradas as configurações para $k > 4$, pelos mesmos motivos apresentados para a estratégia $\Im(2 \times 2)$. O total de combinações, para $0 \leq k \leq 4$, é um número muito grande de transformações que tornaria inviável

um algoritmo de busca que as utilizasse. Apenas algumas das transformações da estratégia $\Im(4\times4,4)$, somada à estratégia $\Im(4\times4,0)$, num total de 40 transformações, foram selecionadas para compor a estratégia $\Im(4\times4)$.

A figura 6.2 mostra um esboço das configurações de pontos de borda correspondentes às transformações $\Im(m\times m,k)$.

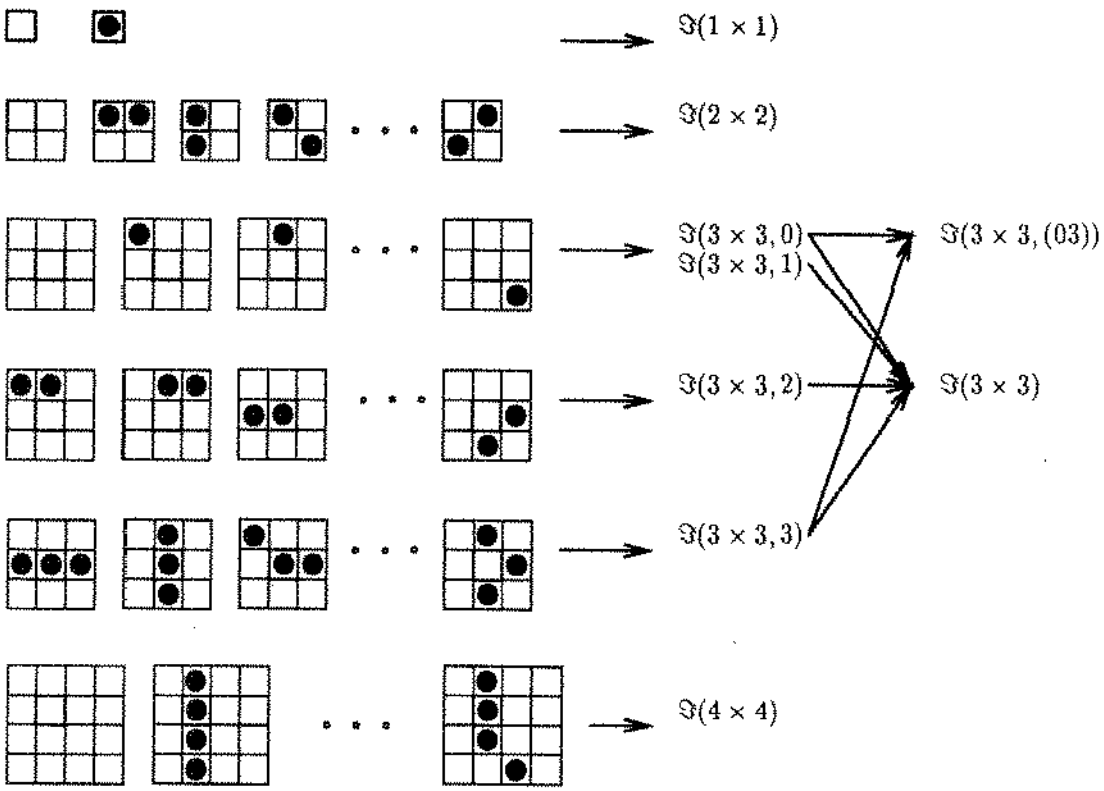


Figura 6.2: Vizinhanças exaustivas com estruturas contendo contornos finos, não curvados, (não ou semi)-fragmentados.

6.1.1 Um algoritmo de Busca Local

O desempenho de uma busca local que explora deterministicamente uma vizinhança $\mathcal{N}(S)$ pode ser prejudicado pelo "tamanho" de $\mathcal{N}(S)$. Vizinhanças muito grandes podem tornar a busca lenta, pois tais vizinhanças podem conter um vasto número de soluções que de alguma forma serão avaliadas a fim de se procurar por uma solução melhor que S. Por outro lado, grandes vizinhanças podem alcançar melhores soluções que estão bem "distantes" de S. Tomemos como exemplo uma vizinhança $\mathcal{N}(S)$, que contenha todas as soluções do espaço definido pelo problema, ou seja, todas as combinações existentes menos

1. Construa uma solução inicial (randômica ou induzida pela matriz de gradiente D) para a imagem I e armazene em S
2. $i \leftarrow 0$
3. Escolher *Vizinhança* $\in \{(1\times1),(2\times2),(3\times3),(3\times3,(03)),(4\times4)\}$, e escolher *offset* ≥ 0.0
4. Para cada ponto $p \in S$, onde p é escolhido sequencialmente, faça:
   (a) Aplique todas as transformações de $\Im_j$(*Vizinhança*)
   (b) Calcule o custo incremental $\Delta F(S, S'_j)$, para cada j, onde $j <$ (*Total de configurações da* Vizinhança).
   (c) Escolha a transformação de menor custo incremental S'_{best}. Se $\Delta F(S, S'_{best}) <$ *offset*, então efetive a transformação: $S \leftarrow S'_{best}$.
5. Se o critério de parada não foi atingido então
   (a) $i \leftarrow i + 1$
   (b) Volte para o passo 4

Algoritmo 6.1: Busca Local BLdb.

S, as quais satisfazem as restrições do problema. A melhor delas será obviamente o ótimo global para a instância investigada do problema. Tal algoritmo é conhecido como uma *enumeração explícita* do espaço de soluções, o que, para a grande maioria dos problemas combinatórios, é um procedimento inviável. A questão de se desenvolver um algoritmo de busca local eficiente, consiste em encontrar estratégias inteligentes de transformação $\Im$, que tomem a solução corrente S e a partir dela gerem soluções candidatas pertencentes a $\mathcal{N}(S)$, de maneira que as soluções geradas sejam promissoras e capazes de superar S em qualidade. As estratégias de transformação que definem $\mathcal{N}(S)$ devem restringir o processo de geração de soluções candidatas, a fim de que o algoritmo não percorra todas as soluções possíveis de maneira exaustiva. Por outro lado, a vizinhança $\mathcal{N}(S)$ não deve ser tão restrita, a fim de que o algoritmo não seja rapidamente retido em um pobre ótimo local.

A partir dessas idéias é que surgiram as vizinhanças $\mathcal{N}_{m\times m}(S,p)$, definidas na seção anterior, as quais foram primeiramente testadas através de uma busca local que utiliza a função de custos apresentada no capítulo 4, cujo o esboço encontra-se no algoritmo 6.1.

A variável *offset*, do algoritmo 6.1, determina o limiar de degradação permitido para o custo da solução corrente, ou seja, determina em "quanto" o custo de S pode degradar

a cada passo. Em geral, atribui-se zero ao *offset* nas primeiras iterações, o que significa que não serão aceitas quedas no custo da solução corrente. Depois que a busca "cai" em um ótimo local, ou seja, quando não transformação proposta sobre a solução corrente que reduza seu custo, então o *offset* recebe um valor maior que zero a fim de induzir degradações em S durante toda uma iteração, onde uma iteração corresponde a uma busca por todos os pontos pertencentes a S. Depois disso a busca continua com o *offset* $= 0$, na esperança de que a iteração com degradações tenha sido suficiente para superar o "vale" representado pelo ótimo local atingido. Assim, talvez a busca poderá percorrer novas soluções e não recair em ótimos locais já visitados. É importante ressaltar que o uso de um $offset > 0$ pode não impedir que o algoritmo retorne a um ótimo local já visitado, no entanto representa uma tentativa de superar tais situações através de degradações na solução corrente.

No algoritmo 6.1, apenas uma vizinhança $\mathcal{N}_{m\times m}$ é utilizada em todo processo, ou seja, o valor de m é fixo para os conjuntos $\mathcal{N}_{m\times m}$, sendo escolhido no início do algoritmo, no passo 3. As melhores vizinhanças encontradas para a busca local aqui apresentada foram a $\mathcal{N}_{3\times3}(S,p)$, e a $\mathcal{N}_{3\times3(03)}(S,p)$, talvez por que suas respectivas transformações $\Im(3\times3)$ e $\Im(3\times3,(03))$ gerem *todas* configurações 3×3 que respeitam as características resguardadas pela função de custo, enquanto $\Im(4\times4)$ não atende da mesma forma, devido ao grande número de configurações possíveis para uma janela 4×4. Para facilitar a referência, o algoritmo 6.1 que usa a vizinhança $\mathcal{N}_{3\times3}(S,p)$ será denominado por BL-45, enquanto o que usa $\mathcal{N}_{3\times3(03)}(S,p)$ será indicado como BL-17. O algoritmo 6.1 será denominado de BLdb.

Uma observação importante é o que é considerado ótimo local para a abordagem proposta de BLdb. Nesta nova proposta, um ótimo local S^* não se estabelece somente pelo fato de que não existe solução $S' \in \mathcal{N}(S^*)$ tal que $F(S') < F(S^*)$, o que é uma consideração comum na maioria dos métodos para busca (veja capítulo 3). Um ótimo local para a BLdb é uma solução S^* tal que não existe solução $S' \in \mathcal{N}_{m\times m}(S^*,p)$, qualquer que seja $p \in S^*$. Assim, somente após uma iteração inteira onde não ocorra alteração causada pela busca na solução corrente, é que então se admite a caracterização de um ótimo local.

6.2 Uma VNS para deteção de contornos

O esquema original de uma VNS (vide cap. 3) sugere que a cada vez que se encontre melhoria para a solução corrente, volte-se a utilizar vizinhança de índice 1 dentro de uma sequência r de vizinhanças especialmente escolhidas para serem utilizadas pelo método. No caso das vizinhanças $\mathcal{N}_{m\times m}$, uma sequência razoável é $r = (\mathcal{N}_{1\times1}, \mathcal{N}_{2\times2}, \mathcal{N}_{3\times3}, \mathcal{N}_{4\times4})$, que apresenta uma ordem natural e crescente no tamanho de suas vizinhanças. Logo, pelas

diretrizes do método original, uma VNS deve voltar a utilizar a vizinhança $\mathcal{N}_{1\times1}$ sempre que depois de uma iteração encontrou-se melhoria na solução corrente S_i em relação à solução da iteração anterior S_{i-1}. Ou seja, se pelo menos um ponto $p \in S_{i-1}$ foi tal que através das transformações ocorridas em p durante a iteração i, obteve-se $\Delta F(S) < 0$ ($F(S_i) - F(S_{i-1}) < 0$), então volta-se a utilizar $\mathcal{N}_{1\times1}$ na iteração $i+1$. Caso contrário, deve-se passar a usar a vizinhança imediatamente subsequente em r, que seria $\mathcal{N}_{k+1\times k+1}$. A abordagem aqui proposta de uma VNS para deteção de contornos, que será denominada por VNSdb, é um pouco diferente do esquema proposto por *Mladenovic* e *Hansen* [19]. Em vez de forçar a utilização de $\mathcal{N}_{1\times1}$ depois de cada iteração em que se verificou melhoria de custos, prossegue-se a busca usando a vizinhança corrente $\mathcal{N}_{k\times k}$, até que se atinja a vizinhança de ordem $k = 4$, quando então não se dispõe de vizinhanças subsequentes em r que sejam mais "densas" que a corrente. Só aí restaura-se o uso de $\mathcal{N}_{1\times1}$.

Ao mudar de vizinhança, nas primeiras iterações $i, i+1, \ldots i+$(*no. de iterações com* offset) em que se usa $\mathcal{N}_{k\times k}$, logo após ter-se usado $\mathcal{N}_{k-1\times k-1}$ na iteração $(i-1)$, a VNS proposta realiza sua busca onde se aceita, para cada ponto invetigado na solução corrente S, degradações de custo menores ou iguais a um certo limiar *offset*. Estas iterações justificam-se pelo fato de que certos pontos de S, que apresentam custos mais próximos de zero do que os demais pontos, são os principais candidatos a terem novas configurações testadas pela busca. Portanto degradações nesses pontos contribuem para alterar suas estabilidades, e assim, nas próximas iterações esses pontos certamente serão objeto de busca do algoritmo. Percebe-se que nessas iterações especiais ocorrerão eventuais quedas no custo da solução corrente, o que não deve indicar para o algoritmo que seja momento de se passar a usar a vizinhança subsequente $\mathcal{N}_{k+1\times k+1}$. Portanto, nessas iterações onde se admitem degradações, o algoritmo não deve mudar de vizinhança, o que só deve ser considerado quando o limiar de degradação voltar a ser *offset* $= 0$. O melhor valor encontrado experimentalmente para o limiar *offset* foi 0.5.

Portanto, a VNSdb só retorna ao uso de $\mathcal{N}_{1\times1}$ depois que passa por $\mathcal{N}_{2\times2}, \mathcal{N}_{3\times3}$ e $\mathcal{N}_{4\times4}$, exatamente nesta ordem. Trata-se de um ciclo onde ao se atingir um ótimo local de $\mathcal{N}_{4\times4}$ retorna-se ao uso de $\mathcal{N}_{1\times1}$. Percebe-se que o retorno a $\mathcal{N}_{1\times1}$ não se dá mais por que se encontrou melhoria, mas sim por que não se consegue encontrar solução melhor que a corrente usando a vizinhança mais densa $\mathcal{N}_{4\times4}$. O algoritmo 6.2 traz uma descrição detalhada do funcionamento da nova abordagem.

A versão do algoritmo 6.2 que utiliza a sequência de vizinhanças $(\mathcal{N}_{1\times1}, \mathcal{N}_{2\times2}, \mathcal{N}_{3\times3}, \mathcal{N}_{4\times4})$ será denominada de VNS-17, devido a vizinhança $\mathcal{N}_{3\times3}$ indica 17 soluções. Já a versão que usa a sequência $(\mathcal{N}_{1\times1}, \mathcal{N}_{2\times2}, \mathcal{N}_{3\times3,(03)}, \mathcal{N}_{4\times4})$ será denotada por VNS-45, pois $|\mathcal{N}_{3\times3,(03)}| = 45$.

1. Construa uma solução inicial para a imagem I que seja randômica ou induzida pela matriz de gradiente D, e armazene em S
2. Escolher sequência de vizinhanças: $\mathcal{SN} \leftarrow \{(1 \times 1), (2 \times 2), (3 \times 3), (4 \times 4)\}$ ou $\mathcal{SN} \leftarrow \{(1 \times 1), (2 \times 2), (3 \times 3, (03)), (4 \times 4)\}$
3. Escolher mínima-vizinhança $\leq |\mathcal{SN}|$, $k \leftarrow$ mínima-vizinhança
4. Escolher *offset* ≥ 0.0
5. $i \leftarrow 0$, $d \leftarrow$ número de iterações com "offset"
6. Para cada ponto $p \in S$, onde p é escolhido sequencialmente, faça:
   (a) Aplique todas as transformações de $\Im_j(k)$
   (b) Calcule o custo incremental $\Delta F(S, S'_j)$, para cada j, onde $j <$ *(Total de configurações da k).*
   (c) Escolha a transformação de menor custo incremental S'_{best}. Se $\Delta F(S, S'_{best}) <$ *offset*, então efetive a transformação: $S \leftarrow S'_{best}$.
7. $S_i \leftarrow S$
8. Se ótimo local foi atingido, ou seja, se $F(S_i) = F(S_{i-1})$ então
   (a) $k \leftarrow k + 1$
   (b) Se $k > |\mathcal{SN}|$ então $k \leftarrow$ mínima-vizinhança
   (c) $d \leftarrow$ número de iterações com "offset"
9. $d \leftarrow d - 1$
10. Se $d \leq 0$ então *offset* $\leftarrow 0$
11. Se critério de parada não foi atingido então
    (a) $i \leftarrow i + 1$
    (b) Volte para o passo 6

Algoritmo 6.2: Busca em vizinhanças variáveis para deteção de bordas - VNSdb.

6.3 Conceitos avançados de Metaheurísticas

Tanto na VNSdb como na BLdb, abordagens apresentadas neste capítulo, investigou-se qual a melhor maneira de percorrer as soluções durante a busca: se de maneira sequencial ou randômica. Testes computacionais foram realizados com as implementações para varredura randômica e sequencial, onde a forma sequencial dá-se percorrendo os pontos da matriz da solução orientando-se da esquerda para direita e de cima para baixo. Em todas imagens submetidas ao VNSdb ou ao BLdb, foram obtidas soluções finais com varredura sequencial sempre melhores que as soluções obtidas através de varredura randômica.

Outro ponto importante é como se deve compor a solução inicial S_0. Testes foram realizados com soluções iniciais randômicas e com soluções S_0 para as quais os valores de $s_0(p)$, para $p \in S_0$, são obtidos por um sorteio ponderado pelos valores $D(p)$, onde D é a matriz de gradiente (vide cap. 4). Esta última opção de inicialização mostrou-se superior à primeira, conferindo uma aceleração na convergência do algoritmo.

Subotimalidades

Dado que a VNSdb explora uma vizinhança de maneira determinística, pode-se portanto chegar a um ótimo local, onde garante-se que não existe solução $S' \in \mathcal{N}_{m\times m}(S,p)$, qualquer que seja p, tal que $F(S') < F(S)$, onde F é a função de custos adotada. Acontece que, durante a busca nem todos os pontos da solução corrente chegam a esse estado de otimalidade ao mesmo tempo. Alguns pontos de S são tais que em poucas iterações não encontram melhores configurações em $\mathcal{N}_{m\times m}(S,p)$, enquanto outros pontos ainda são passíveis de busca em relação a $\mathcal{N}_{m\times m}(S,p)$. A questão é identificar quais pontos já se encontram "estáveis", para então não investigá-los em iterações subsequentes. Para tanto, dado que o algoritmo está investigando a vizinhança $\mathcal{N}_{m\times m}(S,p)$, a cada iteração i (onde uma iteração corresponde a uma varredura total da solução S) guarda-se o conjunto C_i de pontos que foram modificados pela busca. Esta informação será utilizada na próxima iteração $(i+1)$, onde somente serão investigados os pontos de S que pertençam a C_i e alguns de seus pontos vizinhos. Tais pontos também devem ser investigados pela busca a fim de verificar se a modificação da configuração em um ponto p não interferiu no estado dos pontos próximos a ele. Esse tipo de interdependência é determinado pela função de custo.

Em uma solução S, o custo em um ponto p é influenciado por outros pontos vizinhos. Isto se dá por que a função de custo considera fatores como curvatura c_c e espessura c_t (vice cap. 4), os quais exigem uma investigação de no máximo a janela $S(p, 5\times 5)$. As transformações habituais da busca em p, como as $\Im_j$ da SAt e as $\Im(m\times m)$ da

VNS proposta, se dão em $S(p, 3 \times 3)$. Portanto, as alterações que se dão em $S(p, 3 \times 3)$ influenciam o estado de todos os pontos contidos em $S(p, 7 \times 7)$, e consequentemente altera o custo desses pontos. Se um ponto q é alterado em uma dada iteração i, então todos os pontos da janela $S(p, 7 \times 7)$ serão candidatos à busca na iteração $i + 1$. Outros pontos que se apresentem fora desse escopo, não serão influenciados pela alteração. E se ainda esses pontos fora de $S(p, 7 \times 7)$ estiverem em um estado de otimalidade local, então na próxima iteração também permanecerão em otimalidade se não houver outra alteração na solução que interfira no seu estado.

Usando esses argumentos de "subotimalidade", resolveu-se guardar as localidades das alterações de uma iteração para a próxima, a fim de se utilizar essa informação e restringir o processo de busca aos pontos que possam ter seu estado influenciado pelas alterações da iteração anterior. O uso dessas restrições mostrou uma queda de 50% nos custos computacionais, enquanto os resultados não apresentaram distinção daqueles obtidos sem o uso dessas restrições de "subotimalidade".

A importância da troca de vizinhanças

A troca de vizinhanças é um ponto fundamental para o sucesso de uma VNS. Foram feitas experiências neste trabalho, utilizando uma vizinhança que corresponde a união das vizinhanção $\mathcal{N}_{m \times m}$: $\mathcal{N}_u = \mathcal{N}_{1\times1} \cup \mathcal{N}_{2\times2} \cup \mathcal{N}_{3\times3} \cup \mathcal{N}_{4\times4}$. A $\mathcal{N}_u$ foi utilizada em uma busca local e os resultados foram inferiores a VNS-17 e VNS-45. É provável que, com a utilização de uma vizinhança como a $\mathcal{N}_u$ e sem a utilização de uma outra vizinhança diferente e não contida em $\mathcal{N}_u$, o algoritmo não tenha a capacidade de "escapar" de um ótimo local, assim como pode fazer uma VNS, através da mudança da vizinhança utilizada até então.

Capítulo 7

Experiência computacional

Quando se propõe uma nova abordagem heurística para um problema, geralmente testes computacionais comparativos são realizados em relação à melhor abordagem apresentada na literatura atual. Para tanto, foram implementadas e testadas as novas abordagens para deteção de contornos: a BLdb e a VNSdb do capítulo 6, além das abordagens heurísticas principais da literatura: a SAt e o AGdb (vide cap. 5). Todos estes algoritmos são baseados em métodos de otimização diferentes entre si (vide cap. 3), no entanto eles abordam deteção de contornos como um problema de minimização de uma função de custos. Tal função, que avalia a qualidade das soluções, bem como a definição de bordas em que é baseada, são apresentadas no capítulo 4.

Para avaliar a performance das implementações foram medidos os tempos de CPU e os relativos custos das melhores soluções, pois o custo das soluções por iteração não representava uma boa medida, dado que cada iteração tinha um número de operações distintas para cada método. Tome-se como exemplo a VNSdb, onde o número de operações por ponto da imagem pode crescer muito de uma iteração para outra, devido à mudança de uma vizinhança $\mathcal{N}_{m\times m}$ para uma $\mathcal{N}_{m+1\times m+1}$. Nesta última há muito mais vizinhos a serem avaliados por vez do que na outra. Dados esses incovenientes, optou-se por medir a evolução do custo da melhor solução encontrada com o decorrer do tempo de CPU.

Todos os resultados apresentados neste texto foram obtidos pela execução das implementações numa estação de trabalho SPARC 1000 com sistema SunOS 5.5.

7.1 Experimentos Realizados com a SAt

Para ressaltar a qualidade da VNSdb, era necessário compará-la com a implementação das melhores versões possíveis da SAt e do AGdb. Experimentos foram feitos avaliando o

comportamento da convergência da SAt em relação à função de temperatura utilizada, e também em relação à composição da solução inicial S_0. Como já foi comentado na seção 5.1, a melhor função de temperatura encontrada foi a do tipo geométrica: $T_i = T_0(1-\alpha)^i$, onde i é a iteração corrente do algoritmo, T_0 é a temperatura no instante 0 (zero) e $0.01 \leq \alpha \leq 0.02$. Os melhores valores para T_0 ficaram em torno de 35×10^{-2}. A inspiração para os testes que levaram à obtenção empírica dessas funções geométricas foi a própria função logarítmica proposta pela a abordagem original da SAt [28]: $T_i = 1/log(i+c)$, onde i é a iteração corrente e c uma constante para acelerar a convergência a zero, sendo melhores para valores em torno de $c = 2$. A intenção seria obter uma função de temperatura que produzisse um resfriamento mais rápido que o da função logarítmica. Testes computacionais foram feitos com funções geométricas do tipo: $T_i = T_0(1-\alpha)^i$. Os valores testados para T_0 foram os valores de $T_i = 1/log(i+2)$ para as primeiras iterações, que correspondem a $T_i > 0.3$ quando $i \leq 25$. Os valores testados para α foram: 0.07, 0.06, 0.05, 0.04, 0.03, 0.02, 0.015, 0.01, e 0.005.

Já a solução inicial que melhores resultados produziu foi a composta randomicamente. Diferentemente da VNSdb, proposta no capítulo 6, a SAt parece ser influenciada negativamente por uma solução inicial que seja baseada no resultado da matriz de gradiente. Estes valores provavelmente induzem resultados que são difíceis de serem superados pelas estratégias de busca da heurística, ou seja, levam o algoritmo a prender-se em uma certa região do espaço de soluções de onde é "difícil" se sair, pois representa um "vale" em relação à função de custo. Uma vez dentro do "vale", as estratégias de busca para superá-lo podem não ser suficientes para transpô-lo, e a busca então terá que se restringir a explorar as soluções que se encontram nessa região restrita, enquanto possam existir soluções de qualidade superior fora da região. A figura 7.1 apresenta um exemplo simples, em duas dimensões, desse típico comportamento de buscas descendentes.

SAt de Tan *versus* **Busca Local** As buscas locais BL-17 e BL-45, que correspondem às duas versões do algoritmo BLdb do capítulo 6, foram implementadas a fim de serem comparadas com os resultados da implementação da SAt. Mais de trinta instâncias-imagens foram submetidas às implementações e os tempos de CPU foram medidos. Uma amostragem dos resultados está disposta na tabela 7.1. Além da SAt para função de temperatura $T_i = 0.35 \times 0.99^i$, também executou-se a SAt para $T_i = 0$, a fim de mostrar a importância da "cristalização" gradual da temperatura da SAt, e das consequentes degradações provocadas pelas temperaturas diferentes de zero. Como pode ser visto na tabela 7.1, há uma razoável superioridade da SAt, com $T_i = 0.35 \times 0.99^i$, em relação à versão da SAt restrita, com $T_i = 0$. Isto pode também ser percebido no exemplo da figura 7.2, onde se pode ver o gráfico da evolução dos custos da melhor solução encontrada, em relação ao tempo de CPU, para imagem B.3. Já as buscas locais BL-17 e BL-45 concorrem

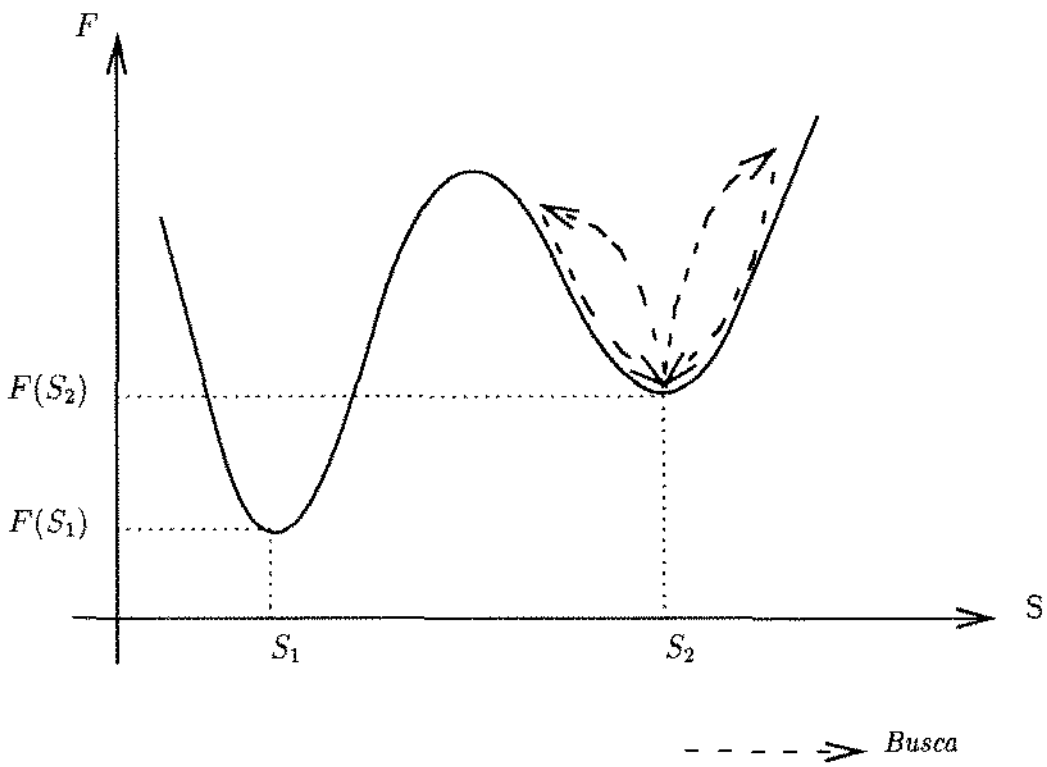


Figura 7.1: Exemplo de busca que se aprisiona em um "vale" do espaço de soluções $\mathcal{S}$ pela função de avaliação F.

de igual para igual com a SAt, embora as duas nem "sempre" superem as soluções finais da SAt. As imagens finais para a imagem-instância B.1 da tabela 7.1 podem ser vistas na figura 7.3 ou na seção B.1 do apêndice B. Este exemplo corresponde a uma imagem artificial acrescida de um ruído gaussiano de variância $\sigma^2 = 75$. Pode-se notar que os resultados das BL-45 e BL-17 foram menos influenciados pelo ruído que os resultados da SAt.

Os índices da tabela 7.1 correspondem às seções do apêndice B onde estão dispostas as respectivas imagens e suas soluções finais para SAt, BL-45 e BL-17.

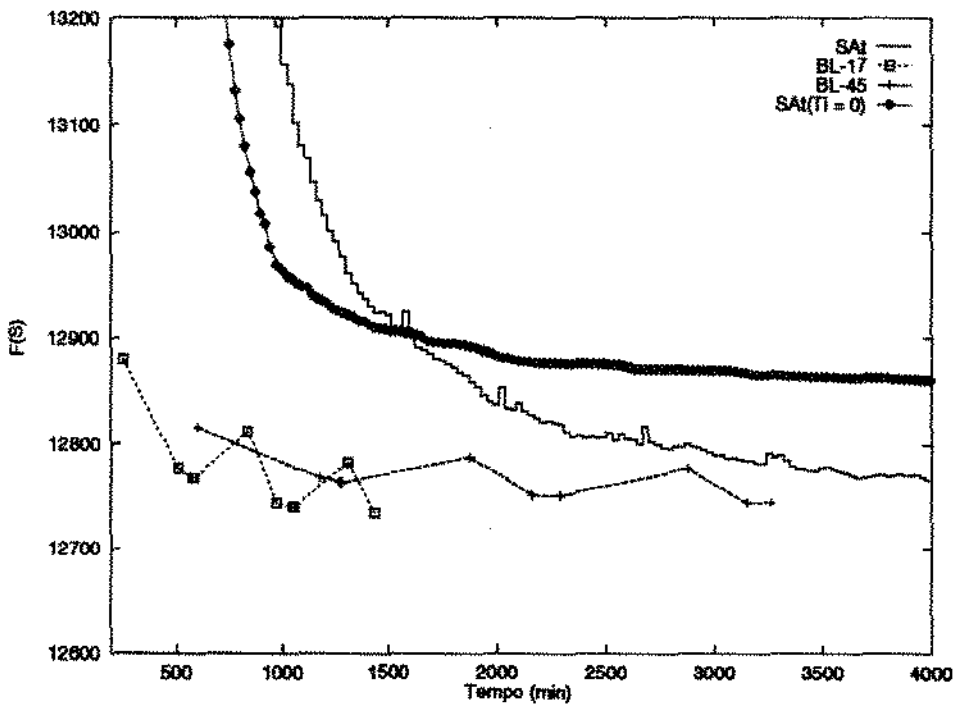


Figura 7.2: Gráfico *Custo*×*Tempo* para as execuções da SAt com $T_i = 0.35 \times 0.99^i$, SAt com $T_i = 0$, BL-17 e BL-45

7.2 Experiências com o AGdb

O operador de condicionamento ponderado, proposto na seção 5.2.1 para AGdb, apresentou melhores resultados que o de condicionamento randômico do AGdb original. Em especial, o sorteio da solução a ser condicionada feita no início de cada geração (*iteração*) infere melhores resultados que a escolha feita a cada passo do condicionamento. Ou seja, quando se escolhe uma solução para condicionamento, é preferível escolhê-la no início de uma geração, e deixar que a solução escolhida seja a mesma durante todas as chamadas ao

Imagens	Tempo de CPU (min)	10	20	40	80	120
B.1	SAt ($T_i = 0$)	3753.56	3749.12	3746.94		
(100x100)	SAt ($T_i = 0.35 \times 0.99^i$)	3718.81	3712.40	3712.40		
	BL-45	3701.32				
	BL-17	**3693.34**				
B.3	SAt ($T_i = 0$)	13304.98	12926.37	12875.72	12855.14	12840.82
(233x175)	SAt ($T_i = 0.35 \times 0.99^i$)	13836.07	12976.91	12805.72	12754.17	12742.67
	BL-45	12768.24	12764.09	12751.35	12744.18	
	BL-17	**12768.02**	**12740.12**	**12734.98**		
B.4	SAt ($T_i = 0$)	5086.79	5044.48	5041.78	5038.27	5038.27
(233x174)	**SAt** ($T_i = 0.35 \times 0.99^i$)	**5111.92**	**5034.84**	**5017.13**	**5011.96**	
	BL-45	5017.66	5017.06	5012.56	5012.46	
	BL-17	5018.59	5012.19			
B.5	SAt ($T_i = 0$)	11955.62	10350.19	10244.07	10215.41	10213.36
(320x200)	SAt ($T_i = 0.35 \times 0.99^i$)	12643.32	10560.42	10184.76	10106.42	10092.06
	BL-45	10171.82	10150.84	10130.50	10123.59	
	BL-17	**10130.31**	**10076.11**	**10073.58**		
B.6	SAt ($T_i = 0$)	11393.93	11156.32	11137.59	11131.32	11125.95
(185x216)	SAt ($T_i = 0.35 \times 0.99^i$)	11474.90	11131.47	11073.06	11057.58	11057.43
	BL-45	**11062.24**	**11061.06**	**11052.14**	**11052.14**	
	BL-17	11075.59	11058.93			
B.7	SAt ($T_i = 0$)	7901.31	7785.49	7771.15	7765.27	7761.58
(233x145)	SAt ($T_i = 0.35 \times 0.99^i$)	8250.59	7833.56	7714.29	7688.39	7686.84
	BL-45	7709.24	7709.05	7689.87	7689.87	
	BL-17	**7694.35**	**7685.35**			
B.8	SAt ($T_i = 0$)	10714.01	10519.34	10489.87	10477.35	10472.77
(259x162)	SAt ($T_i = 0.35 \times 0.99^i$)	11376.46	10579.09	10438.96	10394.00	10387.76
	BL-45	10402.29	10401.85	10382.13	10381.97	
	BL-17	**10396.97**	**10378.26**	**10376.34**		
B.9	SAt ($T_i = 0$)	8682.89	8575.50	8562.57	8543.05	8535.26
(225x142)	SAt ($T_i = 0.35 \times 0.99^i$)	8943.53	8595.45	8498.61	8477.02	8476.71
	BL-45	8497.16	8494.87	8482.85		
	BL-17	**8479.61**	**8470.06**			

Tabela 7.1: Resultados comparativos entre a SAt, a SAt restrita com temperatura constante $T_i = 0$, a BL-45 e a BL-17

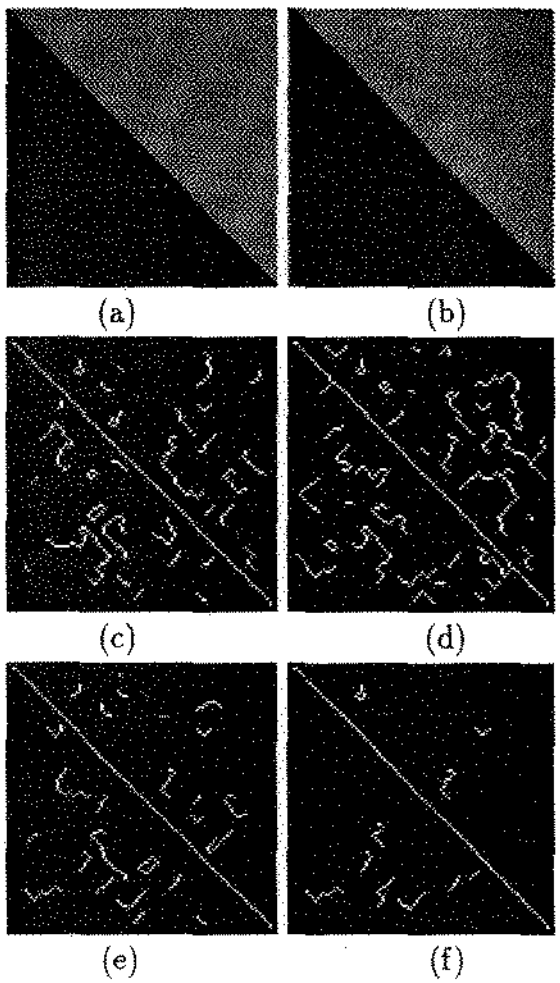


Figura 7.3: (a) Imagem original; (b) Imagem com ruído Gaussiano de variância $\sigma^2 = 75$; (c) Resultado da SAt; (d) Resultado da SAt com $T_i = 0$; (e) Resultado da BL-45; (f) Resultado da BL-17. Todos os resultados foram obtidos para matriz de gradiente com limiar $\xi = 10$.

procedimento de condicionamento de uma iteração. Assim, a mesma solução é submetida a esse operador de busca local durante uma iteração inteira. Os experimentos indicaram que essa opção é melhor do que se condicionar sempre a melhor solução da população corrente, ato este que infere uma supremacia de uma dada solução, que logo no início do algoritmo se torna superior às demais, devido à aplicação do operador de condicionamento, e que consequentemente é escolhida para condicionamento nas iterações seguintes. Assim esta solução torna-se a melhor solução da população e dificilmente pode ser superada pelas soluções resultantes da aplicação de outros operadores. Esta supremacia de uma solução prejudica a manutenção da diversidade da população de um algoritmo genético. Isto se dá porque essa solução "super-valorada" tem maior probabilidade de ser escolhida para participar das operações de *crossover*, e com isso imprime suas características às novas soluções produzidas. Estes sintomas podem levar o algoritmo a uma uniformização da população, e material genético que poderia ser importante para a composição de soluções cada vez melhores pode ser simplesmente descartado por uma reprodução exagerada das características da melhor solução corrente. O condicionamento ponderado permite que boas soluções próximas em custo da melhor solução possam ser escolhidas por sorteio baseado nos seus custos, assim garantindo que soluções distintas sejam condicionadas e promovam uma maior diversidade das soluções.

Em todas instâncias testadas, o caráter estocástico do condicionamento ponderado obteve vantagens em relação ao procedimento determinístico de se escolher para condicionamento a melhor solução corrente, como ocorre com o operador de condicionamento original do AGdb.

Outra observação retirada das experiências realizadas é que a estratégia *elitista total*, proposta na seção 5.2.1, apresentou-se superior em relação a *elitista simples* da abordagem original, embora a elitista total apresente uma tendência de homogeneização da população de soluções depois de 80 iterações.

Seguem algumas considerações sobre os resultados obtidos pelas implementações baseadas no AGdb:

Crossover Inteligente *versus* **Crossover Simples** O *crossover* inteligente sobre regiões, o **cir**, proposto na seção 5.2.1, obteve os melhores resultados entre todos os demais tipos de *crossover* apresentados no capítulo 5. O **cir** sempre produz boas soluções no decorrer da execução do algoritmo. As soluções geradas são geralmente de custo inferior aos das soluções geradoras, mesmo em estágios avançados da execução, quando já a população de soluções apresenta-se estabilizada e uniforme em termos de custos.

O *crossover* inteligente sobre grades, o **cig**, apresentou resultados superiores ao *crossover* simples (**cs**), caracterizando-se como o segundo melhor tipo de *crossover* proposto.

As soluções geradas são de custo inferior aos das geradoras até por volta da geração 70. O *crossover* inteligente pontual, **cip**, mostrou resultados pobres no sentido de produzir soluções de custo inferior aos das soluções geradoras. Nas primeiras iterações, quando as soluções ainda não têm contornos bem definidos, as soluções geradas apresentam melhorias de custo em relação às geradoras. Após poucas iterações (por volta de 15), quando as soluções já apresentam uma relativa qualidade, as soluções "filhas" passam a apresentar degradações de custo significativas em relação aos custos dos "pais".

O *crossover* randômico apresentou resultados pobres, pois assim como o *crossover* inteligente pontual, o caráter aleatório da formação das novas soluções, parece provocar "quebras" nas linhas de contorno, e consequentes degradações nas soluções geradas em relação ao custo das geradoras. A tabela 7.2 apresenta os custos finais do AGdb para várias imagens submetidas aos vários tipos de *crossovers*. Os resultados foram computados em 200 gerações (iterações), usando estratégia elitista total, sem mutação, com condicionamento ponderado por geração, feito de forma sequencial. Além disso, o tamanho da população foi fixado em 50 soluções, e a razão de uma solução submetida ao condicionamento foi de 0.05. As soluções finais correspondentes à linha 9 da tabela 7.2 estão dispostas na figura 7.4.

Imagens	*Crossovers*			
	cs	**cip**	**cig**	**cir**
1	13740.152952	13803.023018	13752.198745	**13693.373311**
2	7734.406620	7844.470553	7741.128050	**7727.387871**
3	12825.930781	12862.820472	12822.174881	**12793.442207**
4	9355.763598	9430.085725	9349.863968	**9309.503989**
5	5038.978889	5058.797450	5039.905852	**5024.920857**
6	8253.368303	8294.882720	8265.596562	**8240.927783**
7	7933.457446	8055.576335	**7919.934961**	7926.166598
8	8046.823884	8147.491788	8063.952857	**8027.493693**
9	12845.800776	12888.792146	12837.628311	**12808.537291**

Tabela 7.2: Custos finais obtidos pelo algoritmo genético para várias imagens utilizando-se de diversos tipos de *crossover*.

Condicionamento Sequencial *versus* Randômico O condicionamento randômico apresentou-se superior ao condicionamento sequencial em quase todos os resultados. A única combinação de operadores em que os resultados favoreceram o condicionamento sequencial é a execução do algoritmo genético com *crossover* inteligente sobre regiões e com

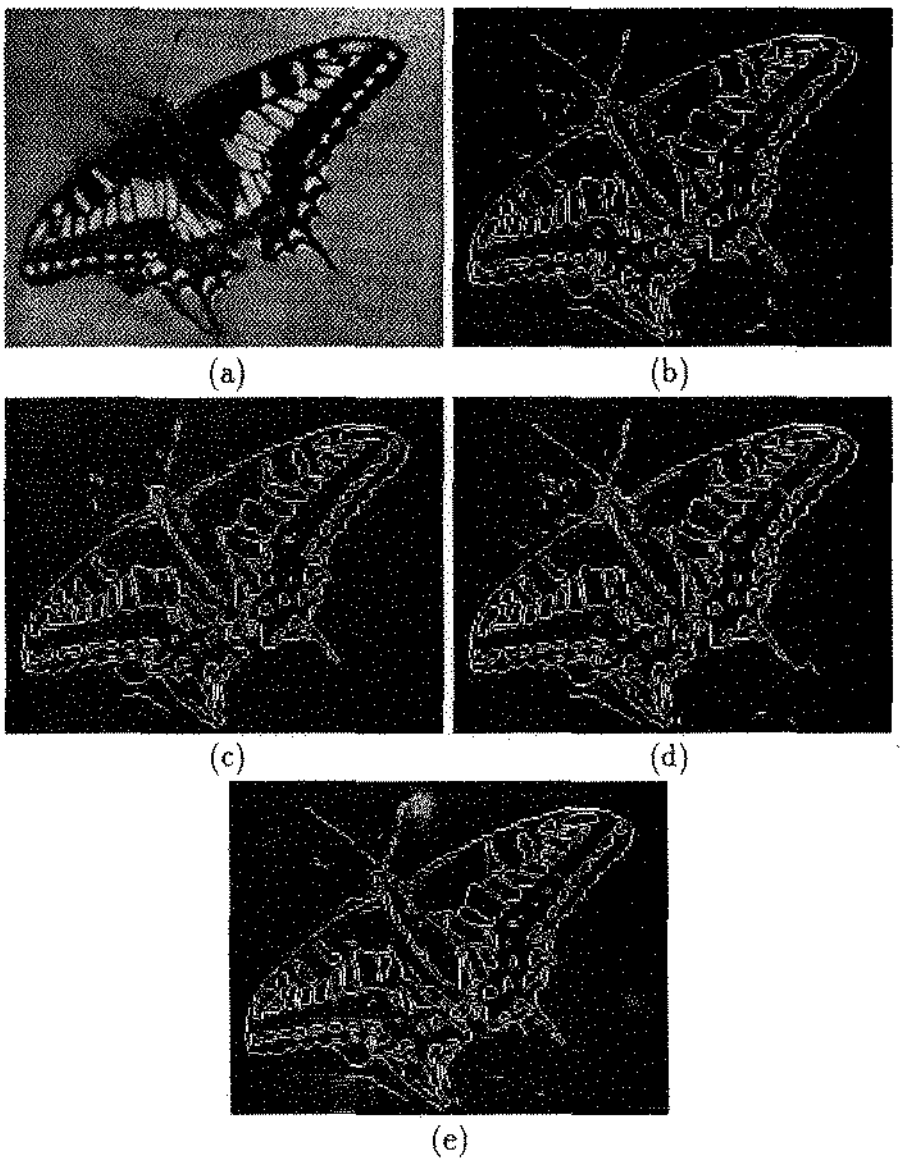


Figura 7.4: (a) Imagem original; (b) Resultado do AGdb com **cs**; (c) Solução final para AGdb com **cip**; (d) Resultado final com AGdb com **cig**; (e) Resultado para o **cir**. Estes resultados foram obtidos para limiar da matriz de gradiente $\xi = 20$.

condicionamento sequencial. Tal combinação mostrou ser a melhor entre todas as outras que associavam os tipos de operadores de condicionamento e de *crossover*, pois obteve os menores custos para todos os exemplos de imagens submetidos ao algoritmo.

Os custos no decorrer do algoritmo Observações devem ser feitas com relação ao comportamento da população de soluções no decorrer da execução do algoritmo genético.

O **cip** parece ter uma rápida convergência. Levando sua população rapidamente a um estado que caracterizará sua configuração final. Já o **cig** e o **cir** apresentam uma convergência mais suave, além de terem uma capacidade maior de obter soluções de menor custo.

Mesmo antes da iteração 100, há uma tendência à formação de uma população de soluções homogêneas. Apesar disso, os operadores de **cig** e o **cir** apresentam habilidade para obtenção de resultados melhores que os demais.

7.3 SAt versus AGdb

Apesar das modificações propostas para o AGdb (vide seção 5.2.1), cujos resultados computacionais obtidos comparativamente com o AGdb original estão dispostos na seção 7.2, as implementações de todas versões do AGdb foram superadas pela da SAt. Mesmo utilizando operadores como o **cir**, que imprimiu significativa melhoria ao AGdb, não houve incremento de sua convergência em relação à da SAt. Cerca de 20 instâncias (*imagens*) foram testadas e para todas soluções finais obtidas, as da SAt tiveram custos inferiores e levaram menos tempo para serem atingidas pelo método. Veja a tabela 7.3, onde estão dispostos os resultados computacionais comparativos entre a SAt e o AGdb para 4 das 20 instâncias testadas. A versão do AGdb utilizada é a mesma que a utilizada para obter os resultados da tabela 7.2, sendo que as linhas AGdb-50, AGdb-75 e AGdb-100 correspondem aos resultados do AGdb para populações com 50, 75 e 100 soluções respectivamente. Os resultados da SAt são para as funções de temperatura $T_i = 0$ e $T_i = 0.35 \times 099^i$.

7.4 Resultados para a VNSdb

As implementações da VNS-17 e da VNS-45 mostraram considerável superioridade em relação às implementações do AGdb, da SAt, da BL-17 e da BL-45, seja em termos da qualidade e dos custos das soluções finais, seja em termos do tempo computacional empreendido. Para não carregar as figuras e tabelas, as comparações serão restritas

Tempo de CPU (min)		10	20	40	80	120
1	AGdb-50	20649.16	16004.44	13356.05	12876.70	12823.46
	AGdb-75	23811.91	20639.71	15494.48	13076.38	12890.52
	AGdb-100	19744.76	15642.66	13100.62	12841.45	12813.95
	SAt ($T_i = 0$)	13304.99	12926.38	12875.73	12855.15	12840.82
	SAt ($T_i = 0.35 \times 0.99^i$)	**13836.07**	**12976.91**	**12805.73**	**12754.18**	**12742.68**
2	AGdb-50	12242.77	6278.07	5074.91	5035.29	5029.32
	AGdb-75	17166.71	13754.52	7099.83	5092.20	5048.51
	AGdb-100	12880.09	6926.05	5062.81	5027.39	5027.39
	SAt ($T_i = 0$)	5086.80	5044.48	5041.79	5038.27	5038.27
	SAt ($T_i = 0.35 \times 0.99^i$)	**5111.92**	**5034.84**	**5017.14**	**5011.96**	
3	AGdb-50	31654.90	22186.37	12877.51	10353.52	10185.35
	AGdb-75	25710.14	18976.21	11666.18	10235.93	10182.68
	AGdb-100	23361.82	16516.74	11420.11	10263.71	10198.71
	SAt ($T_i = 0$)	11955.63	10350.20	10244.08	10215.42	10213.37
	SAt ($T_i = 0.35 \times 0.99^i$)	**12643.32**	**10560.43**	**10184.77**	**10106.43**	**10092.07**
4	AGdb-50	20102.47	13456.15	11276.93	11084.09	11074.02
	AGdb-75	20034.61	16509.39	11766.58	11123.61	11105.41
	AGdb-100	17510.91	13148.30	11191.80	11088.52	11080.06
	SAt ($T_i = 0$)	11393.93	11156.32	11137.59	11131.32	11125.96
	SAt ($T_i = 0.35 \times 0.99^i$)	**11474.90**	**11131.47**	**11073.06**	**11057.59**	**11057.43**

Tabela 7.3: Resultados comparativos entre a SAt e o AGdb.

aos resultados obtidos para a SAt, a VNS-17 e a VNS-45, pois a implementação do AGdb apresentou-se sensivelmente inferior à da SAt. Veja a tabela 7.4 e as imagens das soluções finais na figura 7.7, que são referentes à instância A.4 da tabela 7.4. A instância A.1, que corresponde à uma imagem adicionada com um ruído Gaussiano de variância $\sigma^2 = 144$, apresenta suas soluções finais na figura 7.6. Na figura 7.7(b), mostra-se o resultado do cálculo do gradiente (vide seção 4.1), e na figura 7.4(c) mostra-se o resultado da aplicação de um limiar sobre o resultado de gradiente, que diferentemente do limiar ξ utilizado para mapear os valores da matriz de gradiente D para o intervalo $[0, 1]$, mapeia D para valores ou 0 ($< 1.8\xi$) ou 1 ($\geq 1.8\xi$) (veja seção 4.1). Percebe-se que a figura 7.7(c) apresenta linhas de borda espessas, pois não houve preocupação em rejeitar-se tal qualidade ao se aplicar o limiar. Já a figura 7.7(d), correspondente à aplicação da SAt com $T_i = 0.35 \times 0.99^i$, mostrando linhas não espessas, mas no entanto respondendo incorretamente em regiões que não caracterizam contornos. O melhor resultado parece ser o da figura 7.7(e), correspondente à saída final da VNS-45, onde não aparecem os mesmos enganos do resultado da SAt. No entanto, através de uma observação mais atenta, percebe-se que certas regiões muito estreitas, como as que caracterizam certas linhas estreitas no desenho de entrada, foram eliminadas pelo algoritmo, provavelmente devido a critérios como a *espessura* c_t.

Os índices das imagens-instâncias da tabela 7.4 indicam os índices das seções do apêndice A, onde estão dispostas as imagens da tabela mais exemplos adicionais, bem como suas soluções correspondentes para a SAt, VNS-17 e VNS-45. A imagem A.4, tem seu gráfico de custos para a SAt e a VNS plotados na figura 7.5.

Tempo de CPU (min)		10	20	40	80	120
A.1	SAt	3373.63	3372.72			
	VNS-17	**3361.34**				
	VNS-45	**3361.34**				
A.2	SAt	3732.98	3724.15	3724.15		
	VNS-45	3712.84				
	VNS-17	**3712.61**				
A.4	SAt	14547.02	10804.21	10262.84	10121.72	10092.66
	VNS-17	10097.35	10093.78	10067.97	10067.91	
	VNS-45	**10172.87**	**10143.58**	**10108.00**	**10065.84**	
A.9	SAt	17964.15	16609.65	16263.36	16131.74	16115.88
	VNS-45	16161.29	16118.91	16118.59	16102.32	
	VNS-17	**16198.40**	**16096.38**	**16096.38**	**16096.38**	
A.10	SAt	11416.34	10627.70	10561.92	10548.84	10548.42
	VNS-45	10562.35	10551.86	10550.44	10547.54	
	VNS-17	**10549.85**	**10549.85**	**10547.42**	**10547.42**	
A.11	SAt	20014.49	14211.07	12980.95	12763.38	12710.46
	VNS-45	12728.74	12679.77	12678.05	12667.85	
	VNS-17	**12765.03**	**12665.09**	**12665.09**	**12665.09**	
A.12	SAt	9227.22	8612.11	8489.69	8450.67	8445.95
	VNS-45	**8482.33**	**8482.33**	**8446.54**	**8443.22**	**8443.22**
	VNS-17	8450.71	8446.57	8445.34	8445.34	
A.13	SAt	12692.57	11984.00	11880.98	11855.14	11852.08
	VNS-45	11874.53	11853.48	11852.25	11844.67	11844.67
	VNS-17	11849.86	11848.50	11848.50	11847.65	

Tabela 7.4: Resultados comparativos entre a SAt, com $T_i = 0.35 \times 0.99^i$, e as implementações das duas versões da VNSdb: VNS-17 e VNS-45, que correspondem à VNSdb com vizinhanças $\mathcal{N}_{3\times3}(S,p)$ com 17 e 45 configurações, respectivamente.

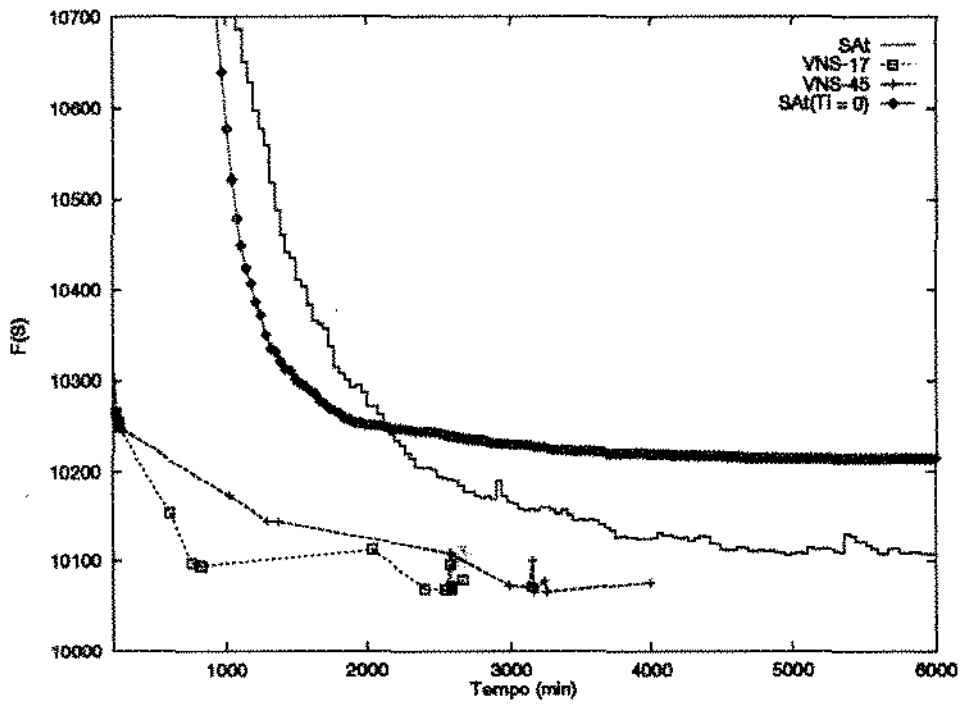


Figura 7.5: Gráfico *Custo*×*Tempo* para as execuções da SAt com $T_i = 0.35 \times 0.99^i$, SAt com $T_i = 0$, VNS-17 e VNS-45

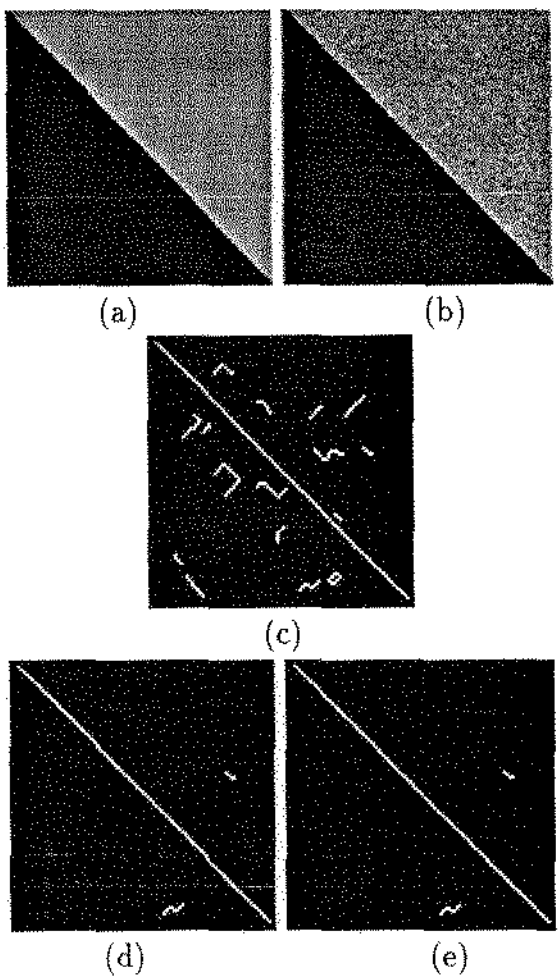
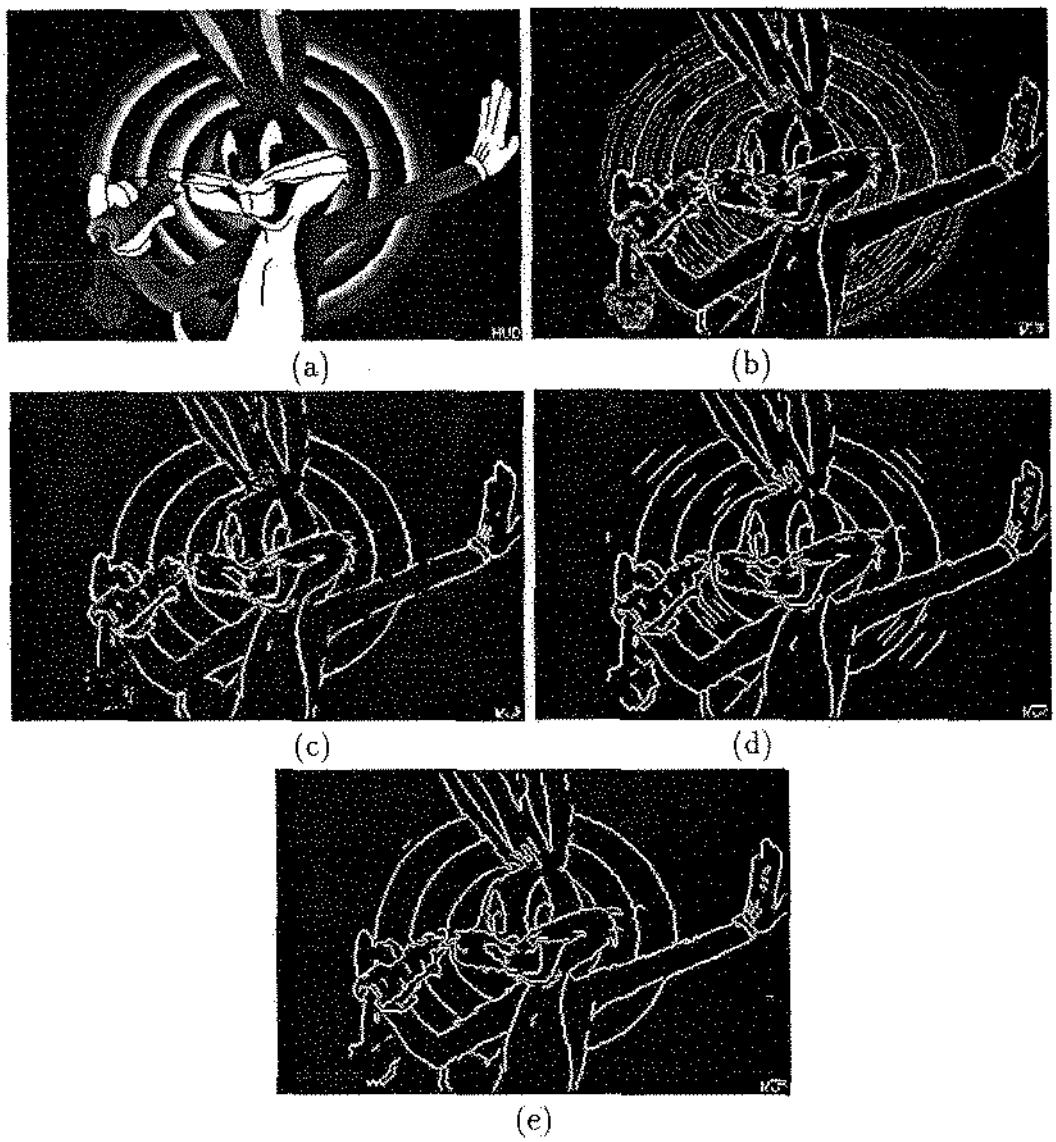


Figura 7.6: (a) Imagem original; (b) Imagem com ruído Gaussiano de variância $\sigma^2 = 144$; (c) Resultado da SAt (Custo 3372.72); (d) Resultado da VNS-17 (3361.34); (e) Resultado da VNS-45 (3361.34). Estes resultados foram obtidos para matriz de gradiente com limiar $\xi = 15$.



(a) (b) (c) (d) (e)

Figura 7.7: (a) Imagem original; (b) Matriz de gradiente para $\xi = 30$; (c) Resultado de limiar $\xi = 30$ aplicado sobre a gradiente; (d) Resultado da SAt (Custo: 10092.66); (e) Resultado da VNS-45 (10065.84).

Capítulo 8

Conclusões

Neste trabalho, apresentou-se o método heurístico de *busca em vizinhanças variáveis* [19] (vide seção 3.5), como uma nova maneira de minimizar a função de custos (vide cap. 4), proposta por *Tan et al.* [27], para o problema de deteção de contornos. O novo algoritmo foi denominado de VNSdb.

Das fases que compõem uma típica segmentação por deteção de bordas, a *binarização*, que pode ser caracterizada como uma busca pela configuração ótima das linhas de borda, é o processo-objeto de otimização nos trabalhos de *Tan et al.* [27], *Tan et al.* [28] e *Bhandarkar et al.* [3] (vide cap. 5), que utilizam os métodos heurísticos *busca local*, *simulated annealing* e *algoritmos genéticos*, respectivamente. Os referidos algoritmos foram implementados, e várias experiências foram realizadas, na busca da melhor configuração possível das implementações, a fim de que a VNSdb pudesse ser comparada criteriosamente com as abordagens anteriores.

A principal característica de uma *busca em vizinhanças variáveis* é o uso sistemático de vizinhanças distintas, a fim de imprimir uma busca que tenha capacidade de sair de ótimos locais indesejáveis. Para tanto, a VNSdb usa 4 vizinhanças distintas, que são utilizadas uma de cada vez pelo método. A VNSdb investiga suas vizinhanças de maneira determinística, ou seja, ao contrário da *simulated annealing* de *Tan et al.*, a SAt, que sorteia uma solução S' vizinha da solução corrente S, para então compará-las, a VNSdb compara a solução corrente com todas as soluções pertencentes ao conjunto-vizinhança utilizado naquele momento. Isto parece conferir uma convergência mais rápida à VNSdb, mas não é argumento suficiente para deduzir que a VNSdb consegue melhores soluções finais que a SAt, pois simplesmente a VNSdb pode ser retida em um ótimo local durante a busca, enquanto a SAt, com sua busca estocástica, pode chegar a uma solução final de qualidade bem superior. No entanto, a VSNdb mostrou-se superior, talvez pela sua constante alternância de vizinhanças, que lhe confere uma facilidade para superar ótimos

locais de pouca qualidade.

Com relação ao algoritmo genético de *Bhandarkar at al.* [3], o AGdb, sua implementação não se apresentou competitiva em relação à VNSdb e à SAt. Novos operadores foram propostos, que conseguiram sensíveis melhorias no AGdb (vide seção 5.2.1). Apesar disto, suas soluções finais, não chegam a superar os custos para nenhum dos resultados da SAt e da VNSdb, além dos tempos computacionais serem visivelmente desfavoráveis para o AGdb. Sua má performance deve estar associada com o tamanho da população de soluções, que não pode ser numerosa, devido a problemas de memória das máquinas utilizadas. Além disso, mesmo que fosse possível manter uma grande população, a carga computacional dos operadores do AGdb, que envolvem procedimentos com um grande número de imagens, certamente ainda comprometeria seus tempos computacionais.

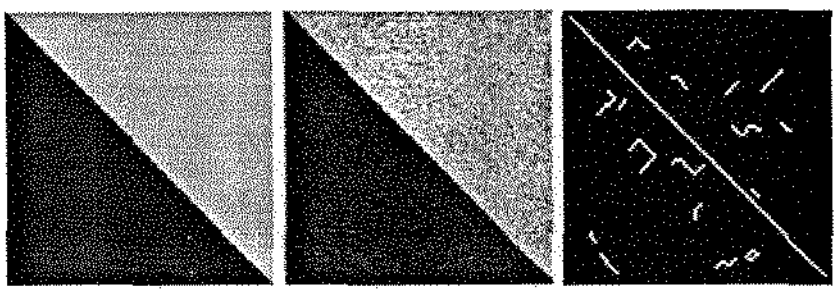
Enfim, todo os resultados indicam a boa qualidade da VNSdb, sem exceções dentro dos experimentos realizados.

Trabalhos futuros Uma extensão do trabalho seria a paralelização da abordagem. Dado que a função de custo em um ponto p sofre uma influência dos pontos que estão em sua janela $(p, 7 \times 7)$, pode-se paralelizar a busca sobre as imagens soluções, através da criação de conjuntos de pontos que sejam independentes entre si em relação à função de custos, os quais poderiam ser operados ao mesmo tempo, sem que houvesse influências negativas entre as mudanças realizadas sobre as soluções. Uma reavaliação da função de custos utilizada também seria interessante, na medida que indique novos critérios que conduzam os algoritmos a resultados de melhor qualidade.

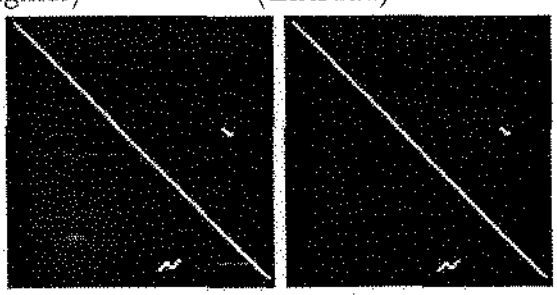
Apêndice A

VNSdb × SAt

A.1 (100×100), $\xi = 15$



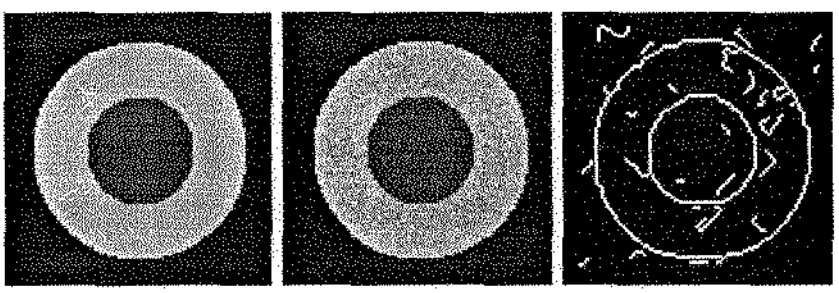
(Original) (Entrada) SAt



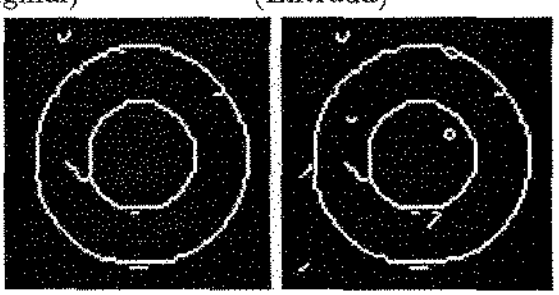
VNS-45 VNS-17

Tempo de CPU (min)	10	20	40	80	120
SAt	3373.63	3372.72			
VNS-45	3361.34				
VNS-17	3361.34				

A.2 (100×100), $\xi = 10$



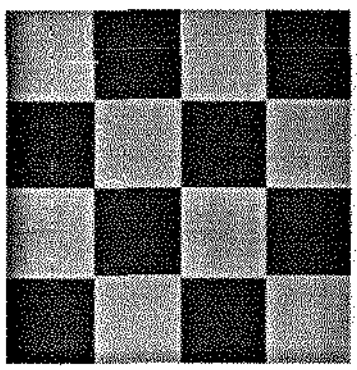
(Original) (Entrada) SAt



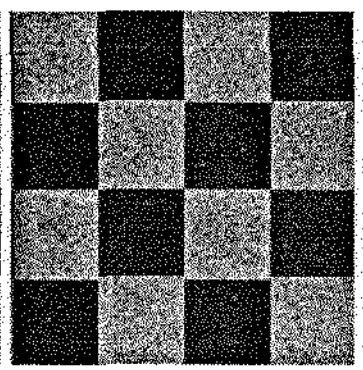
VNS-45 VNS-17

Tempo de CPU (min)	10	20	40	80	120
SAt	3732.98	3724.15	3724.15		
VNS-45	3712.84				
VNS-17	3712.61				

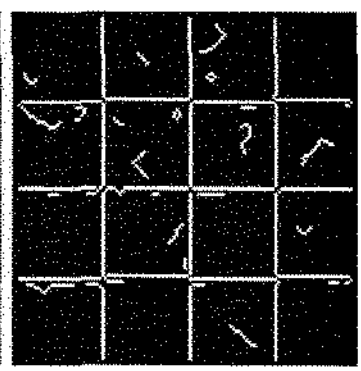
A.3 (128×128), $\xi = 15$



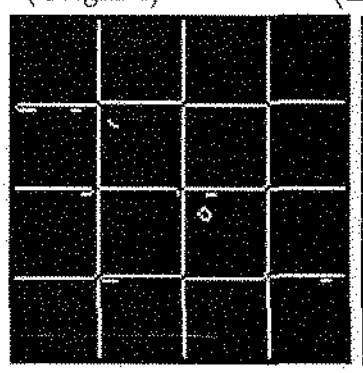
(Original)



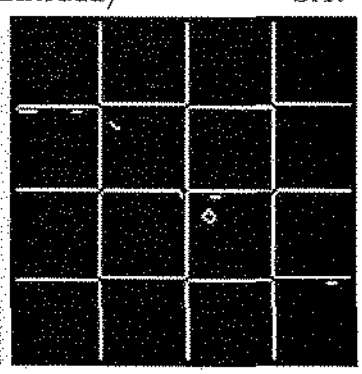
(Entrada)



SAt



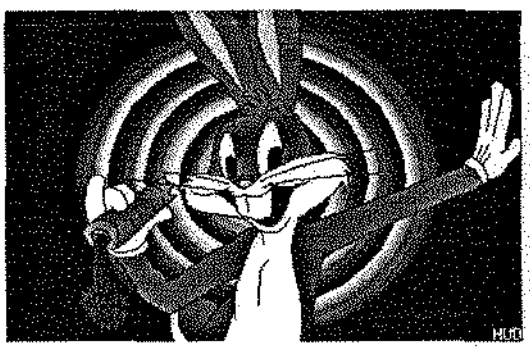
VNS-45



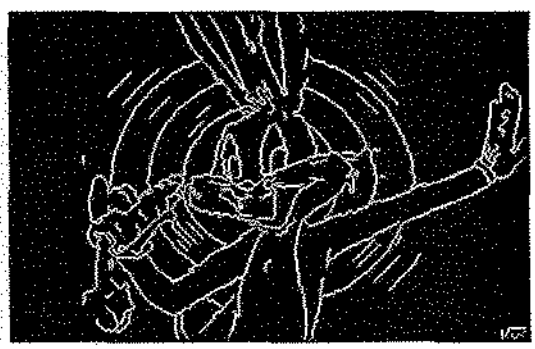
VNS-17

Tempo de CPU (min)	10	20	40	80	120
SAt	6171.76	6133.95	6127.08		
VNS-45	6112.10				
VNS-17	6112.73	6112.39			

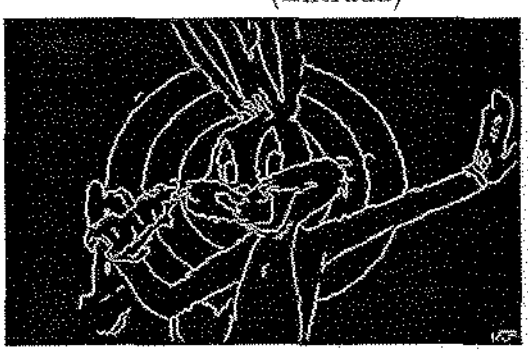
A.4 (320 × 200), $\xi = 30$



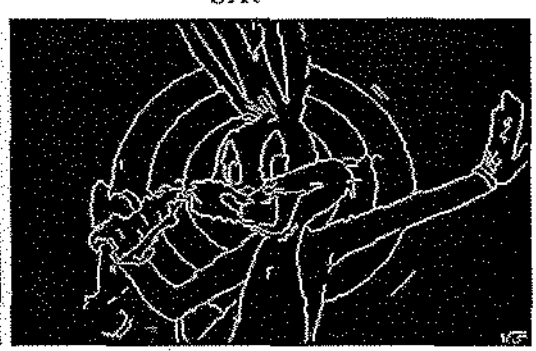
(Entrada)



SAt



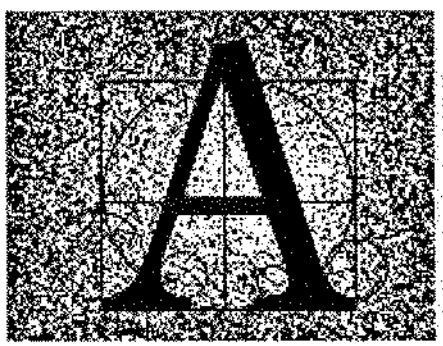
VNS-45



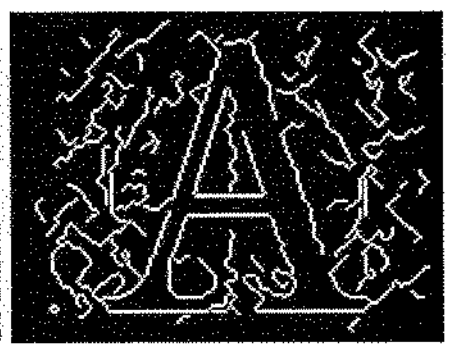
VNS-17

Tempo de CPU (min)	10	20	40	80	120
SA	14547.02	10804.21	10262.84	10121.72	10092.66
VNS-45	10172.87	10143.58	10108.00	10065.84	
VNS-17	10097.35	10093.78	10067.97	10067.91	

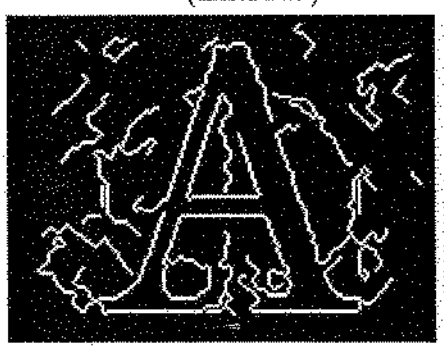
A.5 (160×120), $\xi = 70$



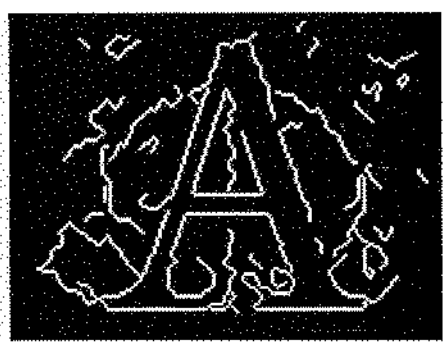
(Entrada)



SAt



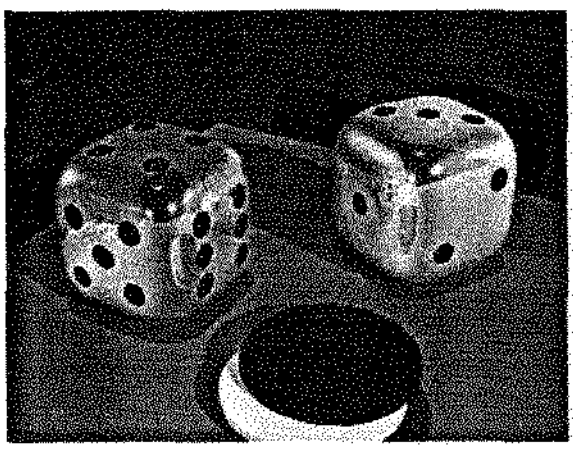
VNS-45



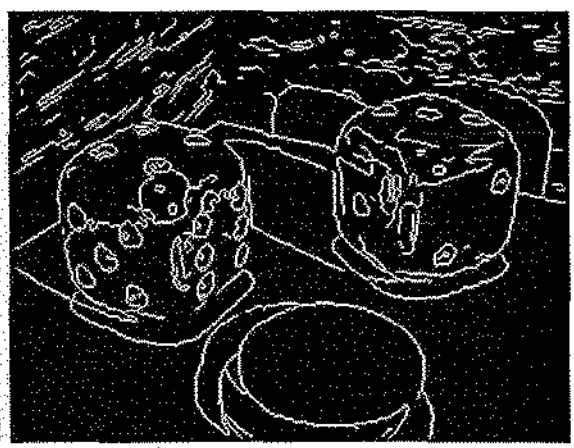
VNS-17

Tempo de CPU (min)	10	20	40	80	120
SA	7681.96	7543.40	7519.40	7517.07	
VNS-45	7500.81	7496.70			
VNS-17	7517.81	7497.30			

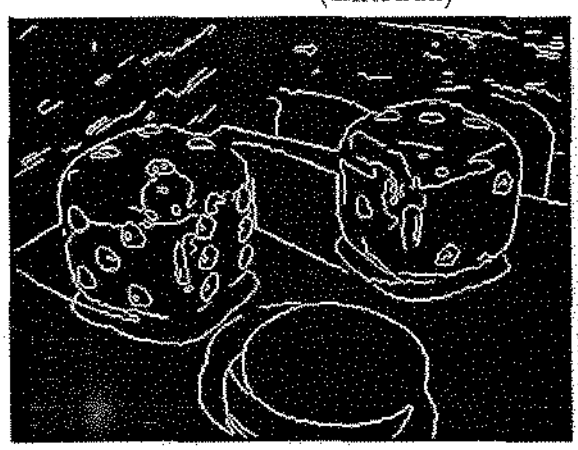
A.6 (320×240), $\xi = 15$



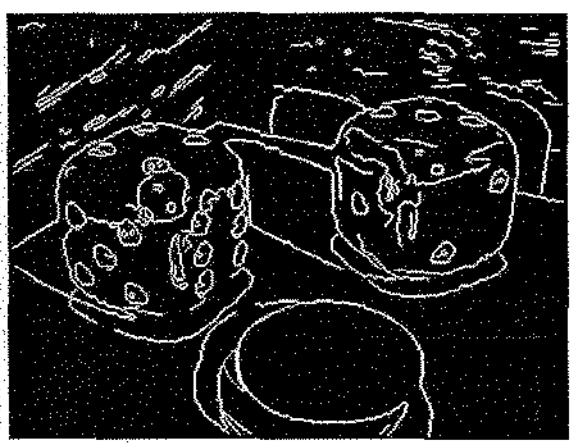
(Entrada)



SAt



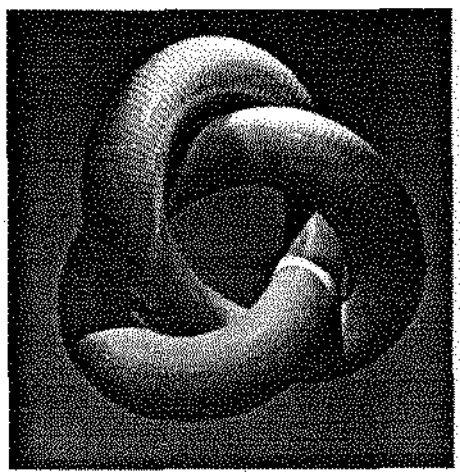
VNS-45



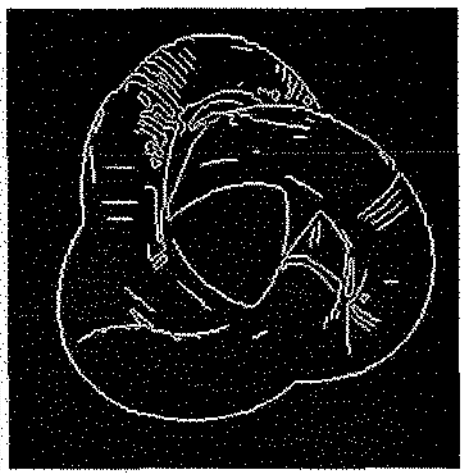
VNS-17

Tempo de CPU (min)	10	20	40	80	120
SA	34084.68	26305.48	21535.94	20770.15	20652.17
VNS-45	20627.47	20573.26	20571.52	20546.86	
VNS-17	20658.48	20546.16	20546.16	20546.16	

A.7 (256×256), $\xi = 10$



(Entrada)



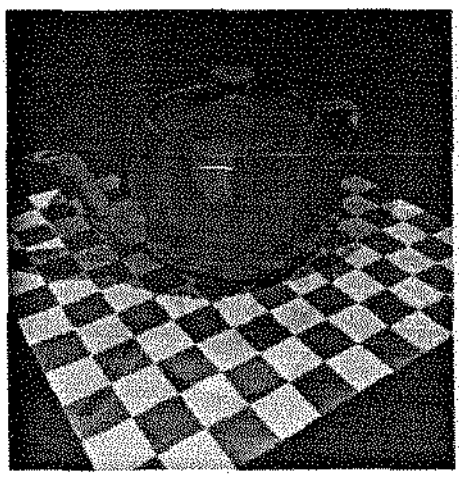
SAt

VNS-45

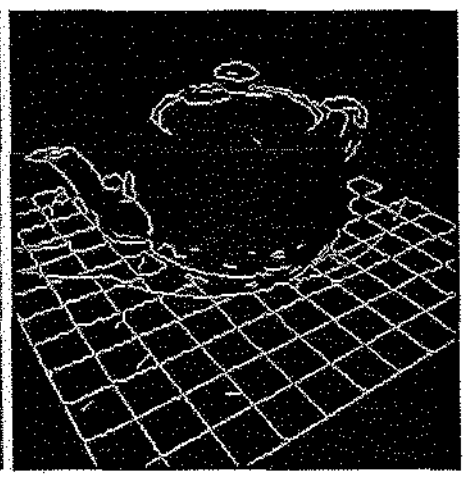
VNS-17

Tempo de CPU (min)	10	20	40	80	120
SA	10933.10	9024.15	8812.20	8752.65	8745.44
VNS-45	8774.81	8753.55	8744.44	8740.30	
VNS-17	8746.99	8745.70	8745.70		

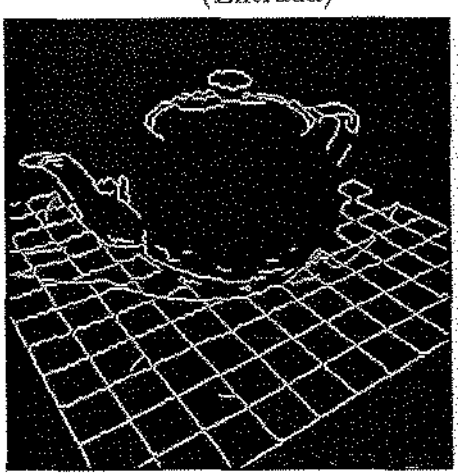
A.8 (256 × 256), ξ = 10



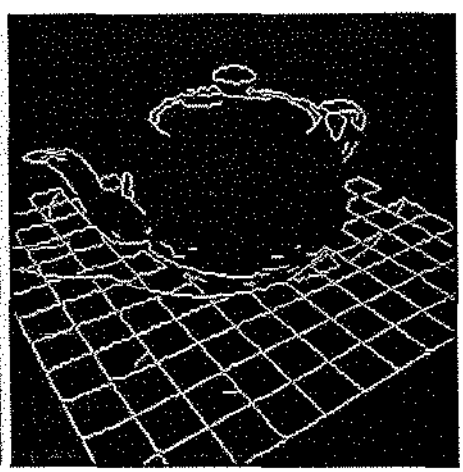
(Entrada)



SAt



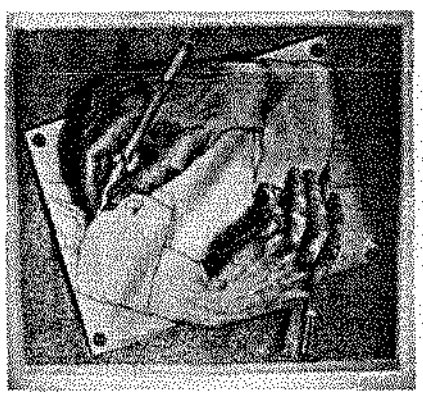
VNS-45



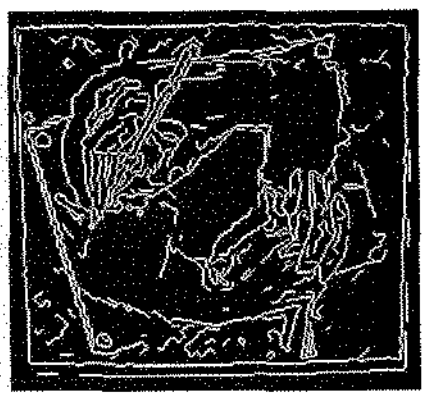
VNS-17

Tempo de CPU (min)	10	20	40	80	120
SA	10247.67	9141.87	9050.35	9018.05	9013.18
VNS-45	9048.82	9018.33	9017.45	9006.74	
VNS-17	9013.43	9011.62	9011.62	9011.62	

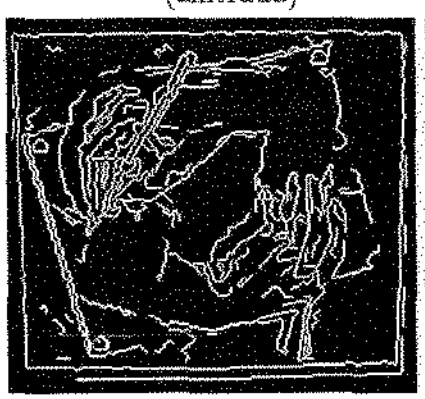
A.9 (234×212), $\xi = 15$



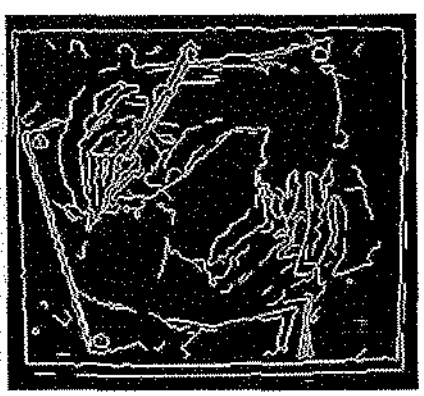
(Entrada)



SAt



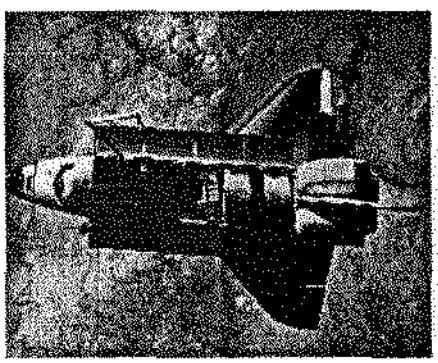
VNS-45



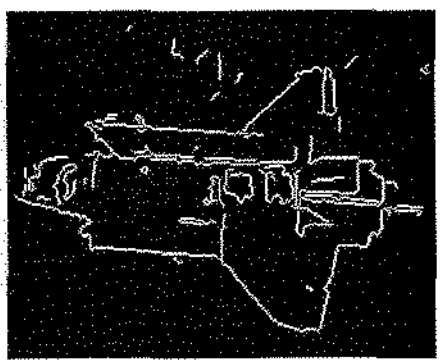
VNS-17

Tempo de CPU (min)	10	20	40	80	120
SA	17964.15	16609.65	16263.36	16131.74	16115.88
VNS-45	16161.29	16118.91	16118.59	16102.32	
VNS-17	16198.40	16096.38	16096.38	16096.38	

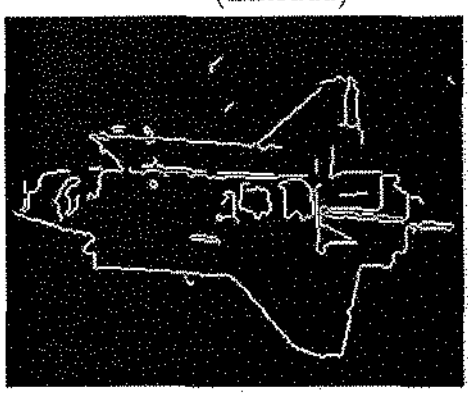
A.10 (244×194), $\xi = 40$



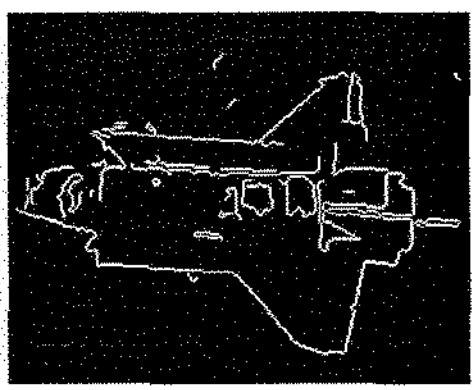
(Entrada)



SAt



VNS-45

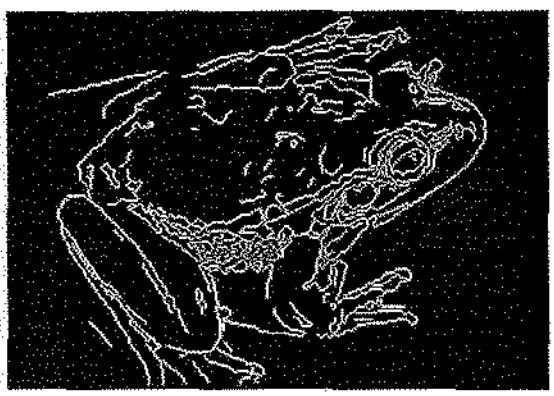


VNS-17

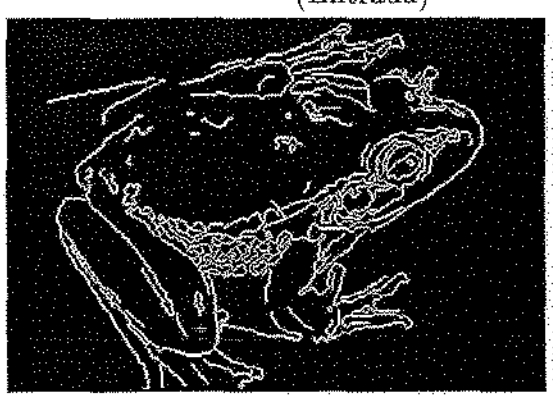
Tempo de CPU (min)	10	20	40	80	120
SA	11416.34	10627.70	10561.92	10548.84	10548.42
VNS-45	10562.35	10551.86	10550.44	10547.54	
VNS-17	10549.85	10549.85	10547.42	10547.42	

A.11 (308×211), $\xi = 15$

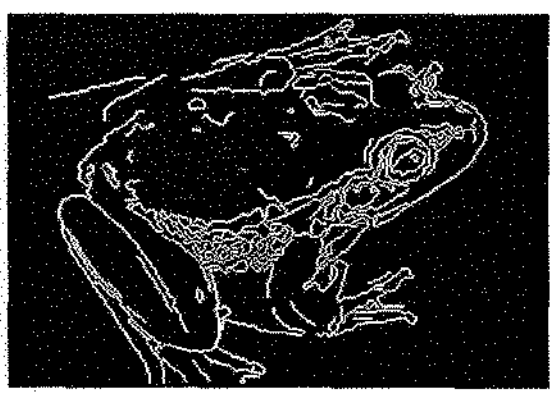
(Entrada)



SAt



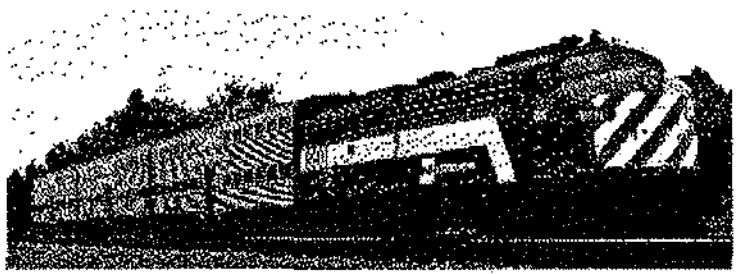
VNS-45



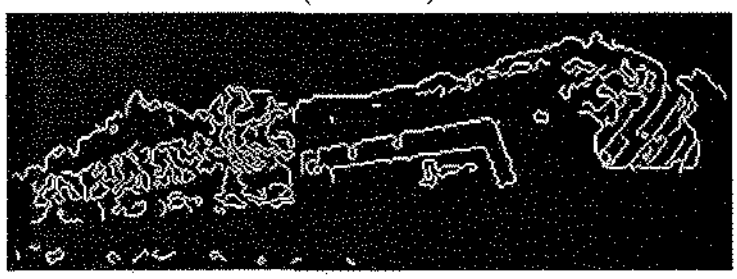
VNS-17

Tempo de CPU (min)	10	20	40	80	120
SA	20014.49	14211.07	12980.95	12763.38	12710.46
VNS-45	12728.74	12679.77	12678.05	12667.85	
VNS-17	12765.03	12665.09	12665.09	12665.09	

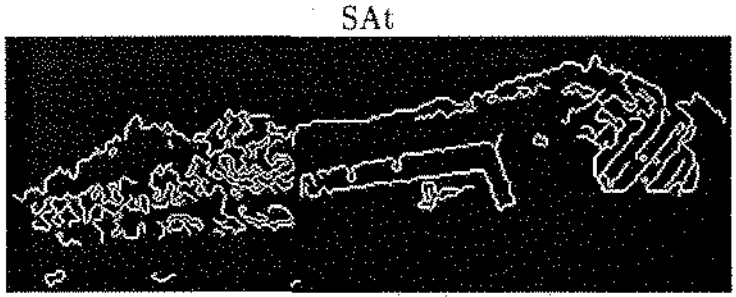
A.12 (359×125), $\xi = 70$



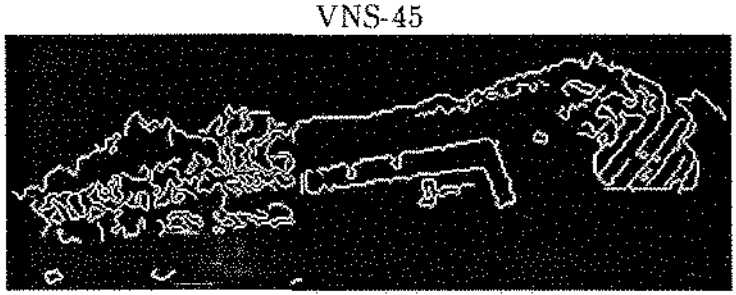
(Entrada)



SAt



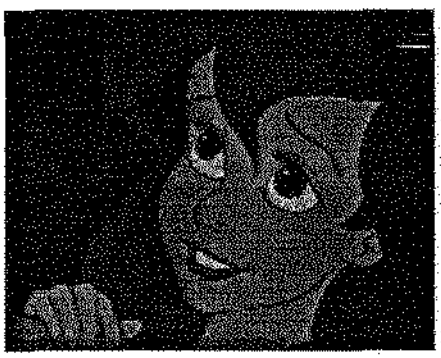
VNS-45



VNS-17

Tempo de CPU (min)	10	20	40	80	120
SA	9227.22	8612.11	8489.69	8450.67	8445.95
VNS-45	8482.33	8482.33	8446.54	8443.22	8443.22
VNS-17	8450.71	8446.57	8445.34	8445.34	

A.13 (247×190), $\xi = 8$



(Entrada)

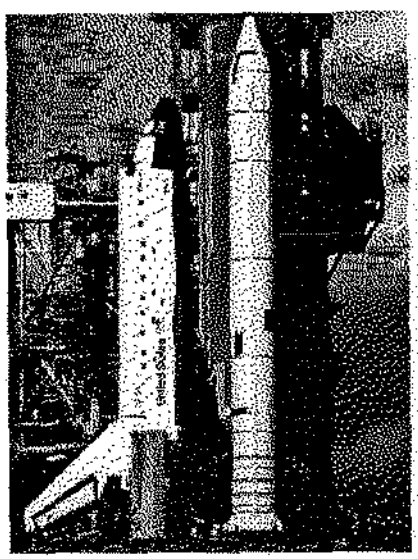
SAt

VNS-45

VNS-17

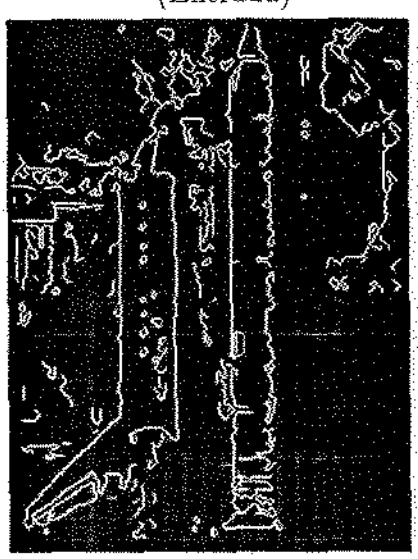
Tempo de CPU (min)	10	20	40	80	120
SA	12692.57	11984.00	11880.98	11855.14	11852.08
VNS-45	11874.53	11853.48	11852.25	11844.67	11844.67
VNS-17	11849.86	11848.50	11848.50	11847.65	

A.14 (248×326), $\xi = 70$

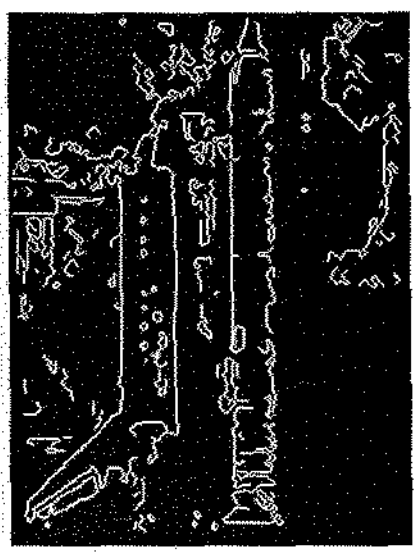


(Entrada)

SAt



VNS-45



VNS-17

Tempo de CPU (min)	10	20	40	80	120
SA	22853.89	18289.71	17350.81	17137.65	17090.69
VNS-45	17097.59	17055.68	17054.65	17054.42	17054.42
VNS-17	17172.27	17055.27	17055.27	17055.27	

A.15 (243×319), $\xi = 15$

(Entrada)

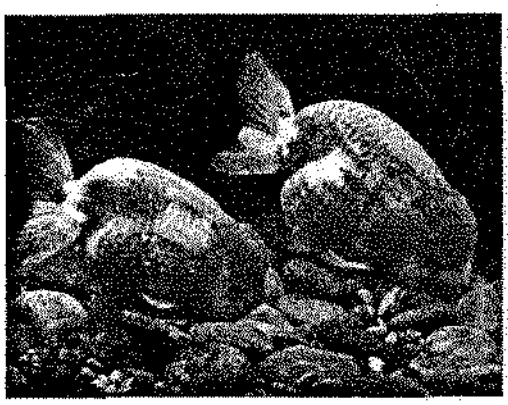
SAt

VNS-45

VNS-17

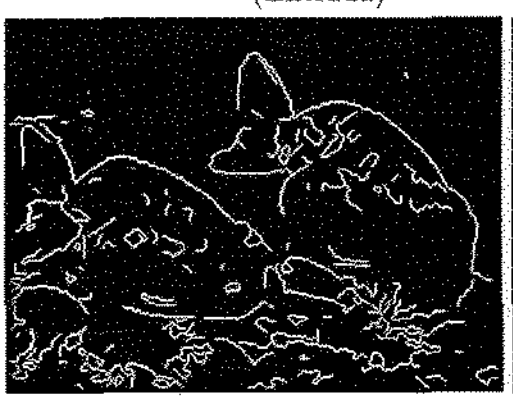
Tempo de CPU (min)	10	20	40	80	120
SA	27808.53	23211.16	22335.11	22143.50	22107.41
VNS-45	22134.79	22134.79	22085.39	22062.10	22062.10
VNS-17	22319.85	22181.79	22061.65	22057.22	22057.22

A.16 (309×231), $\xi = 30$



(Entrada)

SAt



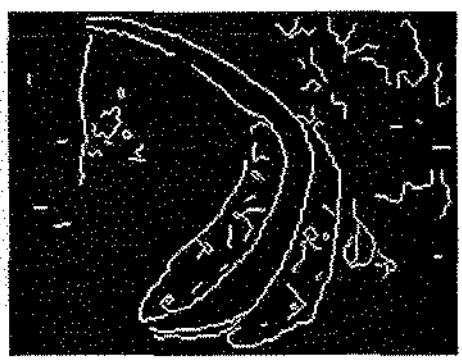
VNS-45

VNS-17

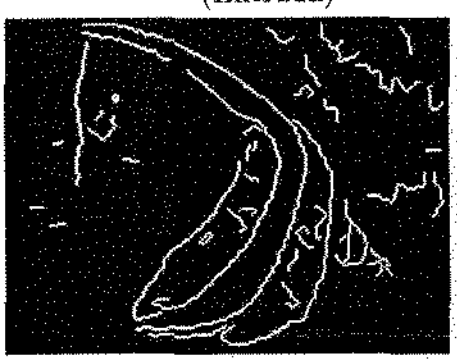
Tempo de CPU (min)	10	20	40	80	120
SA	26505.57	22857.57	21836.99	21646.26	21613.87
VNS-45	21640.20	21599.77	21598.69	21576.62	
VNS-17	21685.25	21578.71	21578.71	21578.71	

A.17 (221×165), $\xi = 25$

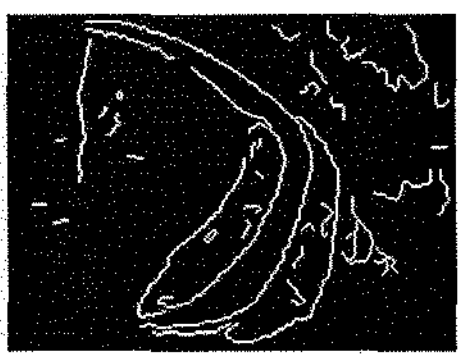
(Entrada)



SAt



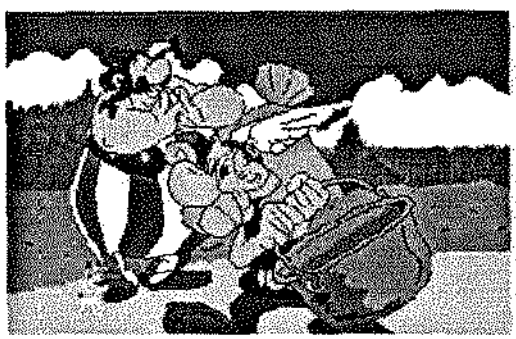
VNS-45



VNS-17

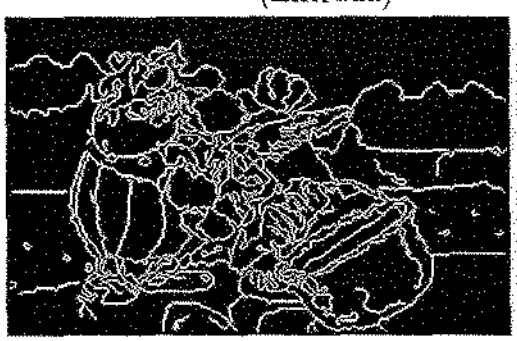
Tempo de CPU (min)	10	20	40	80	120
SA	8839.82	8590.19	8533.46	8525.10	8525.10
VNS-45	8526.62	8525.73	8518.66		
VNS-17	8519.79	8519.79	8518.48		

A.18 (288×180), $\xi = 25$



(Entrada)

SAt

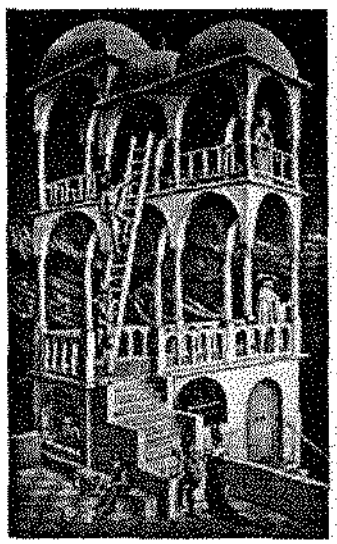


VNS-45

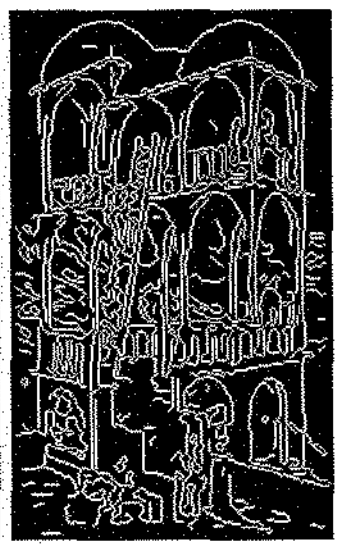
VNS-17

Tempo de CPU (min)	10	20	40	80	120
SA	15212.17	13777.12	13516.28	13415.16	13390.97
VNS-45	13433.74	13367.87	13366.76	13359.24	13359.24
VNS-17	13469.39	13372.15	13372.15	13372.15	

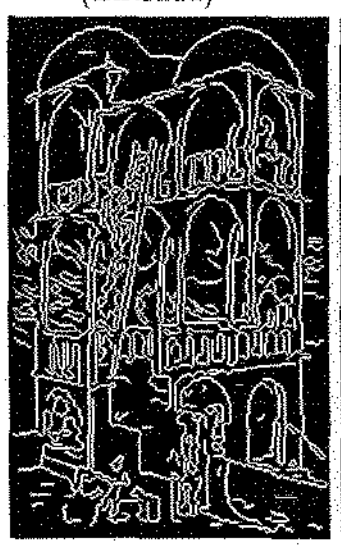
A.19 (171×274), $\xi = 15$



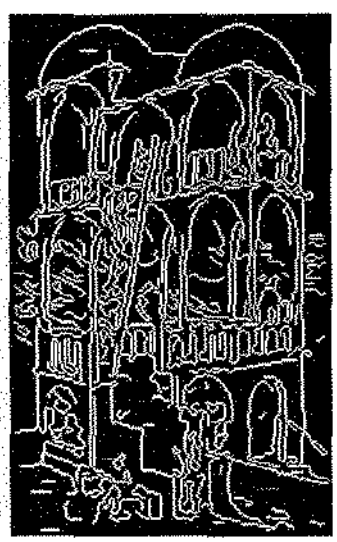
(Entrada)



SAt



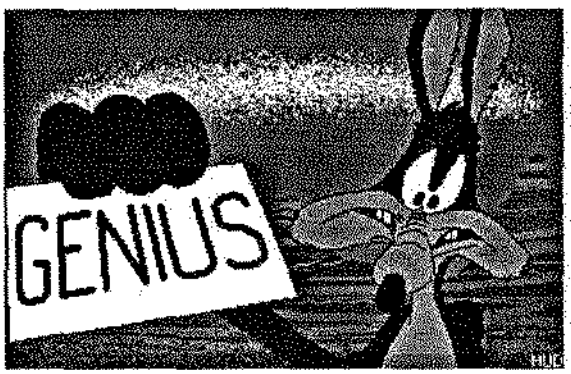
VNS-45



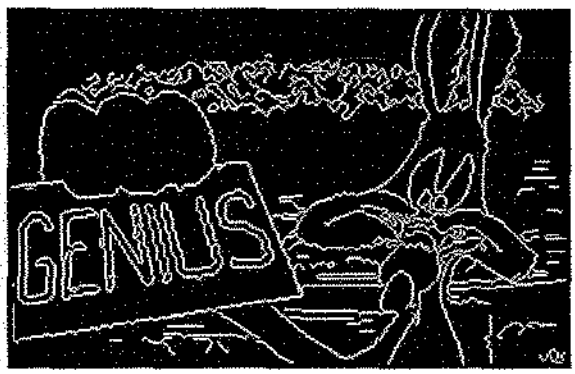
VNS-17

Tempo de CPU (min)	10	20	40	80	120
SA	19390.79	18060.97	17723.73	17588.29	17559.18
VNS-45	17588.05	17529.68	17529.33	17523.55	17513.09
VNS-17	17659.55	17524.44	17520.90	17520.90	

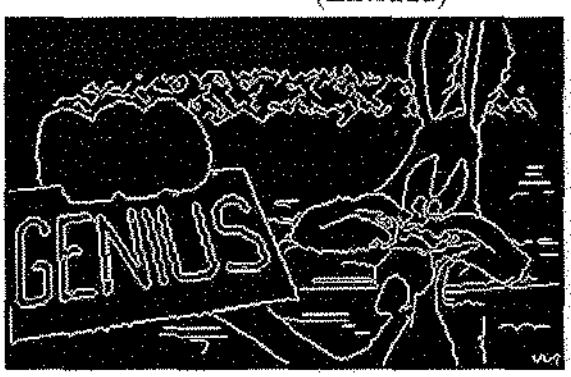
A.20 (320×200), $\xi = 30$



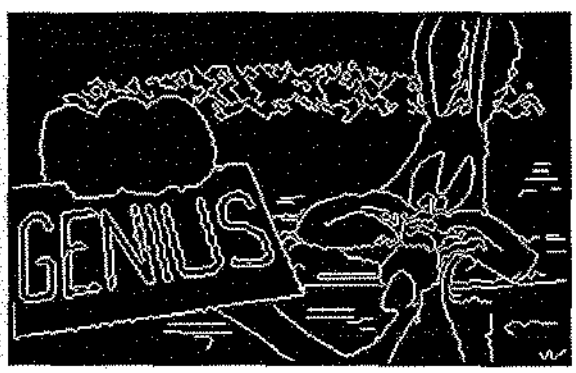
(Entrada)



SAt



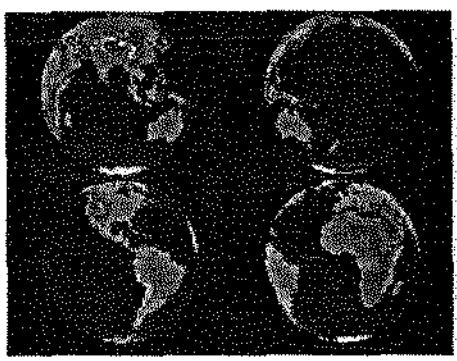
VNS-45



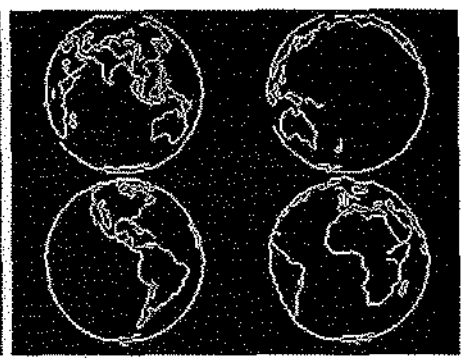
VNS-17

Tempo de CPU (min)	10	20	40	80	120
SA	18188.79	15704.52	15293.63	15141.90	15120.07
VNS-45	15169.46	15109.29	15107.28	15101.13	15089.04
VNS-17	15210.04	15093.45	15092.68	15092.68	

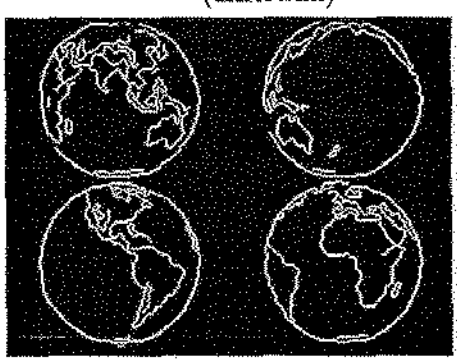
A.21 (256×192), $\xi = 15$



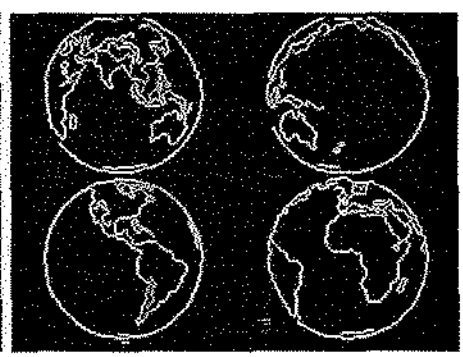
(Entrada)



SAt



VNS-45

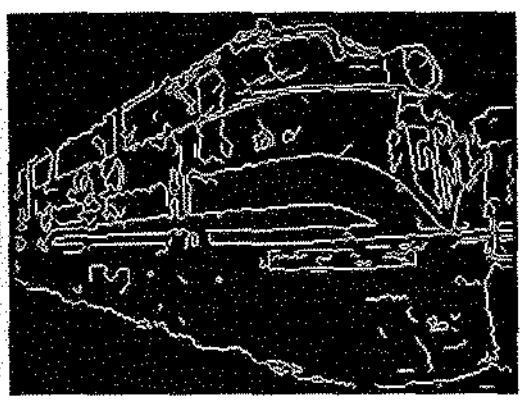


VNS-17

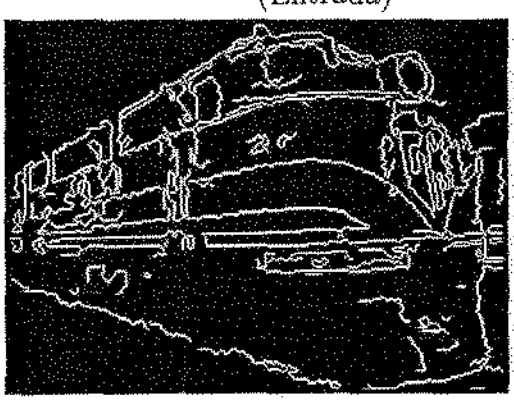
Tempo de CPU (min)	10	20	40	80	120
SA	7524.11	7094.65	7016.02	6992.65	6989.31
VNS-45	7022.42	6984.07	6984.07	6978.14	
VNS-17	6985.27	6982.39	6982.39	6982.39	

A.22 (288×212), $\xi = 40$

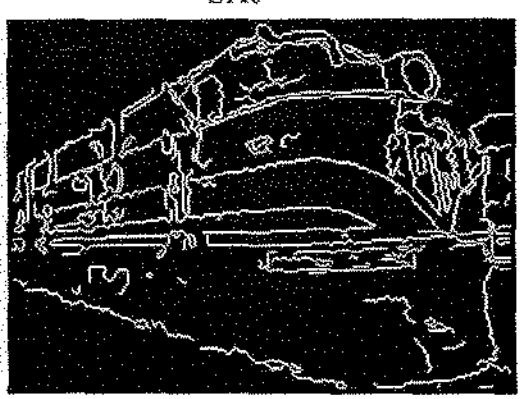
(Entrada)



SAt



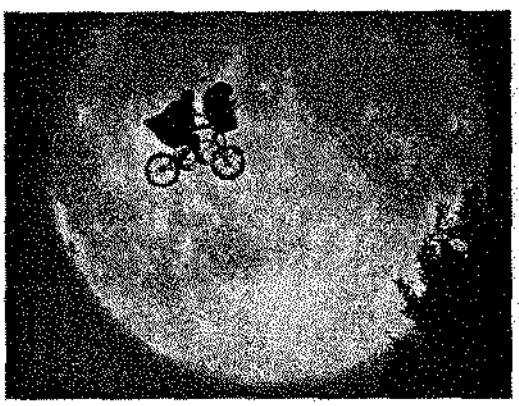
VNS-45



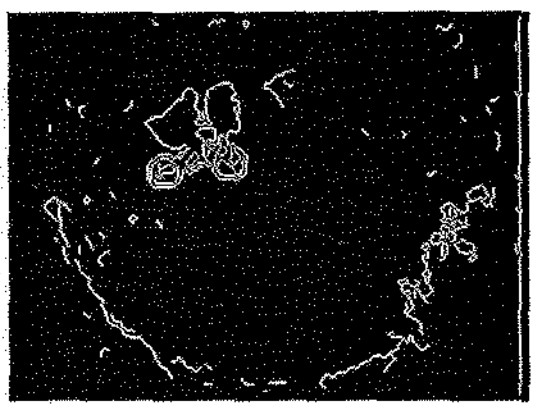
VNS-17

Tempo de CPU (min)	10	20	40	80	120
SA	22461.61	19464.58	18837.24	18685.02	18666.64
VNS-45	18702.32	18657.39	18656.69	18647.17	
VNS-17	18754.79	18647.43	18647.43	18647.43	

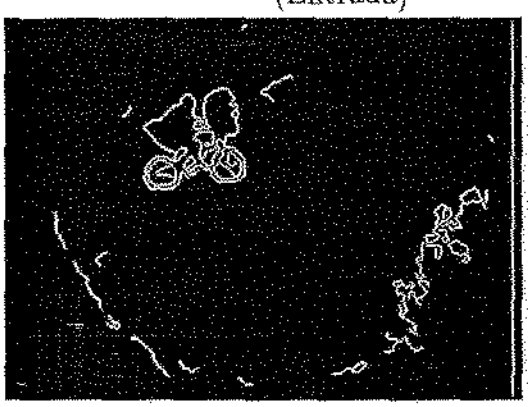
A.23 (295×216), $\xi = 25$



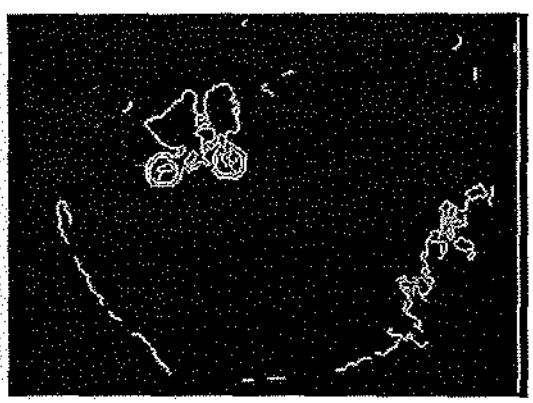
(Entrada)



SAt



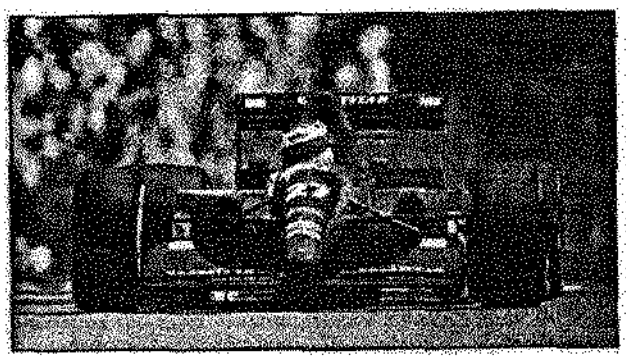
VNS-45



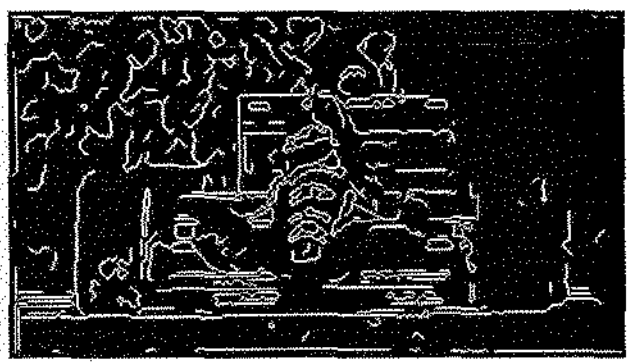
VNS-17

Tempo de CPU (min)	10	20	40	80	120
SA	23280.45	19815.59	19245.05	19194.33	19190.77
VNS-45	19214.77	19214.77	19195.17	19195.17	19181.59
VNS-17	19176.21	19175.25	19175.25		

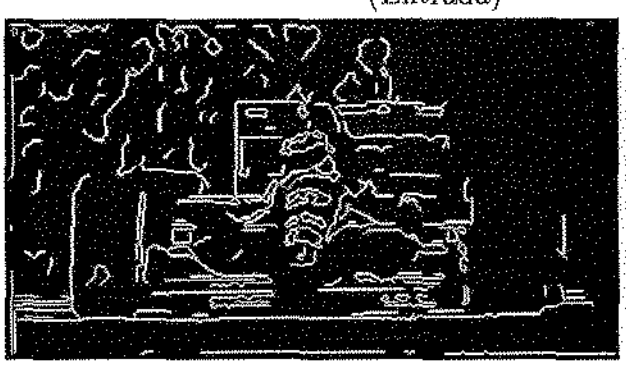
A.24 (351×193), $\xi = 25$



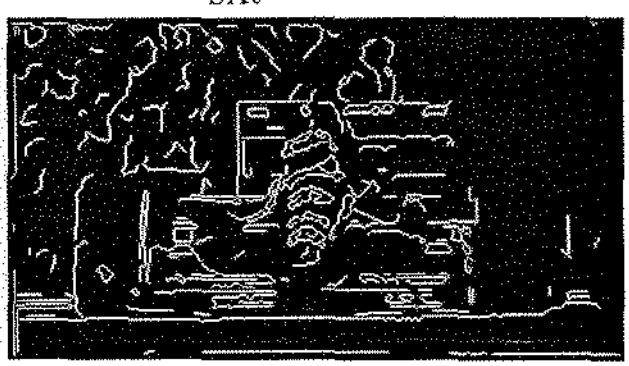
(Entrada)



SAt



VNS-45

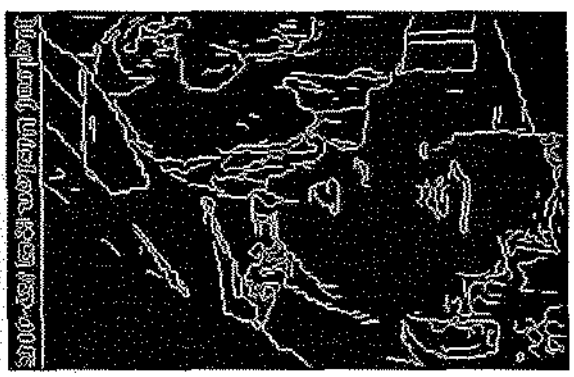


VNS-17

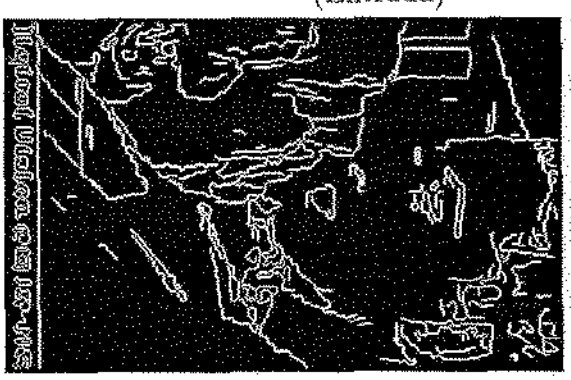
Tempo de CPU (min)	10	20	40	80	120
SA	26689.99	22892.72	22067.28	21912.11	21873.20
VNS-45	21908.43	21864.56	21862.27	21847.34	
VNS-17	21940.56	21839.86	21839.86	21839.86	

A.25 (320×200), $\xi = 10$

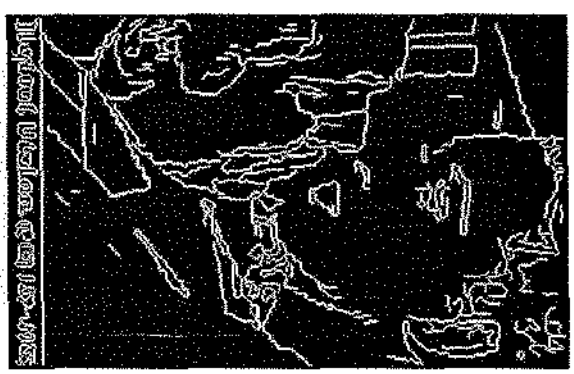
(Entrada)



SAt



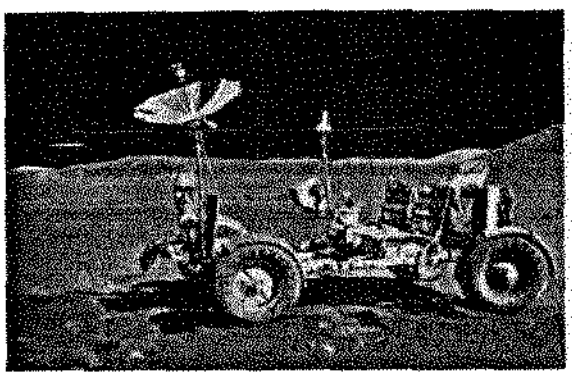
VNS-45



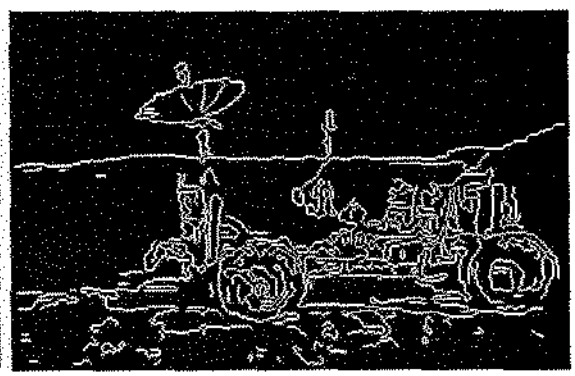
VNS-17

Tempo de CPU (min)	10	20	40	80	120
SA	22170.10	18380.31	17584.37	17420.05	17388.62
VNS-45	17435.71	17378.50	17376.35	17371.45	17371.45
VNS-17	17479.82	17372.20	17371.43	17371.43	

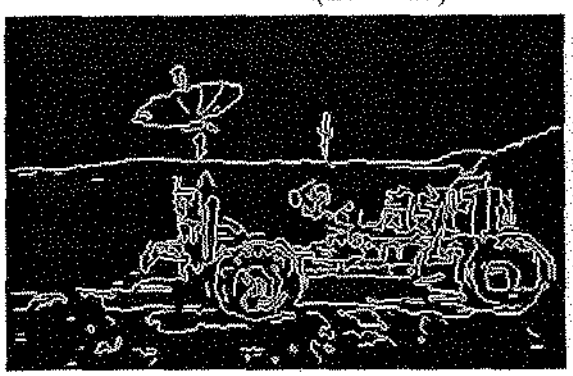
A.26 (320 × 200), $\xi = 15$



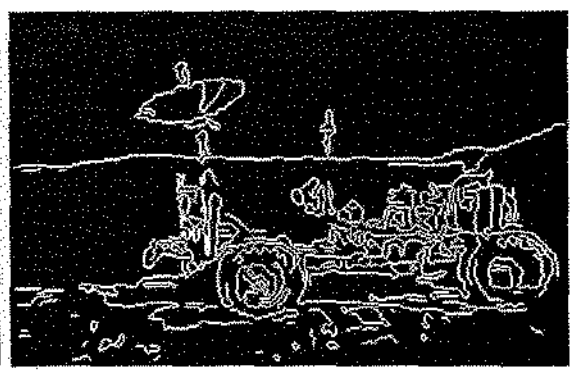
(Entrada)



SAt



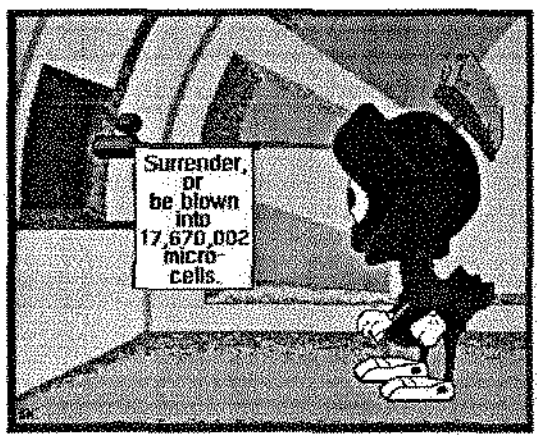
VNS-45



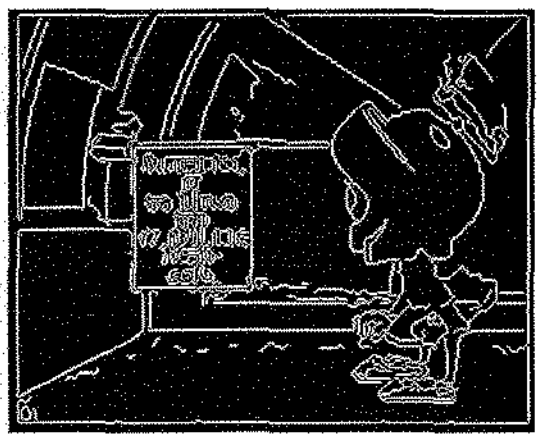
VNS-17

Tempo de CPU (min)	10	20	40	80	120
SA	24043.15	17699.47	15728.30	15428.92	15367.88
VNS-45	15388.00	15322.30	15319.09	15319.09	
VNS-17	15408.56	15311.29	15311.29	15311.29	

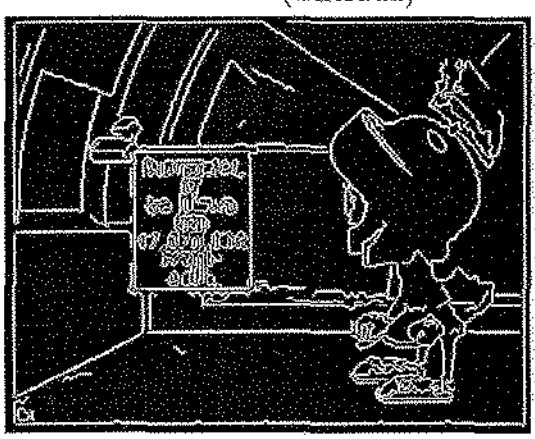
A.27 (300×235), $\xi = 20$



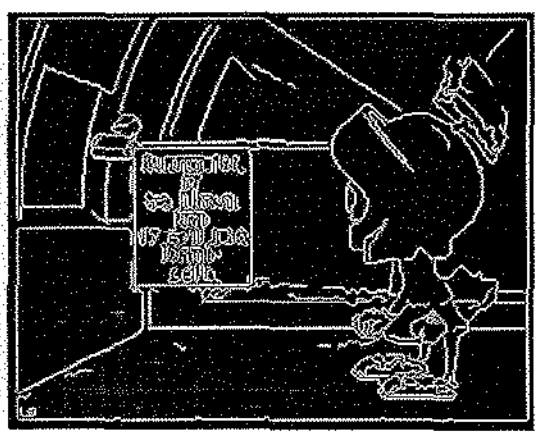
(Entrada)



SAt



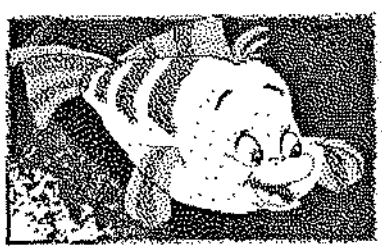
VNS-45



VNS-17

Tempo de CPU (min)	10	20	40	80	120
SA	16744.19	13642.93	13221.51	13116.27	13093.73
VNS-45	13141.03	13141.03	13093.45	13090.38	13090.38
VNS-17	13165.72	13081.95	13081.95	13081.95	

A.28 (212×128), $\xi = 70$



(Entrada)

SAt

VNS-45

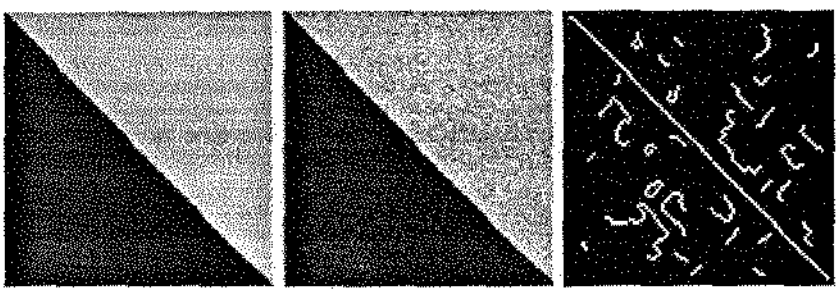
VNS-17

Tempo de CPU (min)	10	20	40	80	120
SA	8299.33	8086.75	8035.66	8023.97	8023.97
VNS-45	8019.78	8019.30	8015.80		
VNS-17	8019.00	8019.00	8019.00		

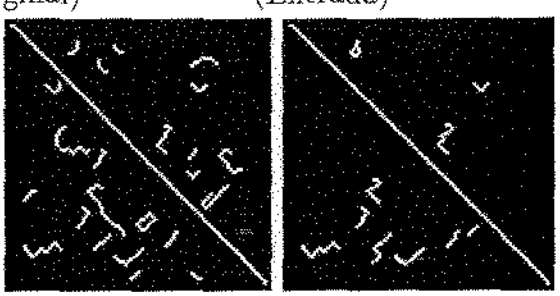
Apêndice B

SAt × BLdb

B.1 (100×100), $\xi = 10$



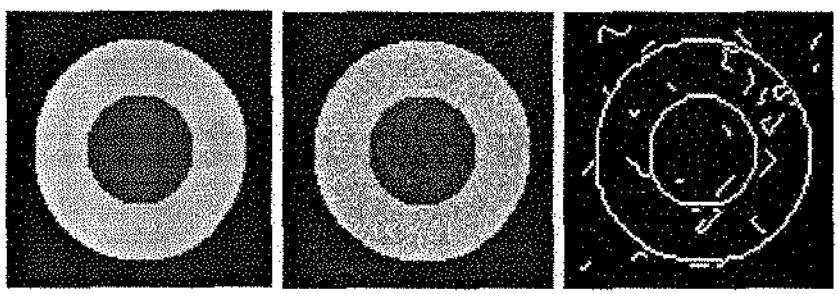
(Original) (Entrada) SAt



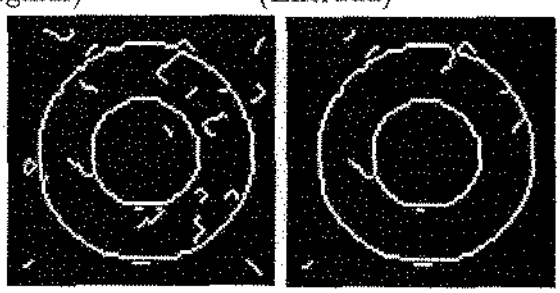
BL-45 BL-17

Tempo de CPU (min)	10	20	40	80	120
SAt ($T_i = 0.35 \times 0.99^i$)	3718.81	3712.40	3712.40		
BL-45	3701.32				
BL-17	**3693.34**				

B.2 (100×100), $\xi = 10$



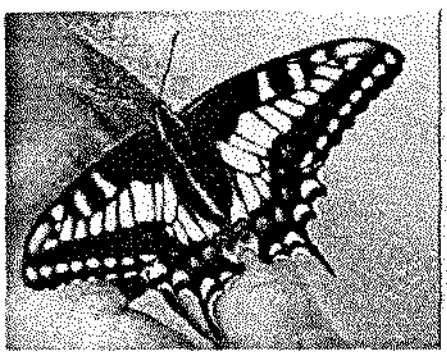
(Original) (Entrada) SAt



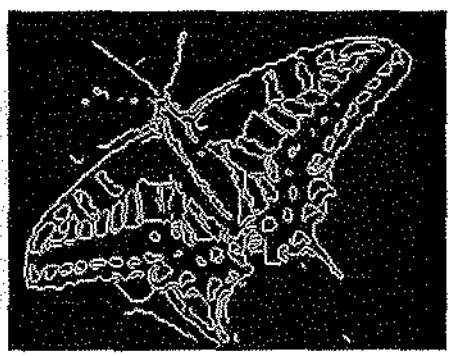
BL-45 BL-17

Tempo de CPU (min)	10	20	40	80	120
SAt ($T_i = 0.35 \times 0.99^i$)	3732.98	3724.15	3724.15		
BL-45	3720.43	3720.43			
BL-17	**3711.58**				

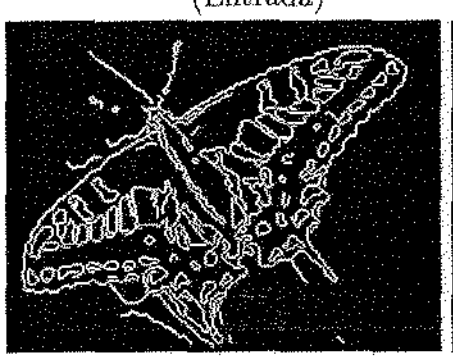
B.3 (233×175), $\xi = 20$



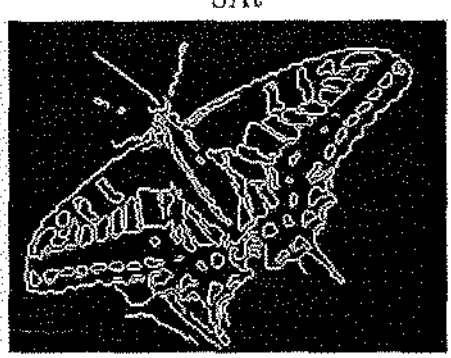
(Entrada)



SAt



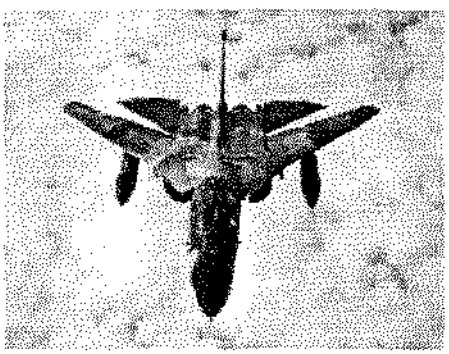
BL-45



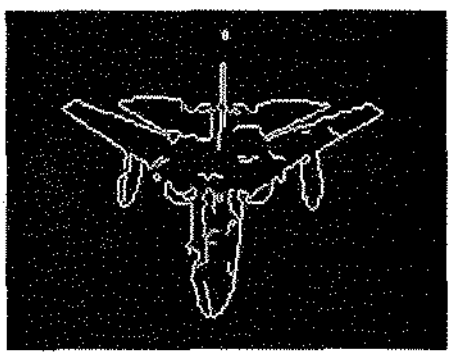
BL-17

Tempo de CPU (min)	10	20	40	80	120
SAt ($T_i = 0.35 \times 0.99^i$)	13836.07	12976.91	12805.72	12754.17	12742.67
BL-45	12768.24	12764.09	12751.35	12744.18	
BL-17	**12768.02**	**12740.12**	**12734.98**		

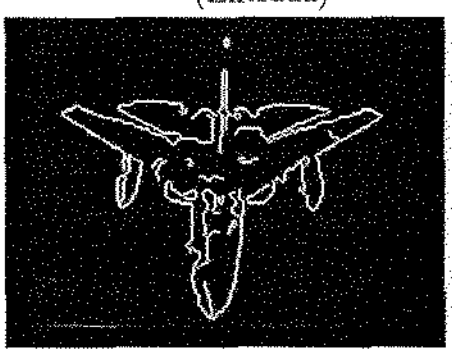
B.4 (233 × 174), ξ = 25



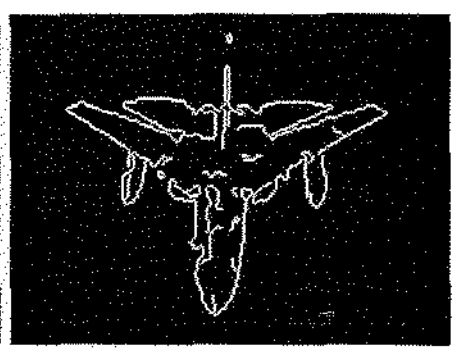
(Entrada)



SAt



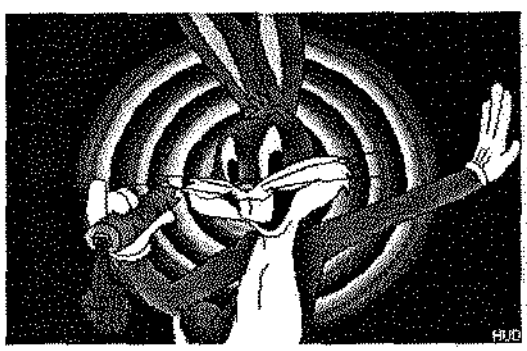
BL-45



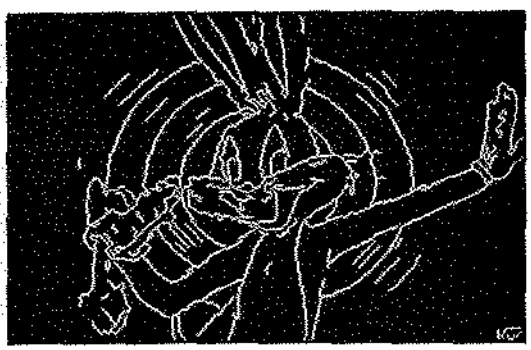
BL-17

Tempo de CPU (min)	10	20	40	80	120
SAt ($T_i = 0.35 \times 0.99^l$)	**5111.92**	**5034.84**	**5017.13**	**5011.96**	
BL-45	5017.66	5017.06	5012.56	5012.46	
BL-17	5018.59	5012.19			

B.5 (320 × 200), $\xi = 30$

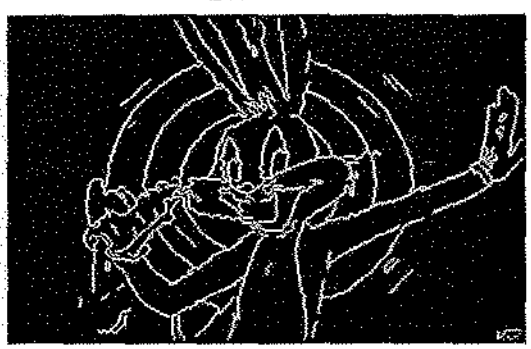


(Entrada)



SAt

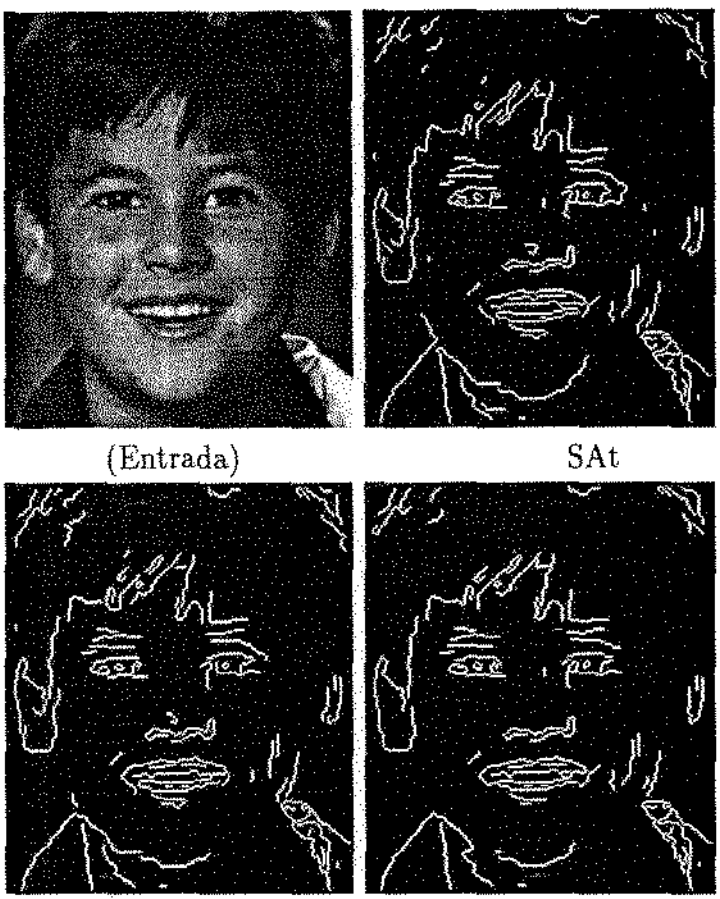
BL-45



BL-17

Tempo de CPU (min)	10	20	40	80	120
SAt ($T_i = 0.35 \times 0.99^i$)	12643.32	10560.42	10184.76	10106.42	10092.06
BL-45	10171.82	10150.84	10130.50	10123.59	
BL-17	**10130.31**	**10076.11**	**10073.58**		

B.6 (185 × 216), $\xi = 14$

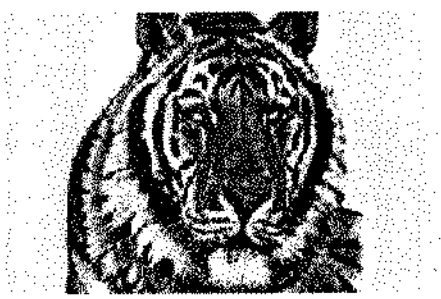


(Entrada) SAt

BL-45 BL-17

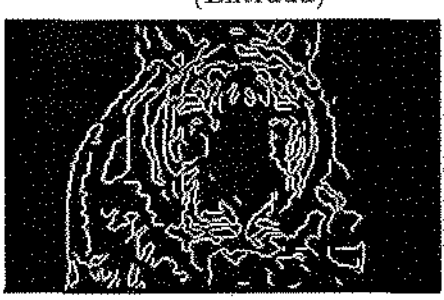
Tempo de CPU (min)	10	20	40	80	120
SAt ($T_i = 0.35 \times 0.99^i$)	11474.90	11131.47	11073.06	11057.58	11057.43
BL-45	**11062.24**	**11061.06**	**11052.14**	**11052.14**	
BL-17	11075.59	11058.93			

B.7 (233×145), $\xi = 45$

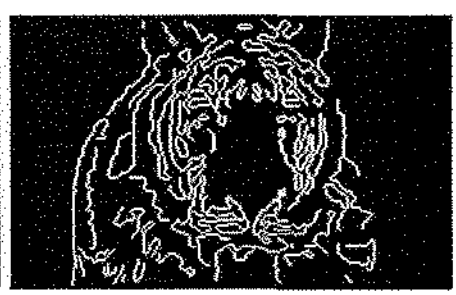


(Entrada)

SAt



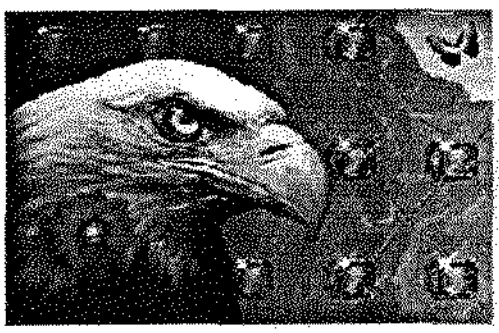
BL-45



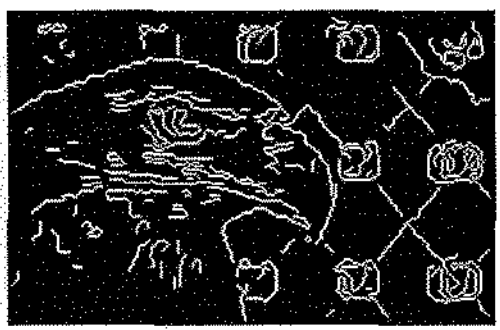
BL-17

Tempo de CPU (min)	10	20	40	80	120
SAt ($T_i = 0.35 \times 0.99^i$)	8250.59	7833.56	7714.29	7688.39	7686.84
BL-45	7709.24	7709.05	7689.87	7689.87	
BL-17	**7694.35**	**7685.35**			

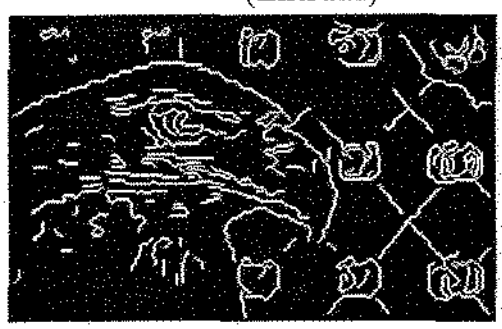
B.8 (259×162), $\xi = 20$



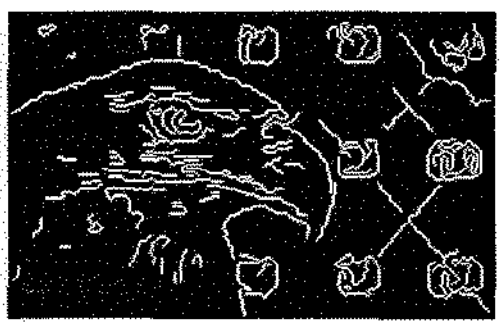
(Entrada)



SAt



BL-45

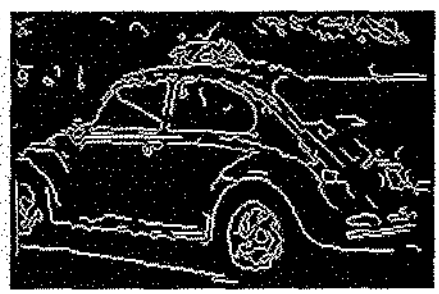


BL-17

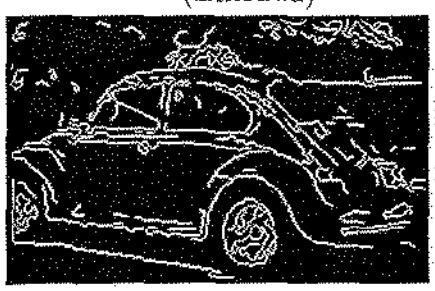
Tempo de CPU (min)	10	20	40	80	120
SAt ($T_i = 0.35 \times 0.99^i$)	11376.46	10579.09	10438.96	10394.00	10387.76
BL-45	10402.29	10401.85	10382.13	10381.97	
BL-17	**10396.97**	**10378.26**	**10376.34**		

B.9 (225×142), $\xi = 20$

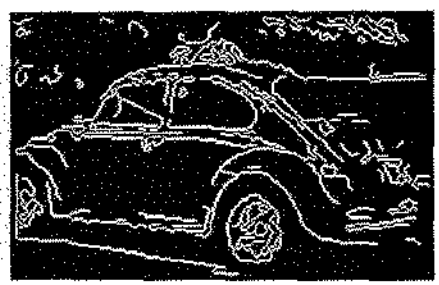
(Entrada)



SAt



BL-45



BL-17

Tempo de CPU (min)	10	20	40	80	120
SAt ($T_i = 0.35 \times 0.99^i$)	8943.53	8595.45	8498.61	8477.02	8476.71
BL-45	8497.16	8494.87	8482.85		
BL-17	**8479.61**	**8470.06**			

Bibliografia

[1] E. H. L. Aarts and P. J. M. van Laarhoven. Statistical cooling: a general approach to combinatorial optimization problems. *Philips J. Res.*, 40:193–226, 1985.

[2] S. T. Acton and A. C. Bovik. Anisotropic edge detection using mena field annealing. In *Proc. IEEE Intl Conf. Acoustics Speech and Signal Processing*, volume II, pages 393–386, 1992.

[3] Suchendra M. Bhandarkar, Yiqing Zhang, and Walter D. Potter. An edge detection technique using gentic algorithm-based optimization. *Pattern Recognition*, 27:1159–1180, 1994.

[4] J. Canny. A computacional approach to edge detection. *IEEE Transactions Systems Man. Cybern.*, 8(6):679–698, 1986.

[5] P. Carnevalli, L. Coletti, and S. Patarnello. Image processing by simulated annealing. *IBM Journal of Research and Development*, 29(6):569–579, november 1985.

[6] D. de Werra and A. Hertz. Tabu search tecniques. *OR Spektrum*, 11:131–141, 1989.

[7] E. J. Delp and C. H. Chu. Detecting edge segments. *IEEE Transactions on Systems Man and Cybernetics*, SMC-15(1):144–152, 1985.

[8] Rachid Derriche. Using canny's criteria to derive a recursively implemented optimal edge detector. *International Journal of Computer Vision*, pages 167–187, 1987.

[9] R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis*. Wiley, New York, 1973.

[10] Thomas A. Feo and Mauricio G. C. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 1995.

[11] K. S. Fu and J. K. Mu. A survey on image segmentation. *Pattern Recognition*, 13:3–16, 1981.

[12] David Edward Goldberg. *Genetic algorithms in search, optimization, and machine learning.* Addison Wesley, 1989.

[13] R. M. Haralick. Digital step edges from zero crossing of second directiona derivatives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-6(1):58–68, 1984.

[14] S. Kirkpatrick, C. D. Gellat Jr., and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[15] R. Kirsch. Computer determination of the constituent structure of bilogical images. *Computer Biomed. Res.*, 4:315–328, 1971.

[16] D. Marr and E. Hildreth. Theory of edge detection. In R. Kasturi and R. C. Jain, editors, *Computer Vision*, pages 77–107. IEEE, Los Alamitos, Ca, 1991.

[17] J. Marroquin, S. Mitter, and T. Poggio. Probabilistic solution of ill-posed problems in computational vision. *Journal of the American Statistical Association*, 82(397):76–83, march 1987.

[18] W. Martelli. An application of heuristic search methods to edge and contour detection. *Communication of the ACM*, 19(2):73–83, february 1976.

[19] Nenad Mladenovic and Pierre Hansen. Variabel neighborhood search. Les Cahiers du GERAD, 1996.

[20] T. Peli and D. Malah. A study of edge detection algorithms. *CGIP - Computer Graphics Image Processing*, 20:1–21, 1982.

[21] D. M. S. Prewitt. Object enhancement and extraction. In B. S. Lipkin and A. Rosenfeld, editors, *Picture Processing and Psycopictorics*, pages 75–149. Academic Press, New York, 1970.

[22] L. G. Roberts. Machine perception of three-dimensional solids. In J. T. Tippett, editor, *Optical an Electron-Optical Information Processing*, pages 159–197. MIT Press, Cmbridge, Ma, 1965.

[23] A. Rosenfeld, R. A. Hummel, and S. W. Zucker. Scene labeling by relaxation operations. *IEEE Transactions on Systems Man and Cybernetics*, 6:420–433, 1976.

[24] A. Rosenfeld and A. C. Kak. *Digital Picture Processing.* Academic Press, 1982.

[25] M. Sonka, C. J. Wilbricht, S. R. Fleagle, S. K. Tadikonda, M. D. Winniford, and S. M. Collins. Simultaneous detection of both coronary borders. *IEEE Transactions on Medical Image*, 12(3), 1993.

[26] Milan Sonka, Vaclav Hlavac, and Roger Boyle. *Image Processing, Analysis and Machine Vision*, chapter Segmentation, pages 112–191. Chapman & Hall Computing, 1993.

[27] Hing Leong Tan, Saul B. Gelfand, and Edward J. Delp. A comparative cost function approach to edge detection. *IEEE Transactions Systems Man. Cybern.*, 19:1337–1349, 1989.

[28] Hing Leong Tan, Saul B. Gelfand, and Edward J. Delp. A cost minimization approach to edge detection using simulated annealing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(1):3–18, 1991.

[29] V. Torre and T. A. Poggio. On edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(2):147–163, 1986.