Aplicação de Estruturas de Dados Espaciais Eficientes na Estimação de Intensidade de Processos Pontuais

ANDRÉ ÁVILA DA C. SANTOS [1]
RENATO M. ASSUNÇÃO [2]
[1]Departamento de Ciência da Computação – ICEx – UFMG
acsantos@dcc.ufmg.br
[2]Departamento de Estatística – ICEx – UFMG
assuncao@est.ufmg.br
Laboratório de Estatística Espacial (LESTE)
30161-970 – Belo Horizonte – MG, Brasil

Resumo: Este artigo aborda o desenvolvimento de métodos computacionais para a estimação de intensidade de processos pontuais através da geração de mapas de kernel. É mostrado que o impacto dos algoritmos de busca por vizinhos mais próximos é crucial para o bom desempenho desses métodos, sendo assim necessária a escolha de um algoritmo apropriado. É proposto um novo algoritmo baseado numa estrutura de dados bastante simples para a localização desses vizinhos de forma eficiente. Em seguida, as características específicas do problema são levadas em conta, possibilitando pequenas modificações na estrutura capazes de reduzir consideravelmente o número de operações realizadas pelo algoritmo, em comparação à sua versão original. Apesar de lidar apenas com o caso bidimensional, demonstra-se ainda neste artigo que o algoritmo é flexível o suficiente para permitir buscas em regiões multidimensionais sem grande perda de eficiência.

1 Introdução

A estimação de kernel é um método de análise de padrões espaciais de eventos pontuais em $\Re^2$ muito utilizado em diversas áreas de pesquisa, especialmente com a recente proliferação de bancos de dados georeferenciados, conseqüência dos avanços obtidos nos sistemas de informação geográficas (GIS) [Gatrell., 1996]. Seu objetivo é obter uma estimativa suavizada da densidade de eventos por unidade de área, uma propriedade de grande relevância para a análise do comportamento de um processo estocástico espacial.

Do ponto de vista computacional, a principal dificuldade em se fazerem estimações de kernel numa certa região surge da necessidade de se identificarem os eventos próximos aos nós de uma grade fina colocada sobre a área de interesse. Daí surge a importância de se utilizarem algoritmos eficientes para a determinação desses conjuntos de vizinhos.

Neste artigo analisaremos o problema de gerar uma suavização por kernel utilizando métodos computacionais e proporemos um algoritmo eficiente baseado numa estrutura de dados bastante simples para determinação dos conjuntos de vizinhos mais próximos.

2 Estimativa de intensidade de processos pontuais

Um dos principais objetivos ao analisar padrões espaciais de pontos aleatórios está em determinar se os eventos observados exibem algum padrão sistemático ou estão distribuídos aleatoriamente em uma região de estudo contida no plano R [Bailey, 1995]. As características mais relevantes do comportamento de um processo espacial estocástico podem ser definidas em termos de suas propriedades de primeira e segunda ordem, sendo que as características de primeira ordem descrevem a forma como o valor esperado (média ou mediana) do processo varia através do espaço, e as de segunda ordem descrevem a covariância (ou correlação) entre valores do processo em diferentes regiões do espaço [Gatrell, 1996].

Aqui estaremos interessados nas propriedades de primeira ordem, descritas em termos da *intensidade* $\lambda(s)$ do processo, que é o número esperado de eventos por unidade de área no ponto $s=(s_x,s_y)$ [Diggle, 1983]. Ela é definida como:

$$\lambda(s) = \lim_{ds \to 0} \left\{ \frac{E(N(\mathbf{ds}))}{ds} \right\}$$

onde **ds** é uma pequena região em torno do ponto s, ds é a área dessa região, $N(\mathbf{ds})$ é o número aleatório de eventos em **ds** e $E(N(\mathbf{ds}))$ é o seu valor esperado.

A estimação de kernel é um método originalmente desenvolvido para obter uma estimativa suavizada de uma densidade de probabilidade univariada ou multivariada a partir de uma amostra de dados observados [Devroye, 1999]. Estimar a intensidade de um padrão espacial de pontos é muito

semelhante a estimar a densidade de probabilidade bivariada, e a estimação de kernel bivariada pode ser facilmente adaptada para dar uma estimativa da intensidade [Diggle, 1988].

O método pode ser sucintamente descrito da seguinte forma. Suponha que *s* representa uma localização qualquer numa região *R* do plano e $s_1,...,s_n$ são as localizações bi-dimensionais dos *n* eventos observados. Observe que *s* é uma posição arbitrária da região que, coincidentemente, poderá ser a localização de um dos *n* eventos mas isto será muito raro. A intensidade $\lambda(s)$, é estimada por

$$\hat{\lambda}_\tau(s) = \frac{1}{\delta(s)} \sum_{i=1}^{n} \frac{1}{\tau^2} k\left(\frac{s - s_i}{\tau}\right)$$

Aqui, *k()* é uma função densidade de probabilidade apropriadamente escolhida, conhecida como *kernel*, que é simétrica em relação à origem. O parâmetro $\tau > 0$ é a largura de banda (*bandwidth*) e determina a quantidade de suavização – essencialmente é proporcional ao raio de um círculo centrado em *s* dentro do qual pontos s_i contribuirão significativamente para $\lambda(s)$.

A função kernel pode ser qualquer uma, escolhida de acordo com as propriedades necessárias, descritas acima. Consideraremos aqui uma das escolhas mais populares, o kernel quártico, definido como:

$$k(u) = \frac{3}{\pi}(1 - u^T u)^2, \; se \; u^T u \le 1$$

onde $u=(u_x,u_y) \in R$. Caso o ponto esteja fora do círculo, ou seja, $u^T u > 1$, então $k(u) = 0$.

Assim, obteremos a seguinte expressão para $\lambda_\tau(s)$:

$$\hat{\lambda}_\tau(s) = \sum_{h_i \le \tau} \frac{3}{\pi\tau^2}\left(1 - \frac{h_i^2}{\tau^2}\right)^2 \qquad (1)$$

onde h_i é a distância entre o ponto *s* e a localização s_i do evento observado, e a soma é feita apenas para valores de h_i que não excedam τ. O estimador em (1) pode ser interpretado como a soma de contribuições dadas por cada um dos *n* eventos aleatórios observados em *R* para a estimação do número de eventos por unidade de área em torno de uma posição arbitrária $s = (s_x, s_y)$. A contribuição do evento s_i é uma função decrescente de sua distância h_i até a posição *s* de interesse. Caso esta distância seja suficientemente grande ($h_i > \tau$), o ponto s_i contribui com zero em (1).

3 Métodos computacionais

Partindo diretamente da seção anterior, podemos obter um algoritmo geral para a suavização por kernel dada uma região *R* bidimensional, uma largura de banda *r* e um conjunto de eventos compostos por pontos dessa nessa região. O objetivo é estimar a função intensidade em (1) nos pontos *s* correspondentes aos nós de uma grade fina colocada sobre a região de estudo. A grade será representada como uma matriz *G* cujas células correspondem aos pontos *s*. Os valores da estimativa (1) serão armazenados na matriz *K*. O algoritmo é descrito na Seção 8.1.

A análise de complexidade desse algoritmo pode ser feita da seguinte forma: seja *m* o tamanho da matriz G e *n* o número de elementos em P. Podemos determinar, então, que a complexidade desse algoritmo será $O(m^2)O(f(n))$, onde $O(f(n))$ é a ordem de complexidade do algoritmo de busca por vizinhos adotado.

Fica claro pela própria definição do kernel que a dependência quadrática em *m* não pode ser reduzida. Felizmente, na maioria das aplicações reais, *m* não será um número muito grande (possivelmente da ordem 10^2), de forma que essa dependência não é muito debilitante. O que determinará, então, a eficiência de um método computacional para a suavização por kernel será o algoritmo utilizado para encontrar os vizinhos de cada nó da grade.

4 Algoritmos de busca por vizinhos mais próximos

Como mostramos anteriormente, um algoritmo eficiente para a suavização por kernel deve estar baseado num método eficiente para computar os vizinhos mais próximos de cada nó da grade. Este é um problema típico em várias áreas da ciência, o que motivou um extenso estudo de algoritmos para a sua resolução nas mais diversas condições, de forma que o algoritmo a ser utilizado dependerá das características do problema específico a ser resolvido.

Para o problema do kernel, estamos particularmente interessados em algoritmos capazes de realizar *range search* (busca em distância). Ou seja, dado um ponto *s*, encontrar todos os pontos *p* num conjunto P de eventos observados tais que a distância $d(p,s) \le r$, onde r é o raio de um círculo centrado em *s*. Além disso, o algoritmo deve ser adequado para conjuntos estáticos de pontos, ou seja, conhecemos todos os pontos *a priori*. Apesar de nosso foco neste artigo ser a suavização por kernel bivariada, buscaremos um algoritmo também flexível o suficiente para permitir buscas em espaços multidimensionais (kernel multivariado), visando adequação à generalização do método.

Há uma imensa literatura na área sobre algoritmos de busca por vizinhos e suas variantes. Nene e Nayar [Nene, 1997] dividiram os algoritmos existentes em cinco classes: a) Busca exaustiva, b) hashing e indexação, c) particionamento estático do

espaço, d) particionamento dinâmico do espaço, e e) algoritmos aleatorizados**.** Faremos uma rápida discussão de cada uma dessas classes:

a) Busca exaustiva: Intuitivamente, o algoritmo de busca por vizinhos mais simples possível é o de busca exaustiva, no qual calculamos d(*p*,*s*) para cada um dos *n* elementos de P, encontrando assim os que satisfazem a condição de vizinhança. Este algoritmo é claramente ineficiente, e tem complexidade O(*dn*), onde *d* é o número de dimensões do espaço de busca. No caso bidimensional, $d = 2$.

b) Hashing e indexação: Estes são os métodos mais rápidos, pois permitem buscas em tempo constante, mas exigem enormes quantidades de espaço para o armazenamento de suas estruturas, que crescem exponencialmente com a dimensão *d*. Normalmente usam-se técnicas híbridas de hashing e indexação para reduzir o problema da dimensionalidade, mas isso pode acarretar falhas na detecção de vizinhos mais próximos. Essa técnica se mostra eficiente somente quando o ponto de referência (cujos vizinhos queremos encontrar) coincide exatamente com um dos eventos observados. Este não é o caso da estimativa de kernel.

c) Particionamento estático do espaço: Os métodos mais comuns para se resolver de forma eficiente o problema da busca em distância são os baseados nessa abordagem, que tem como objetivo particionar o espaço de forma a tornar as buscas por vizinhos mais eficientes. As estruturas mais usadas são as árvores, em especial a kd-tree [Bentley, 1975], que é uma generalização para *k* dimensões da árvore binária de pesquisa. Várias outras árvores de particionamento são citadas em [Sample, 1999]. As árvores devem ser as estruturas preferidas no caso de conjuntos dinâmicos de pontos, mas seu uso para conjuntos de dados estáticos não se justifica, uma vez que suas características mais interessantes (capacidade de manter um bom desempenho mesmo após inserções ou retiradas múltiplas) seriam desperdiçadas. Elas permitem, teoricamente, tempos de busca $O(\log_2 n)$ e tempo de construção $O(n\log_2 n)$, mas Chanzy, Devroye e Zamora-Cura [Chanzy, 2001] mostraram que, na prática, os tempos de busca para a kd-tree crescem polinomialmente com *n*. É conhecida também a degradação dessa estrutura em espaços com muitas dimensões, pois ela mantém relação de dependência exponencial com a dimensionalidade do espaço de busca, como mostrado em [Sample, 1999], [Chavez, 1999] e [Nene, 1997].

Outra técnica de particionamento bastante usada divide o espaço de busca em polígonos de Voronoi, criando assim um diagrama de Voronoi. Um polígono de Voronoi é uma construção geométrica obtida pela interseção de bissetrizes perpendiculares de pontos adjacentes. Apesar de oferecerem desempenho logarítmico em espaços de busca 2-D, permitindo que o vizinhos mais próximo a um ponto sejam encontrados em $O(\log_2 n)$ operações, os custos de construção $O(n\log_2 n)$ e armazenamento $O(n^2)$ da estrutura crescem a um fator de, pelo menos, 2^d [Tsaparas, 1999], tornando-a inútil para busca em espaços de mais de duas dimensões. Ainda que levemos em conta apenas o caso 2-D, Chavez et al [Chavez, 1999] apontam que os algoritmos de busca em distância baseados em polígonos de Voronoi dependem da identificação de agrupamentos (*clusters*) de elementos próximos e argumentam que mesmo com o uso de bons algoritmos para esse fim não é claro que essa técnica se refletirá em bons algoritmos para busca em distância, que é exatamente o tipo de busca utilizado na estimação por kernel.

d) Particionamento dinâmico do espaço: Essa abordagem consiste em particionar o espaço no momento da busca, baseado no ponto de referência. São métodos que oferecem excelente desempenho em espaços multidimensionais, em geral mantendo relações de dependência polinomiais em *d*, mas mostram uma dependência crítica em *r*, que é o raio da busca, de forma que manter *r* pequeno é suficiente para se obterem ótimos tempos de execução. Note que o termo "dinâmico" aqui se aplica ao método de particionamento do espaço de busca. Veremos que essas estruturas são mais eficazes quando aplicadas a conjuntos estáticos de pontos, ou seja, quando todos os pontos são conhecidos *a priori*.

e) Algoritmos aleatorizados: Várias das técnicas baseadas nas abordagens acima usam a aleatorização para diminuir ou eliminar a dependência em *d*, obtendo resultados diversos.

Diversas técnicas baseadas nas abordagens acima foram propostas, e análises detalhadas de várias delas podem ser encontradas em [Tsaparas, 1999] e [Chavez, 1999].

Para determinar uma técnica que seja adequada para o problema do kernel, podemos imaginá-lo como uma "janela" circular de raio *r*, que se desloca através dos pontos da grade fixa sobre a região de estudo. Uma abordagem semelhante em termos de estrutura de dados para busca por vizinhos é a de particionamento dinâmico, na qual se cria uma janela em torno do ponto de referência, e a busca é feita apenas naquela pequena região. Existem diversas estruturas que utilizam essa abordagem para encontrar vizinhos, e utilizaremos aquela proposta por Nene e Nayar [Nene, 1997], introduzindo modificações simples mas efetivas para adequá-la ao problema do kernel, de forma a aumentar sua eficiência.

Veremos que, ao utilizar a abordagem das "janelas" de busca, especificamente com a estrutura escolhida, poderemos utilizar a própria definição do problema do kernel para acelerar as buscas por vizinhos, tornando o algoritmo global mais eficiente.

5 Um novo algoritmo

Mostraremos resumidamente o princípio de funcionamento da estrutura de dados utilizada. Consideraremos o caso particular em que d=2, mas a generalização para qualquer outro valor de d é bastante simples.

Dado um espaço de busca R, um conjunto de eventos observados P, um ponto de referência $g=(g_x, g_y)$, e um raio r, queremos determinar um pequeno subconjunto do espaço que contém todos os elementos de P tais que $d(p,g) \leq r$. Claramente, um candidato razoável é o quadrado de lado $2r$, circunscrito ao círculo de raio r centrado em g. Essa estrutura de dados permite que identifiquemos todos os pontos de P dentro desse quadrado em tempo logarítmico. Depois disso, basta varrermos estes pontos (um pequeno subconjunto de P) para determinar aqueles que estão realmente dentro do círculo centrado em g.

A montagem da estrutura proposta por Nene e Nayar [Nene, 1997] consiste em ordenar independentemente cada um dos eixos de coordenadas de P, o que pode ser feito em $O(n\log_2 n)$ utilizando o algoritmo Heapsort [Aho, 1974], e construir mapas dos vetores ordenados para os vetores originais, de forma que possamos obter o índice de um elemento no vetor ordenado através de seu índice no vetor original em tempo constante. Essa indexação é chamada *forward mapping*. O contrário também acontece, através do *backward mapping*. Portanto, o custo de construção da estrutura é $O(n\log_2 n)$, mas a construção só precisa ser feita uma vez para o kernel, pelo fato de todos os pontos já serem conhecidos *a priori*. O alto custo de construção é justamente a razão pela qual essa estrutura não é eficiente se usada para conjuntos dinâmicos de pontos, pois nesse caso deveriam ser feitas múltiplas reconstruções.

Para determinar o quadrado onde será feita a busca exaustiva, são realizadas duas buscas binárias para cada dimensão. O objetivo dessas buscas é localizar o ponto mínimo e máximo naquela dimensão entre os quais estarão, seguramente, os vizinhos do ponto de referência. Em outras palavras, essas buscas determinam duas partições da matriz original. A intercessão dessas partições é o quadrado de busca. O *forward mapping* permite que essa intercessão seja calculada fazendo apenas comparações de inteiros, uma operação muito eficiente.

No algoritmo (Seção 8.2), o vetor X é chamado de *lista de candidatos*. O passo (3) mostrado é redundante, uma vez que obtida a lista de candidatos, podemos obter o vetor V apenas fazendo comparações inteiras na própria lista, através do *forward mapping*, como apontado anteriormente. Na prática, o passo (4) é executado diretamente, e é chamado *ajuste da lista de candidatos*. Podemos chamar o vetor V *de lista ajustada de candidatos*. Nessa lista será feita a busca exaustiva pelos pontos dentro do círculo de raio r.

Agora podemos determinar a complexidade do algoritmo de kernel utilizando essa estrutura para encontrar os vizinhos mais próximos. Na seção 3 mostramos que a complexidade do algoritmo de kernel seria $O(k^2)O(f(n))$, onde $O(f(n))$ seria a ordem de complexidade do algoritmo de busca por vizinhos adotado.

Intuitivamente, com uma análise bem superficial, é fácil notar que o custo da busca nessa estrutura é $O(n)$. A princípio esse pode parecer um resultado desanimador, mas não é, pois na verdade a notação O esconde a constante que multiplica n, que neste caso é o que determina um desempenho superior.

Nene e Nayar [Nene, 1997] mostram que o custo de encontrar os vizinhos de g numa das dimensões de busca é nP_i, onde P_i é a probabilidade de que um elemento qualquer do conjunto de pontos originais estar a uma distância r de g na dimensão i. É fácil mostrar que, no caso geral, o número médio N_b de elementos na lista ajustada de candidatos após a iteração b (b-ésima dimensão, incluindo a primeira) é dado por:

$$N_b = n\prod_{i=1}^{b} P_i$$

Outro argumento que leva à mesma conclusão é que sua complexidade também depende de r, que normalmente é uma fração suficientemente pequena da região original. Cardinal [Cardinal, 1999] mostra que o custo do algoritmo para pontos distribuídos uniformemente num quadrado de busca unitário é, em termos de n e r:

$$O\left(n\left(r+\frac{1}{1-r}\right)\right)$$

Chavez et al. [Chavez, 1999] mostram ainda que, na prática, o custo com comparações de distâncias entre pontos nessa estrutura se aproxima mais de $O(\log_2 n)$ que de $O(n)$.

Além disso, temos como argumento favorável a essa estrutura a motivação principal dos seus autores, que foi manter o custo linear em d, resolvendo assim o problema da dependência exponencial apresentada pelas outras estruturas que citamos, de forma a possibilitar buscas eficientes em espaços multidimensionais.

Na próxima seção, estudaremos formas de utilizar a própria estrutura do problema do kernel para tornar o algoritmo de busca por vizinhos mais eficiente.

6 Adequando a estrutura de dados ao problema do kernel

Na seção 4 apontamos que o principal motivo de termos escolhido uma estrutura de particionamento dinâmico do espaço, em vez de utilizar uma das clássicas estruturas de particionamento estático (kd-trees, Voronoi, etc.) era a forma similar como a estrutura de dados e o kernel tratam o ponto de referência, além do fato de todos os pontos serem conhecidos *a priori*.

No kernel, os pontos de referência são determinados pela grade colocada sobre a região de estudo. É imediata, então, a aplicação do algoritmo de busca por vizinhos: em cada ponto da grade, determine a janela de lado 2*r* centrada no ponto de referência e acumule o kernel de cada um dos pontos contribuintes para chegar ao valor da intensidade naquele ponto da grade. Em seguida, passe para o próximo ponto da grade, até que toda a grade tenha sido percorrida.

Estimaremos agora uma função de complexidade pouco rigorosa, apenas para efeito de comparação com as melhorias que proporemos a seguir. Consideraremos todas as buscas exaustivas feitas pelo algoritmo de busca por vizinhos como realizando n operações (mesmo sabendo que, na prática, as buscas exaustivas são feitas em subconjuntos de tamanho menor que n), e todas as buscas binárias como realizando $\log_2 n$ operações. Estudaremos o caso do kernel bivariado (d=2).

O procedimento de busca por vizinhos pode ser visto como três etapas diferentes. A primeira consiste em selecionar os pontos na faixa vertical entre x_{min} e x_{max}, onde $x_{min} = x–r$ e $x_{max} = x+r$. Essa operação tem custo $n+2\log_2 n$. A segunda consiste em selecionar os pontos na faixa horizontal entre y_{min} e y_{max}, calculados da mesma forma que x_{min} e x_{max}, e fazer a intercessão das faixas, localizando os pontos dentro do quadrado centrado em *(x,y)*. O custo é o mesmo da etapa anterior. Por último, percorremos os pontos do quadrado para descobrir quais estão dentro do círculo de raio *r* centrado em *(x,y)*. O custo é *n*. Então, a busca por vizinhos tem custo total $3n+4\log_2 n$.

Assim, podemos considerar a função de complexidade *f(n,k)* para a busca por vizinhos em todos os pontos da grade, no algoritmo de kernel:

$$f(n,k) = k^2(3n + 4\log_2 n)$$

onde *k* é o tamanho da grade, conforme mostrado na seção 3.

Nosso método para melhorar o desempenho se baseia no fato de que os dois eixos foram ordenados independentemente. Mostraremos dois esquemas bem simples para tirar proveito disso, e as chamaremos *horizontal store* (armazenamento horizontal) e *vertical store* (armazenamento vertical).

Horizontal store (HS): É imediato perceber que, se percorrermos a grade linha a linha, podemos calcular a lista de candidatos apenas uma vez para todos os pontos naquela linha, ou seja, calculamos a lista de candidatos apenas para o primeiro ponto e a utilizamos em todos os outros pontos daquela linha. Economizaríamos então duas buscas binárias (a localização do máximo e mínimo daquela dimensão) e uma busca exaustiva (a criação da lista de candidatos propriamente dita, a partir do *backward mapping*) para cada ponto da grade, exceto os da primeira coluna.

Vertical store (VS): Pensando agora na outra dimensão, podemos adotar abordagem semelhante, mas que exige a criação de um novo vetor índice. Para cada coluna, armazenaremos o valor do máximo e mínimo daquela dimensão, naquela coluna, num vetor indexado com o número da coluna. Depois do kernel visitar um ponto de uma coluna já visitada, os valores do máximo e mínimo serão obtidos em O(1). Economizaremos, assim, duas buscas binárias em todos os pontos da grade, exceto os da primeira linha.

O novo algoritmo de busca por vizinhos é detalhado na Seção 8.3.

Podemos calcular agora a nova complexidade, levando em conta esse esquema bem simples de armazenar a informação já obtida. O primeiro ponto da grade executará todas as $3n+4\log_2 n$ operações. Os pontos da primeira linha (exceto o primeiro) executarão $2n+2\log_2 n$ operações (devido ao *HS*), e os da primeira coluna (exceto o primeiro) executarão $3n+2\log_2 n$ operações (devido ao *VS*). Todos os outros pontos tirarão proveito tanto do *HS* como do *VS*, e executarão apenas $2n$ operações. Assim, podemos denotar *g(n,k)* a complexidade do algoritmo melhorado pela estratégia descrita:

$$g(n,k) = k\left[n(2k+1) + 4\log_2 n\right]$$

Para calcularmos a economia de operações conseguida com esse método, podemos definir uma função *h(n,k) = f(n,k)/g(n.k)*, obtendo, assim:

$$h(n,k) = \frac{k(3n + 4\log_2 n)}{n(2k+1) + 4\log_2 n} \approx \frac{3}{2}\frac{k}{k+1} \approx 1{,}5$$

para *n* suficientemente grande. Este resultado mostra que as melhorias que introduzimos reduzem o número de operações realizadas pelo algoritmo de busca por vizinhos em quase 50%. O impacto dessas melhorias no tempo de execução pode ser medido empiricamente, como mostrado nos gráficos a seguir.

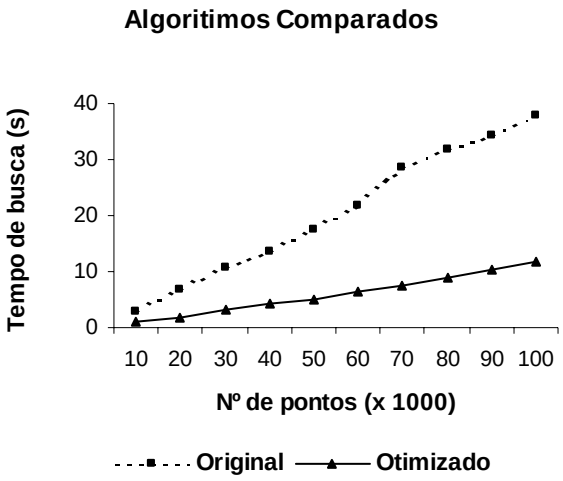

Figura 1: Tempo gasto em buscas por vizinhos pelo algoritmo utilizando a estrutura de dados como originalmente proposta (linha tracejada) e pelo algoritmo utilizando *HS* e *VS* (linha contínua).

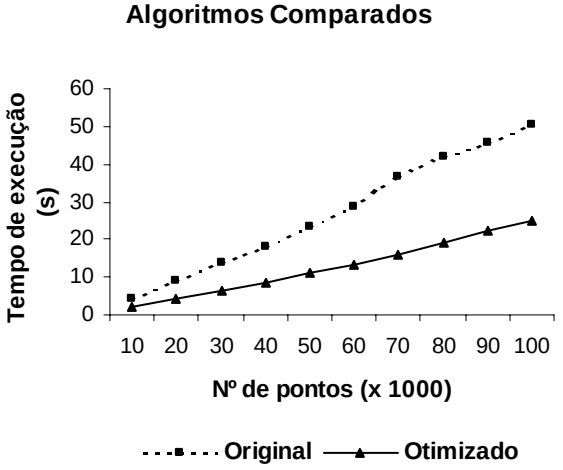

Figura 2: Tempo de execução total de implementações do algoritmo de kernel utilizando as duas versões da estrutura de busca por vizinhos: a estrutura como originalmente proposta (linha tracejada) e a estrutura com *HS* e *VS* (linha contínua) para busca por vizinhos.

No primeiro gráfico (Figura 1), comparamos apenas os tempos gastos pelo algoritmo de kernel em busca por vizinhos. O tempo computado foi o tempo total gasto nessa tarefa, ou seja, a soma dos tempos gastos para encontrar os vizinhos de cada nó da grade colocada sobre a região de estudo. Foram gerados conjuntos de pontos aleatórios, uniformemente distribuídos sobre uma região quadrada. Não foi computado tempo de montagem da estrutura, tempo de E/S nem tempo de cálculo da intensidade em cada ponto da grade. Notamos que o algoritmo de busca otimizado pelos métodos *HS* e *VS* executou de 3 a 4 vezes mais rapidamente que o algoritmo usando a estrutura sem nenhuma modificação.

Para verificar o impacto no algoritmo de kernel como um todo, utilizamos os mesmos conjuntos de dados do teste anterior, mas agora levando em conta o tempo de execução total (Figura 2). Vemos, assim, que o algoritmo otimizado de busca por vizinhos permitiu que o programa do kernel fosse executado em pouco menos de metade do tempo gasto com a estrutura original, para os conjuntos de dados testados.

7 Conclusão

O desenvolvimento de ferramentas computacionais para realizar suavização de padrões espaciais de pontos por kernel passa pela escolha de um algoritmo eficiente de busca por vizinhos mais próximos (*range search*). A escolha de um algoritmo apropriado para executar essa tarefa fundamental pode aumentar sensivelmente o desempenho dessas ferramentas, permitindo que sejam feitas análises mais detalhadas em regiões mais extensas, sem exigir muito tempo de execução.

A estrutura de dados utilizada nesse artigo é de fácil implementação e muito eficiente na prática. As melhorias feitas visando adequá-la ao problema específico do kernel são bastante simples e mostraram uma melhora de desempenho considerável em relação à estrutura original. Apesar de termos tratado o tempo todo com mapas 2-D (kernel bivariado), a estrutura utilizada é suficientemente flexível para permitir sua implementação para a estimação por kernel multivariado, garantindo que o aumento do número de dimensões tem impacto pequeno na complexidade do algoritmo de busca. É nossa intenção lidar com processos pontuais no espaço e no tempo, o que vai levar a um espaço de busca tridimensional. Neste caso, a eficiência do algoritmo proposto será ainda maior em relação às outras estruturas aqui citadas.

8 Apêndice

8.1 Algoritmo 1: Suavização Por Kernel

Entrada: matriz P de dimensão n x 2 contendo as coordenadas (x,y) dos *n* eventos observados.
valor da largura de banda *r*.
matriz G, uma grade fina a ser colocada sobre a região de estudo.

Saída: matriz K contendo os valores de $\lambda_\tau(G_{ij})$ em cada célula da matriz G.

Obs: Definição de V_g:
Seja *g* uma célula G_{ij} na matriz G. V_g é o conjunto dos eventos *v* vizinhos de *g*. Um ponto *p* do conjunto de eventos P é considerado vizinho de *g* se a distância euclidiana d entre *p* e *g* é menor ou igual ao raio do círculo centrado em *g*, ou seja, $v \in V_g$ se, e somente se, $d(v, g) \leq r$.
Definição de *k(v)*:
k(v) é o valor da função kernel no ponto *v*, como definido em (1), na Seção 2.

Algoritmo:

1. coloque a grade G sobre a região de estudo.
2. para cada nó *g* da grade faça
2.1 obtenha o vetor V_g contendo os vizinhos *v* de g.
2.2 para cada *v* em V_g, faça
2.2.1 calcule *k(v)* e acumule o resultado em K_{ij}.
3. retorne K.

8.2 Algoritmo 2: Busca em distância na estrutura *Nene e Nayar* (*d*=2)

Entrada: ponto de referência $g=(g_x, g_y)$
valor do largura de banda *r*.

Saída: vetor N contendo os índices, no conjunto original de eventos P, dos vizinhos de g.

Obs: chamamos o vetor V de "vetor ajustado de candidatos", ou "lista ajustada de candidatos". É o vetor que contém todos os pontos dentro do quadrado de busca de lado 2*r* encontrado para um dado *g*.

Algoritmo:

1. obtenha o vetor X, tal que $X \leftarrow \{p=(p_x,p_y) \mid p_x \in [g_x - r, g_x + r]\}$
2. aplique o *backward mapping* em X.
3. obtenha o vetor Y, tal que $Y \leftarrow \{p=(p_x,p_y) \mid p_y \in [g_y - r, g_y + r]\}$
4. obtenha o vetor V, tal que $V \leftarrow X \cap Y$
5. para cada *v* em V faça
5.1 se $d(v,g) \leq r$ então $N \leftarrow v$.
6. retorne N

8.3 Algoritmo 3: Busca em distância na estrutura utilizada com *HS* e *VS*

Entrada: ponto de referência $g=(g_x,g_y)$
valor do largura de banda r.

Saída: vetor N contendo os índices, no conjunto original de eventos P, dos vizinhos de g.

Obs: o vetor IVS é o índice extra requerido para implementar o *VS*. IVS[*num.coluna*] contém os índices do ponto máximo e mínimo dos pontos naquela coluna.
o vetor X deve ser mantido entre uma chamada e outra da função.

Algoritmo:

1. se g é o primeiro ponto de uma linha
1.1 obtenha o vetor X, tal que $X \leftarrow \{p=(p_x,p_y) \mid p_x \in [g_x - r, g_x + r]\}$
1.2 aplique o *backward mapping* em X.
2. senão use o vetor X da busca anterior.
3. se g não foi ainda mapeado em IVS.
3.1 obtenha o vetor Y, tal que $Y \leftarrow \{p=(p_x,p_y) \mid p_y \in [g_y - r, g_y + r]\}$
3.2 guarde em IVS[*num.coluna*] os pontos máximo e mínimo para aquela coluna
4. senão busque em IVS[*num.coluna*] os índices já armazenados.
5. obtenha o vetor V, tal que $V \leftarrow X \cap Y$
6. para cada *v* em V faça
6.1 se $d(v,g) \leq r$ então $N \leftarrow v$.
7. retorne N

9 Referências

[Aho, 1974] Aho A. V. Hopcroft J. E. e Ullman J. D. *The Design and Analysis of Computer Algorithms.* Addison-Wesley Publishing Company, 1974.

[Bailey, 1995] Bailey T. C. e Gatrell A. C. *Interactive Spatial Data Analysis*. Longman Scientific & Technical, England, 1995.

[Bentley, 1975] Bentley J. L. "Multidimensional binary search trees used for associative searching". *Communications of the ACM* vol. 18, no. 9, 509-517, 1975.

[Cardinal, 1999] Cardinal J. "Faster fractal image coding using similarity search in a KL-transformed feature space". *Fractals : Theory and Applications in Engineering*, 293-306, Springer, 1999.

[Chanzy, 2001] Chanzy P. Devroye L. e Zamora-Cura C. "Analysis of range search for random kd-trees". *Acta Informatica* (37):355-383, 2001.

[Chavez, 1999] Chavez E. Navarro G. Baeza-Yates R. e Marroquín J. L. "Proximity searching in metric spaces". Technical Report TR/DCC-99-3, Dept. of Computer Science, Univ. of Chile, 1999.

[Devroye, 1999] Devroye L. e Krzyzak A. "On the Hilbert kernel density estimate". *Statistics and Probability Letters*, (44):299-308, 1999.

[Diggle, 1983] Diggle P. J. *Statistical Analysis of Spatial Point Patterns.* Academic Press, London, 1983.

[Diggle, 1988] Diggle P. J. e Marron J. S. "Equivalence of smoothing parameter selectors in density and intensity estimation". *Journal of the American Statistical Association* (83):793-800, 1988.

[Gatrell, 1996] Gatrell A. C. Bailey T. C. Diggle P. J. e Rowlingson B. S. "Spatial point pattern analysis and its application in geographical epidemiology". *Transactions, Institute of British Geographers* (21):256-274.

[Nene, 1997] Nene S. A. e Nayar S. K. "A simple algorithm for nearest neighbor search in high dimensions". *IEEE Transactions on Pattern Analysis and Machine* Intelligence, vol. 19 no. 9, 989-1003, 1997.

[Sample, 1999] Sample N. Haines M. Arnold M. e Purcell T. "Optimizing search strategies in k-d trees" *5th WSES/IEEE World Multiconference On Circuits, Systems, Communications & Computers (CSCC 2001)*, 1999.

[Tsaparas, 1999] Tsaparas P. "Nearest neighbor search in multidimensional spaces". Qualifying Depth Oral Report, Technical Report 319-02, Department of Computer Science, University of Toronto, 1999.