TRABALHO PRÁTICO 0:
Análise de monte carlo

Thompson Moreira Filgueiras

[1]Departamento de Ciência da Computação – Universidade Federal de Minas Gerais (UFMG)

thom@dcc.ufmg.br

Resumo. *Este relatório descreve a simulação do agrupamento de pontos atravéz da análise de monte carlo utilizando o score denominado Hopkins Statistic, utilizado para apontar quando um determinado conjunto de dados é provavelmente agrupavel ou nao. O programa é avalia também a diferenca entre o tempo gasto pelo processador com uma tarefa, e o tempo decorrido no relógio. E por fim é feita uma outra avaliação, desta vez estatística, sobre o conjunto de dados avaliado no programa, com a construção de um gráfico que mostra a quantidade de pontos que são ou não agrupáveis.*

1. INTRODUÇÃO

Neste trabalho foi implementado um simulador de agrupamento de dados. Os principais objetivos do trabalho são o exercício da linguagem C, a discussão sobre análise de problemas complexos e sua solução atravéz da análise estatística utilizando a técnica de monte carlo.

O problema específico a ser estudado neste trabalho é a implementação de um simulador de agrupamento de dados que consiste em analizar se um grupo de pontos em coordenadas cartezianas N-Dimensional é ou não agrupável. Uma amostra S é retirada do conjunto de pontos P e uma outra amostra N é gerada pelo programa. Após a retirada das amostras uma rotina avalia qual a menor distancia entre um ponto dos pontos P e outro da amostra N, seja U = somatorio de todas as menores distancias entre P e N e W = somatorio das menores distancias entre todos os pontos da amostra S, a estatistica E avaliada pelo programa: E = U/(U+W), Sendo U e W números necessariamente positivos (pois são provenientes do calculo da distância euclidiana entre dois pontos). 'E' será um número estritamente positivo e no intervalo entre 0 e 1. Se E ¡ 0.5 a análise de monte carlo garante que P e N não são agrupaveis, para E ¿= 0.5 a amostra é possívelmente agrupável, não havendo no entanto garantia de que sejam realmente agrupaveis, e por fim para E ¿0.75 é bem possível que a amostra seja agrupável (não havendo certeza de que isto é verdade).

Um outro problema abordado é a diferença entre o tempo de relógio e o tempo de execussao do programa. Em sistemas operacionais multi-tarefa, aparentemente, várias tarefas são executadas ao mesmo tempo, no entanto esta afirmação é falsa. O que ocorre é que as tarefas são atendidas uma de cada vez, mas em curtos intervalos de tempo da ordem de milisegundos, isto dá impressão ao usuário que muitas tarefas estão em execussão. Por este motivo o tempo de execussão do programa é menor do que o tempo decorrido no relógio (tempo real), pois durante sua execussão, outras tarefas podem ter sido executadas, o que comprometeu o tempo do programa. Para mais informações sobre amostragem: [Amostragem]. Uma explicação mais detalhada sobre análise de monte carlo pode ser encontrada em [MonteCarlo]

O restante deste relatório é organizado da seguinte forma. A Seção 2 discute alguns temas relacionados ao trabalho. A Seção 3 descreve o simulador e a solução proposta. A Seção 4 trata de detalhes específicos da implementação do trabalho. A Seção 5 contém a avaliação experimental do tempo decorrido na execussão e ainda da quantidade de pontos agrupaveis nos testes. A Seção 6 conclui o trabalho.

2. REFERÊNCIAS RELACIONADAS

Podemos dividir as referências associadas ao problema estudado e à solução proposta dentre os seguintes grupos:

- **Análise de monte carlo** Vário problemas encontrados no mundo real, relacionados a alta quantidade de dados, envolve um estudo estatístico do problema, pois fica impraticavel analisar os dados como um todo, mas é bem mais simples retirar amostragens do problema e analisá-los. A técnica de monte carlo se baseia no fato de que provavelmente vários pequenos pedaços de uma amostra bem grande pode sintetizar o comportamento da amostra como um todo.
- **Linguagem C:** O trabalho foi implementado em linguagem C, amplamente utilizada no desenvolvimento de sistemas de alto desempenho, de tempo real, sistemas operacionais, compiladores, dentre outros. É uma linguagem que permite ao programador realizar otimizações com linguagem assembly, o que dependendo da aplicação faz uma grande diferença no desempenho do aplicativo. O compilador utilizado para a linguagem C, neste programa é o GCC (desenvolvido pelo projeto GNU).
- **Projeto e análise de algoritmos:** Algoritmos são procedimentos computacionais capazes de resolver diversos problemas do mundo real. O problema de analisar o agrupamento de dados estudado neste trabalho é um exemplo de como algoritmos podem ser importantes para analise de dados estatisticos complexos. O estudo de algoritmos é essencial para o desenvolvimento de técnicas mais eficientes e eficazes para a solução desses problemas.

3. SOLUÇÃO PROPOSTA

A solução proposta para a simulacao do agrupamento de pontos utilizou uma estrutura de dados em que cada celula guarda além das coordenadas euclidianas do ponto N-dimensional, a distancia U e a distancia W (distancia entre amostra dos pontos gerados e arquivo; distancia entre amostra dos pontos gerados). Essas estruturas são manipuladas através de um conjunto de algoritmos que descrevem o funcionamento da solução.

O simulador implementado gera várias sequencias de pontos e faz a análise com relação ao arquivo de entrada e retira também várias amostras do arquivo original e gera vários números E = U/ U+W para plotacao de gráfico. A cada iteração do programa é gerada uma amostra e um conjunto de pontos. A estrutura de dados escolhida para guardar os pontos gerados é do tipo fila, pois pelo entendimento do autor do trabalho, utilizar um vetor não se caracteriza alocação dinâmica (uma vez que neste tipo de estrutura não é permitido adições ou remoções de novos pontos) A escolha de uma fila para guardar a estrutura ocorreu pois, é um tipo de estrutura que os pontos são adicionados ao final dela o que permitiu a randomização de elementos por lote dentro do arquivo de entrada. Entenda lote como um conjunto de pontos sequencias nos quais somente um desses números serão

sorteados, desta forma há garantia de que toda a estrutura terá igual chance de ser percorrida: lote = numero de pontos/ numero de amostras, em caso de numero de amostras ¿= numero de pontos (do arquivo de entrada) o lote é considerado = 1 (estrutura inteira)

A seguir, serão descritas as estruturas de dados e algoritmos propostos, assim como a análise de complexidade dos algoritmos:

3.1. Estruturas de dados

3.1.1. fila:

Armazena informações relacionadas à um ponto:

1: Requisicao
Coordenadas (P); Distancia (U); Distancia (W);

3.2. Algoritmos

3.2.1. Calcula Distancia entre dois pontos

Faz o calculo da distancia euclidiana entre dois pontos, esta função é $O(D)$ onde D é o numero de dimensoes dos pontos no arquivo de entrada.

2: calcula-distancia(Ponto1, Ponto2)
distancia = raizQuadrada (fori de 0 a P quadrado(P1[i] - P2[i])) retorna distancia

3.2.2. Cálculo da menor distancia entre os pontos da amostra e gerada

Faz o calculo da distancia euclidiana entre todos os pontos da amostra dois a dois, e calcula também a distância entre um ponto do arquivo e um ponto gerado. A primeira parte da função é $O(N * P)$ onde P é o numero de pontos da amostra e N é o número de pontos no arquivo de entrada. Além disto, a segunda parte da função é $O(M * P)$ onde M é o número de pontos gerados aleatóriamente. Podemos perceber que nesta parte do algorítmo, a primeira parte domina a segunda, pois é razoável imaginar que a quantidade de números gerados será menor do que a quantidade de números no arquivo de entrada. Outro detalhe importante é que quanto maior for o tamanho da amostra, mais perto esta função estará de um custo de operações = $O(N^2)$. Por fim esta função realiza $(N * P)$ chamadas da rotina de cálculo de pontos, por esta razão sua complexidade final é $O(N * P * D)$ onde D é o número de dimensões de um dado ponto.

3.2.3. Calculo da estatistica

Calcula o valor de E = U/U+W, para tal ele varre todos os elementos gerados e todos os elementos da amostra somando os valores distU e distW das estruturas. Logo esta função

3: calcula-UW(fila amostra, fila gerada)
paracada ponto da amostra Calcule a distancia com os outros pontos da amostra Salve a distancia caso ela seja a menor e se for diferente de 0 //se distancia é igual a zero é o mesmo ponto paracada ponto gerado Calcule a distancia com os outros pontos da amostra Salve a distancia caso ela seja a menor e se for diferente de 0 //se distancia é igual a zero é o mesmo ponto

é $O(M + P)$ onde P é o numero de pontos da amostra e M é o número de pontos gerados aleatóriamente

4: Hopkins-Statistic(fila amostra, fila gerada)
paracada ponto da amostra some os valores de W paracada ponto gerado some os valores de U E = U/W + U retorne E

3.2.4. Leitura de arquivo

Realiza a leitura de um arquivo de entrada passado como parametro, retorna uma string contendo a primeira palavra do texto e salva em memoria as palavras restantes. Esta parte do algoritmo ignora alguns caracteres pre determinados, nesta implementação específica ele ignora todos os caracteres que não são números. As palavras restantes são recuperadas atravéz da função proximaPalavra.

5: Hopkins-Statistic(fila amostra, fila gerada)
faz leitura do arquivo salva texto em memoria retorna a primeira palavra

3.2.5. Simulação

Controla a execussão do programa principal. Realiza um loop S vezes, onde S é um parâmetro enviado na entrada. Como realiza as chamadas das funções que executam o programa, esta função é $O(S)$ x ($O(M + P)$ + $O(N * P * D)$). Obviamente o gargalo esta função é $O(N * P * D)$, logo podemos considerar a função como sendo $O(S * N * P * D)$ sendo N o número de pontos do arquivo de entrada, P o número de pontos da amostra e D o número de dimensões de um arquivo

4. IMPLEMENTAÇÃO

4.1. Código

4.1.1. Arquivos .c

- **main.c:** Arquivo principal do programa que implementa o simulador de pontos.
- **file.c:** Define as funções relacionadas à leitura e escrita de arquivos.
- **fila.c:** Define as operações relacionadas à fila e à análise estatistica dos dados.

4.1.2. Arquivos .h

- **file.h:** Define os cabeçalhos das funções relacionadas à leitura e escrita de arquivos.
- **fila.h:** Define as estruturas de dados e cabeçalhos de funções relacionadas a manipulação de um TAD fila implementada

4.2. Compilação

O programa deve ser compilado através do compilador GCC através de um makefile, ou pelo script na pasta scripts ou ainda do seguinte comando:

```
gcc main.c file.c fila.c -o aplicativo
```

4.3. Execução

A execução do programa tem como parâmetros:

- Um arquivo de entrada.
- O tamanho de cada amostra.
- O número de testes a serem realizados.
- Um arquivo de saída.

O comando para a execução do programa é da forma:

```
./aplicativo -i<arquivo de entrada> -s <número de testes> -n <tamanho da amostra> -o
```

4.3.1. Formato da entrada

O formato da entrada possui dois numeros na primeira linha, a saber: numero de pontos, numero de dimensoes E existirao outras (numero de pontos) linhas no arquivo, uma para cada ponto. Os valores de cada ponto devem ficar entre 0 e 1. A leitura dos argumentos foi feita atravéz da função getOpt [GetOpt]

```
4 2
0.00 0.21
1.00 0.45
0.54 0.89
0.17 0.64
```

4.3.2. Formato da saída

A saída do programa exibe as estatisticas feitas durante a execusaao, a saber: linha 1: valor minimo linha 2: valor maximo linha 3: valor medio linha 4: desvio padrao exemplo de saida:

```
0.638654
0.644165
0.641232
0.010553
```

Uma abordagem mais detalhada sobre desvio padrão pode ser encontrada em: [DesvioPadrao]

5. AVALIAÇÃO EXPERIMENTAL

Nesta parte foram feitas duas avaliações: A primeira é a comparação entre tempo de execussao e tempo decorrido no relógio, a segunda comparação é entre a variação dos valores no cálculo da estatística.

5.0.3. Variação do tempo

Em todos os testes execultados ocorreu uma variação entre o tempo de execussão e o tempo de relógio, salvo é claro quando a entrada dos arquivos era muito pequena, pois neste caso o instrumento de medida não era capaz de avaliar o tempo decorrido. Mas, para as entradas na pasta 'entradas' este tempo é medido. Exemplo: (execussao do programa com entrada: normal.data S = 100 N = 100) O arquivo timeAnalisis possui os tempos de execussão de todos os testes rodados. Note que neste caso há uma discrepância entre o que foi dito aqui em cima e o resultado no arquivo. Em alguns casos, o tempo de relógio registrado foi inferior ao tempo execultado pela máquina. Na realidade o que aconteceu é que a precisão da função gerrusages é double e da função getdateoftime é long. Nos casos em que o tempo de relogio é menor do que o tempo de máquina ocorreram somente por erro de precisão do long em relaçao ao double. Pois como já explicado este tempo não pode ser menor do que o tempo de maquina.

Algumas entradas encontradas no arquivo:

#Tempo execussao(segundos)	Tempo Relogio(segundos)
515.696228	517.000000
515.700229	518.000000
774.004371	776.000000
772.284264	772.000000
771.692227	773.000000

Tabela 1. Tempo de execussão e tempo de relógio para uma das execussões do programa

5.0.4. Análise estatística

A análise estatística consiste em analisar para um dado conjunto de dados aqueles que provavelmente se agrupam ou não com os outros pontos gerados. Nesta parte do programa foram realizados vários testes com três amostras diferentes de arquivo, três valores de S e N diferentes. Todas as combinações possíveis entre S e N foram testadas, para que a avaliação fosse justa com as várias baterias de testes, o parametro para a função SRAND foi o mesmo em todos eles = 1234. Todos os testes foram feitos em um computador pessoal. Sistema operacional Ubuntu 9.10, processador intel core 2 duo 2gb de memória RAM.

Comparando os arquivos timeAnalisis.txt e as entradas S e N presentes no arquivo scripts/make percebemos que quanto maior o valor de N maior o tempo de execussão, assim como S. Isto é perfeitamente aceitável, pois, Aumentando-se o número de iterações do loop, mais amostras são geradas e mais tempo processando será necessário. Assim

como quanto maior o o tamanho de N (tamanho das amostras) mais pontos serão analizados entre si no gargalo do algoritmo que é $O(N2)$.

Pela análise dos testes gerados podemos concluir que em geral, quanto maio o valor das amostras retiradas e quanto maior o número de iterações mais próximo do resultado esperado a amostra fica. Para este teste em específico, era esperado que o arquivo clustering.data fosse o menos agrupável de todos, pois os pontos contidos neste arquivo possuem uma certa concentração mas em algumas partes, enquanto em outras ele é totalmente esparço. Para o arquivo normal.data o resultado esperado é que ele estivesse na média dos três arquivos pois os valores contidos neste arquivo não estão concentrados, mas também não estão uniformemente distribuidos. Por fim, o arquivo uniform.data era o que esperava-se ser o mais agrupável de todos, uma vez que tanto ele quanto o gerador de números pseudo-aleatórios da linguagem C são constituidos de pontos uniformemente distribuidos. De fato, podemos observar nos arquivos presentes na pasta resultados que o grau de agrupamento dos pontos ocorre em ordem crescente do clustering para o normal e do normal para o uniform.

6. CONCLUSÃO

Foi descrito neste trabalho um simulador de agrupamento de pontos utilizando a tecnica de monte carlo. A tecnica empregada possui uma eficiencia estatistica relativamente grande, uma vez que não necessitamos analizar toda a entrada de dados para descobrir se um conjunto de pontos é possivelmente agrupavel, o que neste caso poupa muito tempo em outras análises de detalhes mais práticos em outras aplicações.

O trabalho ainda discutiu a diferença entre tempo de máquina e tempo de relógio, foi mostrado experimentalmente aquilo que era esperado: o tempo que o programa gastou para ser executado é menor do que o tempo decorrido no relógio.

O trabalho atingiu seus principais objetivos: a prática da linguagem de programação C e o estudo da análise de monte carlo. A parte que mais demandou tempo foi entender no que consistia tal análise, e como implementá-la, pois a implementação em si é fácil para quem já domina a linguagem.

A análise de complexidade da solução foi relativamente simples, uma vez que a estrutura de dados utilizada não foi a mais complexa possivel para utilizar no problema, o que neste caso facilitou.

Algumas melhorias que poderiam ser consideradas neste trabalho são:

- O emprego de uma outra estrutura de dados do tipo arvore, que permite que os calculos das distancias entre os pontos seja calculada de maneira mais rápida.
- Ao gerar um ponto randomicamente já calcular sua distância com os demais pontos do arquivo, assim seria possível economizar memória

Referências

Amostragem. http://www.unp.br/download/mestradoadm/publicacoes/seminarios/amostragem.pdf. web.

DesvioPadrao. http://matematiques.sites.uol.com.br/pereirafreitas/2.2.3varianciadesvio.htm. web.

GetOpt. http://www.gnu.org/s/libc/manual/html_node/Example-of-Getopt.html#Example-of-Getopt. web.

MonteCarlo. http://pt.wikipedia.org/wiki/M%C3%A9todo_de_Monte_Carlo. web.