Copy Models for Data Compression

Mestrado em Engenharia Informática

Universidade de Aveiro

DETI - Aveiro, Portugal

TAI - Teoria Algorítmica da Informação

Guilherme Antunes [103600], Gonçalo Silva [103668], Pedro Rasinhas [103541]

v2024-03-28

Índice

1 Introdução

O volume cada vez maior de dados gerados por várias fontes, desde documentos de texto e imagens, até arquivos de áudio e vídeo, exige métodos eficientes de armazenamento e transmissão. As técnicas de compressão de dados abordam esse desafio representando informações de forma mais compacta, reduzindo os requisitos de armazenamento e os tempos de transmissão, enquanto preservam seu conteúdo de informação essencial.

Na base da compressão de dados está a exploração de padrões e redundâncias inerentes aos mesmos, com o objetivo de representá-los de forma mais concisa e comprimida. Um dos aspectos intrigantes dos dados é sua auto-similaridade inerente, onde certos “pedaços” de dados podem ser vistos como réplicas ou variações de segmentos previamente encontrados. Esta observação forma a base para várias abordagens de compressão de dados, uma das quais nós exploramos neste trabalho prático: o uso de Copy Models.

Os Copy Models operam sob a premissa de que uma fonte de dados gera sequências replicando partes de dados previamente gerados, possivelmente com modificações. Ao prever o próximo símbolo em uma sequência com base em ocorrências passadas, esses modelos exploram as auto-similaridades subjacentes para alcançar a compressão. O princípio subjacente é que uma parte significativa dos dados pode ser reconstruída referenciando padrões previamente encontrados. Imaginemos um documento de texto contendo uma frase como: "the quick brown fox jumps over the lazy dog." Um modelo de cópia poderia reconhecer a repetição de "the" e representá-la referenciando a primeira ocorrência em vez de armazená-la novamente.

Neste contexto e para este trabalho prático, os nossos objetivos são: Em primeiro lugar, compreender os mecanismos internos dos modelos de cópia, isto é: explorar como o modelo identifica e utiliza as similaridades dentro do conjunto de dados. Em segundo lugar, vamos investigar a eficácia prática dos modelos de cópia para compressão de dados. Ao implementar o modelo nós mesmos vamos avaliar o seu desempenho em vários conjuntos de dados. Uma métrica chave para esta avaliação será a taxa de compressão alcançada, que quantifica a redução no tamanho dos dados em comparação com o seu formato original. Além disso, vamos analisar o impacto que diferentes parâmetros do modelo têm na compressão final dos dados.

Em última análise, este trabalho prático serve como uma ajuda para uma compreensão mais profunda desta técnica de compressão de dados.

2 Metodologia

O projeto está dividido em 2 programas, o cpm.cpp e o mutate.cpp. Enquanto o cpm.cpp implementa o objeto principal do projeto, através do Copy Model e do Fallback Model, o mutate.cpp implementa um programa responsável por gerar mutações no conteúdo de um ficheiro tendo por referência uma probabilidade de mutação introduzida via linha de comandos.

Enquanto o Mutate foi desenvolvido apenas para fazer face às exigências feitas no enunciado do projeto, o CPM envolveu trabalho de pesquisa e discussão com outros colegas e professores, pelo que as fontes utilizadas estão referidas nas referências do trabalho.

2.1 Organização

A imagem abaixo demonstra uma amostra da estrutura de ficheiros do nosso projeto. Na pasta ***src*** estão presentes todos os ficheiros relacionados ao código desenvolvido, para o Copy Model, ***cpm.cpp***, e para o Mutator temos ***mutate.cpp***.
Na pasta ***examples*** estão presentes os ficheiros usados nos testes e por último a pasta ***reports*** onde está o relatório que descreve o trabalho desenvolvido. Também está disponível um ficheiro ***README.md*** que descreve como correr o programa.

2.2 Argumentos da Linha de Comandos

Nome	Flag	Descrição	Type
Ficheiro de Input	-f	Ficheiro a ser analisado	String
Tamanho da janela/kmer	-k	Definir tamanho da janela/kmer	Positive Integer
MinTries/Threshold	-t	Número mínimo de tentativas e rácio mínimo entre o número de hits e tentativas	Integer/Double
Número de Copy Models	-n	Definir número de Copy Models a criar	Positive Integer
Alpha	-a	Parametro de “smoothing” para o cálculo de Probabilidades	Double
Tamanho da Janela Fallback	-s	Define o tamanho da janela do model Fallback	Positive Integer
Número de letras para match	-m	Quantidade mínima de letras consecutivas iguais para considerar um match no Copy Model	Positive Integer

3 Copy Model

3.1 Design e Implementação

O Copy Model como já referido trata-se de um modelo de compressão, cujo objetivo é calcular um número total de bits de informação através da identificação e comparação de padrões de repetição intrínsecos aos dados.

O programa começa pela leitura completa do ficheiro referenciado via linha de comandos para memória e imediatamente de seguida computa o alfabeto das letras existentes nesse ficheiro. Após a recolha dos outros parâmetros de inicialização o Copy Model é instanciado uma vez sem qualquer k-mer associado.

A partir deste momento começam a formar-se k-mers de tamanho k, introduzido pelo utilizador, utilizando uma janela deslizante de tamanho k e step 1. Quando a janela se encontra num índice em que não existem caracteres passados suficientes para a preencher, o caractere mais antigo irá preencher os espaços em falta.

CopyModel

Atributos	n	alpha	k	goal	minTries	threshold	alphabetSize	references
Type	unsigned long int	double	int	int	int	double	int	vector<Reference>
Valor Default	Definido no Construtor	Definido no Construtor	Definido no Construtor	Definido no Construtor	Definido no Construtor	Definido no Construtor	Definido no Construtor	vector<Reference>()

Reference

Atributos	kmer	prediction	nHits	nTries	goal	k	alpha	minTries	activated	threshold	alphabetSize
Type	string	char	double	int	int	int	double	int	bool	double	int
Valor Default	Definido no Construtor	Definido no Construtor	0	0	Definido no Construtor	Definido no Construtor	Definido no Construtor	Definido no Construtor	true	Definido no Construtor	Definido no Construtor

Cada Copy Model é composto por várias referências, cujo limite é definido pelo utilizador aquando do executar do programa. Essas referências por sua vez vão estar associadas a uma previsão cada uma e associados a essa previsão vão existir dois contadores, um para os hits e um para as tentativas de previsão que são feitas, ambos inicializados a 0. Sempre que um novo k-mer dá match com algum daqueles que estão armazenados nas referências dos Copy Models tentamos prever qual será o símbolo que o seguirá com base nessas referências ativas, aumentando ambos os contadores em caso de hit ou apenas o das tries em caso de miss.

O conceito de match entre k-mers está feito de maneira que quaisquer k-mers com uma sequência de *m*, introduzido pelo utilizador, caracteres iguais sejam considerados “iguais” apesar de não o serem totalmente.

Quando a relação entre o número de hits e o número de tries é inferior à fração definida no threshold então essa referência é desativada. Quando todas as referências estiverem desativadas, o modelo será reiniciado novamente sem referências. A atribuição de um k-mer e de uma referência é feita com base numa tabela em que sempre que ocorre

um k-mer é armazenado na tabela juntamente com a previsão que este faz para o caractere seguinte, quando um k-mer que já existe nessa tabela for encontrado é adicionada mais uma previsão à lista de previsões do Copy Model e caso o Copy Model esteja desativado então será ativo com o primeiro k-mer a encontrar-se na tabela e com a previsão imediatamente a seguir à última que já terminou o seu ciclo de vida.

3.2 Cálculo de Probabilidades e Previsões

Com base no número de hits e tries o Copy Model permite-nos calcular a probabilidade de o caracter seguinte ser cada uma das letras do alfabeto.

$$P(hit) \approx (Nh + \alpha)/(Nt + 2\alpha),$$

Figura 1 - Fórmula de cálculo da probabilidade de hit

Dado pela fórmula da Fig. 1, em que Nh representa o número de hits observados e Nt representa o número de tries observados até ao momento e α é o parâmetro de suavização, o valor resultante desta operação corresponde à probabilidade de o próximo símbolo ser aquele que está previsto pela referência.

$$P(miss) \approx (1 - P(hit))/(alphabetSize - 1),$$

Figura 2 - Fórmula de cálculo da probabilidade de miss

Em caso de miss a probabilidade é computada com a fórmula da Fig. 2.

Quando estamos perante uma situação de 1 referência por Copy Model a probabilidade é exatamente o valor dado por cada uma das fórmulas anteriores. Em caso de existência de mais de 1 referência por Copy Model então a probabilidade será dada pela média aritmética de todas as probabilidades calculadas por todas as referências ativas.

A probabilidade que um símbolo tem de ocorrer permite-nos calcular a quantidade de informação necessária para o armazenar.

$$Bits \approx -log2(P),$$

Figura 3 - Fórmula de cálculo do número de bits que um símbolo de probabilidade P ocupará aproximadamente

No caso do Copy Model esse cálculo é possível fazer-se pela fórmula da Fig. 3. No entanto, não é assim para todos os modelos.

4 Fallback Model

Fallback Model é um conceito bastante utilizado em alternância com o Copy Model. Enquanto o Copy Model tenta prever o próximo símbolo numa sequência com base em ocorrências passadas, o Fallback Model é usado quando a previsão do Copy Model falha ou quando não existem Copy Models ativos.

4.1 Design e Implementação

Atributos	data	k	size	alphabet	counts
Type	char *	int	int	Alphabet	unordered_map<char, double>
Valor Default	Definido no Construtor	Definido no Construtor	Definido no Construtor	Definido no Construtor	Definido no Construtor

O Fallback Model recebe do utilizador o tamanho da janela sobre a qual irá contabilizar o número de ocorrências de cada elemento do alfabeto. Para uma maior eficiência, o fallback não guarda cada letra presente na janela. Ao invés disso, cada vez que uma letra nova é lida é incrementado o contador de ocorrências dessa letra. Quando o tamanho da janela é superior ao número de caracteres já lidos a soma dos valores de todos os contadores não terá soma igual ao valor introduzido na linha de comandos. Quando finalmente a janela é menor que todo o conteúdo já lido então o Fallback Model começa a decrementar o contador da letra mais antiga que foi contabilizada e que ainda se encontra contabilizada, supondo que o tamanho da janela é 200 e estamos no caractere i então acede ao índice i - 200 e reduzimos o contador correspondente à letra nesse índice.

Como já referido o Fallback Model é chamado apenas quando o Copy Model não funciona, no entanto é atualizado a cada iteração sobre o documento a ser lido para garantir que os contadores estão atualizados.

4.2 Cálculo de Probabilidades e Previsões

$$Bits \approx \sum -log2(p) \times p,$$

Figura 4 - Fórmula de cálculo do número de bits que um símbolo de probabilidade P ocupará aproximadamente

Para estimar a quantidade de bits necessários para codificar cada símbolo fazemos uma distribuição de probabilidades de cada elemento do alfabeto e aplicamos a fórmula da Fig. 4 em que p é a frequência relativa do símbolo em questão e que corresponde ao somatório dos produtos entre o negativo do logaritmo de base 2 de p e o próprio p.

5 Mutate

O programa mutate foi projetado para introduzir mutações aleatórias no conteúdo de um ficheiro. Este processo baseia-se numa probabilidade de mutação especificada pelo utilizador, que determina a probabilidade de um caractere do ficheiro ser substituído por outro caractere, do mesmo alfabeto. O processo de mutação é crucial em várias aplicações, incluindo a testagem da robustez de algoritmos de compressão de dados contra alterações aleatórias nos dados. Ao introduzir mutações, podemos avaliar o quanto um algoritmo de compressão pode manter o seu desempenho ou adaptar-se aos dados alterados.

5.1 Implementação

Argumentos da Linha de Comandos

O programa utiliza a função getopt da biblioteca unistd.h para analisar os argumentos da linha de comando. Esta função permite que o programa aceite opções na forma de -i <inputFileName> -o <outputFileName> -p <probability>, onde:

- i especifica o nome do ficheiro de entrada.
- o especifica o nome do ficheiro de saída.
- p especifica a probabilidade de mutação.

O loop getopt itera através de cada argumento, atribuindo os valores às variáveis apropriadas. Se for encontrada uma opção inválida ou se argumentos obrigatórios estiverem em falta, o programa imprime uma mensagem de utilização e termina o programa.

Validação de Inputs

Após a análise dos argumentos da linha de comandos, o programa realiza várias verificações para garantir que os inputs são válidos:

- Verifica se o ficheiro de entrada existe e pode ser aberto.
- Verifica se o ficheiro de saída pode ser criado.
- Verifica se a probabilidade de mutação está dentro do intervalo de 0 a 1.

Leitura do Ficheiro de Entrada

O programa determina o tamanho do ficheiro de entrada e em seguida aloca a memória necessária e lê todo o conteúdo para ela.

Povoamento do set do Alfabeto

Para facilitar o processo de mutação, o programa cria um set de caracteres únicos (alfabeto) encontrados no ficheiro de entrada iterando através de cada caractere presente no ficheiro e inserindo-o no set cujas propriedades garantem que não existam elementos repetidos.

Atribuir uma seed ao Gerador de Números Aleatórios

Antes de realizar qualquer mutação, o programa define uma seed para o gerador de números aleatórios utilizando a unix timestamp correspondente ao instante da atribuição garantindo que a sequência de números aleatórios gerados durante o processo de mutação seja diferente cada vez que é executado.

Processo de Mutação

O processo de mutação consiste na geração de um número aleatório por cada caractere existente no ficheiro de entrada, caso esse número seja superior à probabilidade de mutação inserida então o caractere em questão irá ser alterado para qualquer um dos elementos do alfabeto, incluindo ele mesmo. Em caso negativo não ocorre nenhuma mutação e avança para o símbolo seguinte.

Escrever os Dados Mutados para o Ficheiro de Saída

Por fim a nova sequência mutada é escrita para o ficheiro de saída requerido inicialmente.

6 Resultados

Para realizar os testes seguintes foi utilizado um computador de processador Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz, 16GB de RAM sobre os quais corre WSL2, Windows 11 e foi considerado o ficheiro chry.txt providenciado no enunciado do projeto.

Para além de comparações entre variações do mesmo parâmetro também foi comparado o tamanho da compressão com outros programas comummente utilizados para comprimir dados.

k	4	8	12	16
Parâmetros				
α	1	1	1	1
Threshold	0.8	0.8	0.8	0.8
Min. Tries	8	8	8	8
Quantidade de Copy Models	1	1	1	1
Tamanho da janela Fallback	200	200	200	200
Número mínimo para match	4	4	4	4
Resultados				
Total Bytes	5327749	5316413	5281577	5292444
Total Info p/S	1.88025	1.87625	1.86396	1.86779
Tempo(s)	5.93511	14.8729	19.8779	23.8125

Tabela 1 – Resultados com Variação de k

α	0.5	1	1.5	2
Parâmetros				
k	12	12	12	12
Min. Tries	8	8	8	8
Threshold	0.8	0.8	0.8	0.8
Quantidade de Copy Models	1	1	1	1
Tamanho da janela Fallback	200	200	200	200
Número mínimo para match	4	4	4	4
Resultados				
Total Bytes	5209419	5281577	5332447	5371297
Total Info p/S	1.83849	1.86396	1.88191	1.89562
Tempo(s)	18.6229	18.2148	18.1244	18.6188

Tabela 2 – Resultados com Variação de α

Min. Tries	0	4	8	12	16
Parâmetros					
k	12	12	12	12	12
α	1	1	1	1	1
Threshold	0.8	0.8	0.8	0.8	0.8
Quantidade de Copy Models	1	1	1	1	1
Tamanho da janela Fallback	200	200	200	200	200
Número mínimo para match	4	4	4	4	4
Resultados					
Total Bytes	5913339	5234033	5281577	5312296	5316779
Total Info p/S	2.08692	1.84718	1.86396	1.8748	1.87638
Tempo(s)	17.7183	18.5921	19.172	22.5525	19.6646

Tabela 3 – Resultados com Variação de Min. Tries

Threshold	0.3	0.5	0.8	1
Parâmetros				
k	12	12	12	12
Min. Tries	8	8	8	8
α	1	1	1	1
Quantidade de Copy Models	1	1	1	1
Tamanho da janela Fallback	200	200	200	200
Número mínimo para match	4	4	4	4
Resultados				
Total Bytes	5376848	5308472	5281577	5281154
Total Info p/S	1.89758	1.87345	1.86396	1.86381
Tempo(s)	21.1528	18.8855	20.0741	20.0407

Tabela 4 – Resultados com Variação de Threshold

Quantidade de Copy Models	1	10	50	100
Parâmetros				
k	12	12	12	12
Min. Tries	8	8	8	8
α	1	1	1	1
Threshold	0.8	0.8	0.8	0.8
Tamanho da janela Fallback	200	200	200	200
Número mínimo para match	4	4	4	4
Resultados				
Total Bytes	5281577	5253948	5201601	5167915
Total Info p/S	1.86396	1.85421	1.83573	1.82384
Tempo(s)	19.6615	36.2229	74.2615	101.295

Tabela 5 – Resultados com Variação de Quantidade de Copy Models

Tamanho da janela Fallback	12	50	200	500
Parâmetros				
k	12	12	12	12
Min. Tries	8	8	8	8
α	1	1	1	1
Threshold	0.8	0.8	0.8	0.8
Quantidade de Copy Models	1	1	1	1
Número mínimo para match	4	4	4	4
Resultados				
Total Bytes	4984938	5261010	5281577	5281577
Total Info p/S	1.75927	1.8567	1.86396	1.86396
Tempo(s)	18.1251	18.2805	18.6321	18.3196

Tabela 6 – Resultados com Variação de Tamanho da janela Fallback

Número mínimo para match	4	7	9	12
Parâmetros				
k	12	12	12	12
Min. Tries	8	8	8	8
α	1	1	1	1
Threshold	0.8	0.8	0.8	0.8
Quantidade de Copy Models	1	1	1	1
Tamanho da janela Fallback	200	200	200	200
Resultados				
Total Bytes	5281577	5328323	5328801	5328764
Total Info p/S	1.86396	1.88046	1.88062	1.88061
Tempo(s)	18.7862	18.4463	18.1021	16.3148

Tabela 7 – Resultados com Variação de Número mínimo para match

Algoritmo	(nenhum)	bzip2	gzip	zstd
Total Bytes	22668225	5692456	6227386	6168131

Tabela 8 – Comparação entre diferentes algoritmos de compressão

7 Conclusão

Apesar de os resultados obtidos nos testes não serem muito dispersos podemos verificar que a melhor taxa de compressão foi obtida para a menor janela de fallback, enquanto a pior foi a que decidiu ignorar o limite mínimo de tentativas para começar a avaliar a referência. Também podemos verificar que com o aumento da quantidade de Copy Models verificamos, como se esperava, um aumento da taxa de compressão. No entanto isto foi a troco dos maiores aumentos de tempo de execução.

Qualquer um dos testes efetuados não reflete necessariamente uma regra, pelo que a conjugação de diferentes valores nos vários parâmetros pode resultar em compressões muito mais eficientes “inesperadamente”, uma vez que depende da “aleatoriedade” dos dados a serem comprimidos.

Apesar de os resultados obtidos pelos compressores mais comuns serem ligeiramente piores que os do Copy Model os mesmos fazem-no bastante mais rápido e não precisam de tantas opções de configuração para os alcançar.

8 Referências

[1] A. J. Pinho, D. Pratas, M. Hosseini e J. M. Silva, “A Reference-Free Lossless Compression Algorithm,” p. 18, 2 Novembro 2019.

[2] A. J. Pinho e D. Pratas, “JARVIS2: a data compressor for large genome sequences,” p. 10, 2023.

[3] A. J. Pinho e D. Pratas, “Copy models for protein sequence compression,” p. 10, 2024.

[4] A. J. Pinho e D. Pratas, “Copy models for data compression,” [Online]. Available: https://elearning.ua.pt/pluginfile.php/336635/mod_resource/content/11/trab1.pdf. [Acedido em Março 2024].