Coloração de Grafos na Resolução de Problemas de Armazenamento Segregado

Rita de Cássia Chagas da Cruz

Instituto Metrópole Digital - IMD / Universidade Federal do Rio Grande do Norte - UFRN

Resumo

O problema de armazenamento segregado envolve questões de segurança e otimização de recursos. Consiste em armazenar elementos incompatíveis utilizando o menor número possível de containers. Propôs-se a encontrar este menor número. O estudo aplicou teoria dos grafos, em particular, a coloração de grafos, para encontrar o número cromático e calcular o número mínimo de containers.

Palavras-chave: teoria dos grafos, coloração de grafos, armazenamento segregado

1. Introdução

A segregação é uma questão frequente quando se trata do manuseio e armazenamento de uma variedade de bens, tanto no contexto de armazenamento local (e.g., um depósito) quanto num contexto de armazenamento de grandes cargas (e.g., um navio cargueiro) [1]. A forma mais simples de segregação é utilizar n compartimentos para n produtos. Posto isto, este trabalho se propõe a utilizar coloração de grafos para resolver problemas de armazenamento segregado de modo eficiente, calculando o número mínimo de compartimentos necessários.

De modo resumido, o problema de coloração de grafos pergunta, dado qualquer grafo, como é possível atribuir "cores" a todos os seus vértices para que (a) nenhum vértice ligado por uma aresta receba a mesma cor e (b) o número de cores diferentes usado é minimizado [2].

2. Descrição do problema real

Bens perigosos são aqueles que são inerentemente perigosos ou passíveis de reação química com outros. O produto de tal reação é geralmente perigoso e medidas de precaução rigorosas devem ser observadas ao manusear e alocar tais materiais. Embora o principal motivo do armazenamento segregado seja por causa desses materiais, uma abstração pode ser feita para armazenamento segregado por qualquer motivo, desde que os elementos em questão possam ser rotulados como compatíveis ou incompatíveis.

Apesar de haver restrições que ajudam na otimização deste armazenamento, como o formato dos containers, de modo que geometricamente ocupem menor área, este trabalho busca uma solução apenas para o problema de quantidade, isto é, a segregação adequada dos elementos utilizando o mínimo de compartimentos. Outro fator é que a incompatibilidade de alguns materiais é por distância, ou seja, tais materiais podem estar juntos num mesmo container desde que separados a uma distância de m metros. Por fins de simplificação, este fator será desconsiderado, atribuindo um caráter binário (compatível ou incompatível) à categorização dos materiais.

3. Modelagem em grafos

Para converter o problema de armazenamento segregado, considere o grafo $G = (V, E)$, onde V é o conjunto de elementos a serem armazenados e $u, v \in V$ são adjacentes se, e somente se, u, v são incompatíveis.

Então, qualquer conjunto de elementos compatíveis corresponde a um conjunto de vértices independentes de G. Assim, um armazenamento seguro corresponde a uma partição de V em subconjuntos independentes de G. A cardinalidade de tal partição mínima de V é o número necessário de containers. A cardinalidade mínima é chamada de número cromático do grafo G [3]. O problema de armazenamento segregado é resolvido calculando o número cromático.

4. Estado-da-arte

4.1. Revisão bibliográfica do problema real e do modelo em grafos

O problema real escopo deste trabalho é conhecido na literatura como problema generalizado de armazenamento segregado (GSSP, do inglês "generalized segregated storage problem"). O GSSP é um problema de otimização combinatória que envolve a alocação de um determinado número de materiais em containers disponíveis sob restrições de segregação (separação física) [4].

O GSSP foi introduzido como uma generalização do problema de armazenamento segregado (SSP, do inglês "segregated storage problem"). O SSP tem sido estudado por vários autores, os quais se dedicam a propor métodos para resolvê-lo. De modo sucinto, cabe o destaque a Shilfer e Naor [5], que introduziram a formulação do SSP ao realizar um estudo cujo objetivo consistia na introdução de um método para calcular a divisão ótima de um armazém em compartimentos de modo a minimizar os custos anuais de armazenamento e o investimento de capital. Em particular, a motivação do estudo foi o cenário em que diferentes variedades de grãos devem ser armazenadas simultaneamente e separadamente.

O GSSP é definido por um conjunto de bens, para o qual a quantidade de cada bem é conhecida, um conjunto de compartimentos, para o qual a capacidade de cada compartimento é conhecida, e o custo unitário de armazenamento de cada bem. As restrições dizem respeito às capacidades dos compartimentos, a segregação adequada das mercadorias, uma vez que podem existir armazenamentos incompatíveis, e à geometria e estrutura dos compartimentos. O objetivo, portanto, da solução do GSSP é determinar uma remessa e sua alocação que satisfaça todos os requisitos e gere um custo mínimo.

Em resumo, o SSP consiste no armazenamento ótimo de bens tal que no máximo um bem ocupe um compartimento. Este tipo de segregação é a forma mais simples de segregação e é chamada de *segregação exclusiva* – e.g., armazenamento de diversos grãos em diferentes compartimentos do silo, de diferentes tipos de alimentos em compartimentos de um freezer, de diferentes tipos de

líquidos em um caminhão-tanque ou diferentes tipos de petróleo bruto em tanques de armazenamento. Entretanto, existem outros modos de segregação e o GSSP é a generalização do SSP que permite estes modos, como a *segregação binária*, na qual a possibilidade de armazenamento dos bens se resume a compatível e incompatível, e a *segregação discreta*, na qual a compatibilidade dos bens ocorre de forma específica dentro de algumas restrições. A imagem abaixo, retirada de [4], exemplifica a segregação discreta, muito comum em navios de carga por causa dos bens perigosos, explicados na seção 2.

Table 2
Table of segregation of freight containers on board container ships [23]

Segregation requirement	Vertical				Horizontal					
	Closed versus closed	Closed versus open	Open versus open		Closed versus closed		Closed versus open		Open versus open	
					On deck	Under deck	On deck	Under deck	On deck	Under deck
"Away from" (1)	One on top of the other permitted	Open on top of the other permitted otherwise as for open versus open	Not in the same vertical line unless segregated by a deck	Fore and aft	No restriction	No restriction	No restriction	No restriction	One container space	One container space or one bulkhead
				Athwartships	No restriction	No restriction	No restriction	No restriction	One container space	One container space
"Separated from" (2)	Not in the same vertical line unless segregated by a deck	As for open versus open	Not in the same vertical line unless segregated by a deck	Fore and aft	One container space	One container space or one bulkhead	One container space	One container space or one bulkhead	One container space	One bulkhead
				Athwartships	One container space	One container space	One container space	Two container space	Two container space	One bulkhead
"Separated by a complete compartment or hold from" (3)	Not in the same vertical line unless segregated by a deck	As for open versus open	Not in the same vertical line unless segregated by a deck	Fore and aft	One container space	One bulkhead	One container space	One bulkhead	Two container space	Two bulkheads
				Athwartships	Two container space	One bulkhead	Two container space	One bulkhead	Three container space	Two bulkheads
"Separated longitudinally by an intervening complete compartment or hold from" (4)	Prohibited	Prohibited	Prohibited	Fore and aft	Minimum horizontal distance of 24 m	One bulkhead and minimum horizontal distance of 24 m	Minimum horizontal distance of 24 m	Two bulkheads	Minimum horizontal distance of 24 m	Two bulkheads
				Athwartships	Prohibited	Prohibited	Prohibited	Prohibited	Prohibited	Prohibited

Note: All bulkheads and decks should be resistant to fire and liquid.

D. Barbucha / European Journal of Operational Research 156 (2004) 54–72

Figura 1: Exemplo de restrições para segregação discreta

Em relação à modelagem em grafos, o GSSP possui diversas abordagens, principalmente por causa dos diferentes modos de segregação. Similaridades entre tais abordagens envolvem a escolha de representação dos bens por meio dos vértices e de suas relações por meio das arestas, além de coloração para o cálculo da quantidade mínima de compartimentos. Ocorrem particularidades tais quais pesos em vértices, pesos em arestas, pares ordenados (p, q) nos vértices

de modo que p é um bem e q é um compartimento, entre outras decisões. Isso porque o GSSP é um problema complexo em detalhes e as restrições podem resultar em muitas informações para a modelagem – vide a figura 1.

Vale destacar que a modelagem construída para este projeto está descrita na seção 3. A escolha é justificada pela simplicidade na abstração do problema, de modo que a segregação em questão é a binária.

4.2. Casos de teste

Para este trabalho, tendo em vista que o modo de segregação é a segregação binária, a informação núcleo de cada caso de teste é uma lista de vértices terminais de arestas. Em particular, os vértices terminais de uma mesma aresta indicam elementos incompatíveis dois a dois, conforme descrito em 3. O fato de a segregação ser binária, implica que é necessário saber apenas quais elementos são incompatíveis (vértices terminais), uma vez que não há outros condicionais, como tipo de material do container ou distância entre containers.

Uma vez definidos os vértices terminais, pode ser construído o grafo correspondente, com base em uma lista de adjacência, a qual determina todas as incompatibilidades daquele conjunto de elementos.

Portanto, cada entrada de teste consiste em um arquivo .txt cuja primeira linha indica o número de vértices e o número de arestas daquele grafo, separados por espaço em branco. Em seguida, há um número de linhas igual ao número de arestas, pois cada linha indica os vértices terminais de uma aresta do grafo, separados por espaço em branco.

Importante destacar que cada elemento deve ser representado como um inteiro não negativo sequencial. Isto é, se existem n elementos a serem armazenados, estes serão rotulados de 0 a $n-1$. Outro detalhe relevante é que o arquivo .txt não deve conter nenhuma linha em branco entre quaisquer informações ou no final. O número de linhas consiste em, exatamente, o número de arestas + 1.

Os arquivos de teste serão de autoria própria, baseados nos grafos da seção Gallery of named graphs da Wikipedia e nos grafos discutidos em aula.

4.3. Links

- D3 Graph Theory: plataforma que permite criar e manipular grafos, inclusive colorir.
  - D3 Graph Thoery: Graph Coloring
  - D3 Graph Thoery: k-Colorable Graph
  - D3 Graph Thoery: Chromatic Number

5. Proposta de abordagem algorítmica

Conforme discutido, o GSSP está diretamente relacionado à coloração de grafos. Em particular, destaca-se a busca pelo número cromático de um grafo G, notado por $\chi(G)$. Nesse sentido, dois trabalhos foram fundamentais para abordagem de solução a ser utilizada neste projeto. Um deles é [6], que agrupou e contextualizou alguns dos principais algoritmos para o problema de coloração, servindo de base para a escolha da abordagem algorítmica desde projeto e para o entendimento de diversos tópicos; o outro é [7], onde Daniel Brélaz propõe DSATUR, o algoritmo escolhido para este projeto.

O algoritmo DSATUR consiste em uma heurística gulosa para coloração de vértices, a qual é exata para grafos bipartidos. Para tanto, Brélaz definiu o conceito de grau de saturação de um vértice como o número de cores diferentes para as quais ele é adjacente. Os passos do algoritmo são descritos abaixo conforme propostos por Brélaz em 1979.

1. Organize os vértices por ordem decrescente de graus.
2. Pinte um vértice de grau máximo com a cor 1.
3. Escolha um vértice com um grau de saturação máximo. Se houver igualdade, escolha qualquer vértice de grau máximo no subgrafo induzido por vértices não-coloridos.
4. Pinte o vértice escolhido com a menor cor (menor número) possível.
5. Se todos os vértices estiverem coloridos, pare. Caso contrário, volte para 3.

Brélaz notou que atribuir uma ordem para a coloração, seguindo a saturação dos vértices, implica que a prioridade é dada aos vértices com menos opções possíveis. Por isso, é possível obter uma coloração com limitante superior mais próximo do número cromático com menos iterações.

Por fim, vale destacar que o trabalho de Brélaz envolve dois métodos. Um deles é o escolhido para o escopo deste trabalho, o qual foi descrito de modo sucinto nesta seção. Segundo Brélaz, tal método performou de modo satisfatório para grafos de até 100 vértices e de densidade entre 0,3 e 0,7. Entretanto, Brélaz propôs um outro método, uma modificação do algoritmo look-ahead de Randall-Brown (1972). A escolha da versão mais simples se justifica pela sua menor complexidade de compreensão e pelo fato de que o grupo amostral suportado é suficiente para o problema aqui em questão.

6. Descrição dos experimentos

6.1. Complexidade algorítmica

A implementação realizada tem complexidade temporal $O((V + E) \cdot \log V))$, onde V é o número de vértices e E é o número de arestas, tendo em vista que percorrer a lista de adjacência leva $O(V + E)$ e a busca no set leva $O(\log V)$.

Em relação à complexidade espacial, a complexidade da lista de adjacência é, em geral, $O(V + E)$, chegando a $O(V^2)$ no pior caso, que consiste em cada vértice conectado a todos os outros – i.e., grafo completo (casos de teste 9, 10, 11, 12, 13 e 14).

6.2. Ambiente de execução

Em relação à máquina responsável pela realização dos testes, a Tabela 1 dispõe suas configurações de hardware e de sistema operacional.

Hardware		
Processador	Memória	Armazenamento
Intel® Core™ i5 – 10210U Quad Core Frequência: 1.60GHz até 4.20GHz 6MB Intel® Smart Cache	8GB RAM DDR4 Até 2400MHz	256GB SDD PCIe NVMe M.2 2280
Windows		
Edição	Versão	Arquitetura
Windows 11 Home Single Language	21H2	x86-64

Tabela 1: Configurações do computador e sistema usados

6.3. Execução dos casos de teste

A linguagem de implementação é C++ e a versão do compilador é g++ (GCC) 10.3.0.

O arquivo dsatur.cpp é o programa responsável pela organização dos elementos. Sua saída é a organização dos elementos em containers e a quantidade de containers necessária. O arquivo dsaturtime.cpp é exatamente igual ao dsatur.cpp com exceção da inserção da medição do tempo. No programa com o registro do tempo gasto, o algoritmo é chamado 10 vezes dentro de uma laço e o tempo exposto é a média do tempo dessas 10 execuções em CPU clocks.

O processo abaixo descreve compilação e execução de dsatur.cpp, mas o método é análogo para dsaturtime.cpp, sendo necessário apenas anterar o nome do arquivo no comando de compilação.

No terminal, o comando

```
$ g++ dsatur.cpp
```

compila e gera o executável de nome padrão, que pode ser executado com

```
$ ./a.out n.txt
```

onde n é o número do caso de teste (existem 14 arquivos de teste disponíveis).

6.4. Casos de teste

A Tabela 2 sintetiza as particularidades de cada caso de teste.

Em relação ao atributo "CPU clocks", ele indica o tempo médio da execução em CPU clocks. Para compreender esta unidade, deve-se ter em mente que a velocidade do clock mede o número de ciclos que a CPU executa por segundo, medido em gigahertz (GHz), e que um "ciclo" é tecnicamente um pulso sincronizado por um oscilador interno, o que pode ser compreendido como uma unidade básica para entender a velocidade de uma CPU. Durante cada ciclo, bilhões de transistores dentro do processador abrem e fecham. Assim, tendo em vista que a velocidade de clock da CPU utilizada para os testes é de 1.60GHz a 4.2GHz, considerando um valor médio de 2.9GHz, podemos concluir a CPU executa 2.9 bilhões de ciclos por segundo.

Os grafos dos casos de teste de 2 a 8 são clicáveis e redirecionam para a página Wiki correspondente.

Caso	Grafo	Vértices	Arestas	Densidade	$\chi(G)$ esperado	$\chi(G)$ obtido	CPU clocks
1	Grafo nulo	100	0	0	1	1	2.23e-05
2	Bidiakis cube	12	18	0.27272	3	3	1.97e-05
3	Brinkmann	21	42	0.2	4	4	4.08e-05
4	Dyck	32	48	0.09677	2	2	4.43e-05
5	Chvátal	12	24	0.36363	4	4	2.6e-05
6	Errera	17	45	0.33088	4	4	3.96e-05
7	Harries–Wong	70	105	0.04347	2	2	0.0001234
8	Biggs–Smith	102	153	0.02970	3	3	0.000151
9	K_7	7	21	1	7	7	2.03e-05
10	K_{10}	10	45	1	10	10	3.51e-05
11	K_{100}	100	4950	1	100	100	0.0027491
12	K_{200}	200	19900	1	200	200	0.0108381
13	K_{400}	400	79800	1	400	400	0.0512886
14	K_{800}	800	319600	1	800	800	0.211293

Tabela 2: Síntese de casos de teste

7. Análise e discussão

O algoritmo se mostrou eficiente, como pode ser percebido com base nos atributos $\chi(G)$ esperado, $\chi(G)$ obtido e CPU clocks. Nesse sentido, os tempos de resposta foram excelentes considerando a explicação realizada em 6.4. Isso porque, se são executados uma média de 2.9 bilhões de ciclos por segundo, então foram necessários aproximadamente 7,28597e-11 segundos (ou 0.00728597 milissegundos) no caso de teste mais longo (caso 14).

Para se ter ideia de quão rápido o programa executa a tarefa de organização dos elementos em containters, o tempo de reação do cérebro humano a estímulos externos é, em média, 150 a 300 milissegundos de acordo com este artigo. Ainda para comparação, o tempo médio de uma piscada humana é 100 milissegundos conforme este outro artigo.

Os tempos obtidos corresponderam às expectativas, uma vez que, em [7], na comparação de heurísticas por tempo, a única mais rápido que a Dsatur em todos os casos é a Welsh-Powell. Sobre os números cromáticos obtidos, também corresponderam ao esperado, dado que, no mesmo artigo, o único algoritmo capaz de apresentar uma taxa de erro menor que o Dsatur em todos os testes foi o DSI (Dsatur with interchanges), uma variação do Dsatur.

Os grafos dos casos de teste, coforme observado na Tabela 2, variam em ordem, tamanho e densidade. O objetivo dos K_n grafos era o estresse do algoritmo, já que estes são os grafos de maior densidade possível e maior número de arestas.

Não foi possível estabelecer relações entre os grafos com base em seus atributos. Os casos de teste foram limitantes nesse sentido, então, apesar de algumas relações poderem ser apontadas e seja tentador tirar conclusões a partir destas, não há um espaço amostral suficiente para afirmar qualquer especulação minimamente embasada em repetição de ocorrências. Apesar disso, ao ordenar os valores de tempo em ordem crescente, é perceptível que os grafos ficam quase que em ordem crescente de tamanho – como pode ser conferido no arquivo rescsv.xlsx. Portanto, é possível que a quantidade de arestas do grafo influencie

mais no tempo de coloração do que a quantidade de vértices.

Sobre a limitação do espaço amostrar, isso se justifica pelo fato de que os casos de teste ficaram restritos à disponibilidade de informações sobre o número cromático dos grafos para garantir a corretude e amparar os testes.

8. Conclusão

Portanto, considerando que a capacidade dos meios de armazenamento é limitada e alguns lucros para a empresa são esperados, foi cumprido o objetivo proposto de aplicar a teoria dos grafos para encontrar a alocação que atenda aos requisitos de segregação e proporcione o uso mais eficiente dos recursos. Em particular, tais resultados foram obtidos com eficiência por meio do algoritmo DSATUR (1979).

Referências

[1] D. Barbucha, W. Filipowicz, Segregated storage problems in maritime transportation, Elsevier (2017) 1–5.

[2] R. Lewis, A guide to graph colouring, Springer (2016) 1–2.

[3] R. Balakrishnan, K. Ranganathan, A textbook of graph theory, Springer (2012) 97–117, 143–175.

[4] D. Barbucha, Three approximation algorithms for solving the generalized segregated storage problem, European Journal of Operational Research (2002) 1–19.

[5] E. S. . P. Naor, Elementary theory of optimal silo storage design, Journal of the Operational Research Society (1961) 54—65.

[6] A. M. Lima, Algoritmos exatos para o problema da coloração de grafos, AcervoDigital da UFPR (2017) 21–28, 44–50.

[7] D. Brélaz, New methods to color the vertices of a graph, Communications of the ACM (1979) 251–256.