Algoritmos aplicados em um problema de otimização combinatória

Matheus Henrique, Antônio Carlos

Outubro, 2022

1 Introdução

A otimização combinatória é uma área desafiadora na ciência da computação e a na matemática aplicada. Trata-se de um conjunto de problemas que não se contentam apenas com uma das soluções possíveis, mas uma que também realize a melhor aplicação dos recursos disponíveis e maximize ou minimize a função objetivo. Frequentemente esses problemas tem uma série de restrições que levam a melhor solução. Os objetivos típicos da otimização combinatória passam por melhor aproveitamento de recursos, tempo, minimizar prejuízos e maximar lucros.

Deste modo, se faz necessário o estudo desses problemas pois os mesmos possuem diversas aplicações práticas. Algumas dessas aplicações são:

- Redes com restrições de conectividade.
- Roteamento de veículos
- Empacotamento de caixas em contêineres
- Atribuições de frequências em telefonia celular

O objetivo desse trabalho é resolver um problema de otimização combinatória, para auxiliar o maior guerreiro do Mundo de Zambis, Zorc, a recrutar seu exército para uma batalha. Zorc possui uma nave de peso W para transportar seu exército. Zorc percorrerá os i povos possíveis do Mundo de Zambis, um povo i é descrito por p_i, recrutando soldados de peso w_i e habilidade h_i. O mundo de Zambis tem um P povos. Alguns povos possuem caminhos entre si, outros não, por exemplo um povo i possui distancia d_{ij} de um povo j, são C caminhos entre os povos. A nave de Zorc só possui combustível para percorrer D metros entre os povos. O objetivo é realizar a otimização combinatória que máxime a habilidade do exército (função objetivo) e respeite as restrições de peso, distância e caminhos entre os povos. Duas técnicas serão aplicadas para auxiliar Zorc. Um algoritmo guloso que pode chegar aproximada ou exatamente da solução com maior habilidade. E um algoritmo que utilize o paradigma da programação dinâmica.

2 Formato de entrada e saída

A entrada é um arquivo com o seguinte padrão:

1. *Numero de instancias do arquivo*
2. *P D W C*
3. *p(i) w(i) h(i) (nas próximas P linhas tal que i vai de 1 até P)*
4. *p(i) p(j) d(i,j)*

Considerando H a habilidade total maximizada e S a quantidade total de soldados selecionados. A saída também é gerada em um arquivo seguindo o padrão: *H p(i) S(i).*

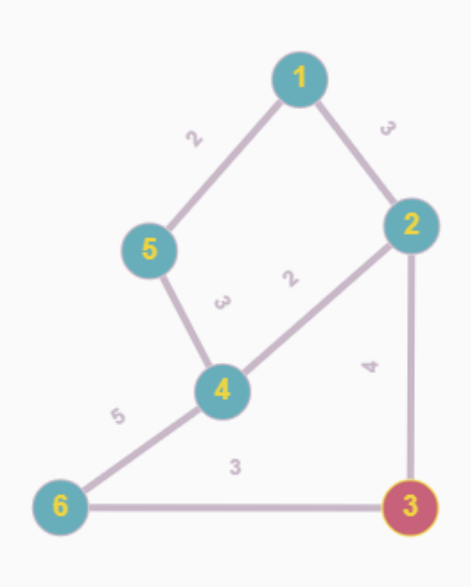

Figura 1: Grafo

3 Modelagem

A estrutura de dados utilizada no problema é um grafo G = (V, A) ponderado e não dirigido, tal que os vértices V representam os P povos e as arestas A representam os C caminhos. O grafo está representado por uma matriz de adjacências, com dimensões $|V|$ x $|V|$, M = (a_{ij}) tal que:

$$a_{ij} = \begin{cases} 0, & \text{se i = j} \\ \mathrm{d}_{ij}, & \text{se } \mathrm{d}_{ij} \in \mathrm{A} \quad \text{e } a_{ij} = a_{ji} \\ -1, & \text{se } d_{ij} \notin \mathrm{A} \end{cases}$$

O grafo da Figura 1 tem a matriz de adjacência:

$$\begin{bmatrix} 0 & 3 & -1 & -1 & 2 & -1 \\ 3 & 0 & 4 & 2 & -1 & -1 \\ -1 & 4 & 0 & -1 & -1 & 3 \\ -1 & 2 & -1 & 0 & 3 & 5 \\ 2 & -1 & -1 & 3 & 0 & -1 \\ -1 & -1 & 3 & 5 & -1 & 0 \end{bmatrix}$$

Definidas as variáveis W como o peso da nave, P como os povos possíveis e D como a distância que pode ser percorrida. Seja $x_1, ..., x_P$ a quantidade de soldados selecionados de cada povo, cada um com seu respectivo peso $w_1, ..., w_P$ e habilidade $v_1, ..., v_P$, maximizar a habilidade total da nave é maximizar o somatório:

$$\sum_{k=1}^{P} v_k x_k, \text{ de tal forma que } (\sum_{k=1}^{P} w_k x_k) < W.$$

Essa relação deve obedecer a próxima, que restringe os caminhos entre os povos e a distância máxima que a nave deve percorrer.

Seja S = $\{p_i, p_j, p_k\}$ o conjunto dos povos que tiveram soldados selecionados para a guerra, $a_{ij} \neq -1$ e $a_{jk} \neq -1$, tal que $a_{ij} + a_{jk} \leq D$.

4 Soluções

4.1 Algoritmo guloso

Um algoritmo que pode nos levar a uma solução aproximada, ou com sorte até mesmo a ótima, em tempo polinomial é o algoritmo guloso. Sua implementação simples, rápida execução e a possibilidade de se chegar na melhor solução é o que torna esse algoritmo atrativo.

A estratégia gulosa atribui uma razão r_i para cada povo p_i definida por $r_i = \frac{p_i}{h_i}$. Em seguida os povos são ordenados de forma decrescente da maior para a menor razão.

O conjunto X de candidatos recebe cada $p_i \in P$. O conjunto Y da solução está inicialmente vazio.

Inicialmente o algoritmo seleciona p_P que é o ultimo povo do conjunto X (melhor povo, menor r_i) e o adiciona na solução Y. Feito isso, enquanto X não estiver vazio, o algoritmo executa os passos:

1. Seleciona p_{P-1}
2. Se viável então adiciona em Y
3. Se inviável então descarta
4. Diminui P em uma unidade

O elemento ser viável ou não é decidido verificando se há caminhos ligando o p_i analisado ao p_j já adicionado em Y e se $d_{ij} \leq D$. W e D são atualizados a cada inserção em Y. H é maximizada sempre selecionando o maior número possível de soldados x_i para cada povo adicionado na solução.

Pseudocódigo:

```
function guloso(povos[], grafo)
    ordena_os_povos(povos);

    while C != vazio do:
        Y = povos[quantidade_de_povos];
        aux = povos[quantidade_de_povos - 1]

        if existe_caminho(aux, povos[quantidade_de_povos]) then:
            if distancia - aux->distancia >= 0 then:
                Y = aux
            endif
        endif
        quantidade_de_povos = quantidade_de_povos - 1
endfunction
```

4.2 Programação dinâmica

A programação dinâmica resolve problemas combinando as soluções para subproblemas, quando eles se sobrepõem. A sacada desse tipo de algoritmo é resolver cada subproblema apenas uma vez, gravando sua resposta e evitando recálculos.

Analisando os subproblemas podemos pensar em naves com capacidades menores tal que $w \leq W$. Assim formulamos $H(w)$ como o maior valor alcançado para uma nave de capacidade w.

Se a solução $H(w)$ é ótima e ela contém o povo $p_(i)$, então também temos uma solução ótima $H(w - w_i)$ sem o povo $p_(i)$. Deste modo podemos formular $H(w) = H(w - w_i) + v_i$. Assim para todo p_i temos que:

$$H(w) = \max_{p_i : w_i \leq w} \{H(w - w_i) + v_i\}$$

O algoritmo que resolve o problema cria duas tabelas de tamanho W+1. A primeira denominada T armazena a habilidade maximizada na posição w_i sempre aplicando a fóruma de maximização. A segunda, denominada S, armazena uma referencia para uma lista encadeada de povos p_i, que descrevem o caminho que resulta na habilidade armazenada em T na posição w_i.

	Tabela T											
Indice	0	1	2	3	4	5	6	7	8	9	10	11
H(w[i])	2	3	3	5	8	15	15	15	19	23	32	36

	Tabela S											
Indice	0	1	2	3	4	5	6	7	8	9	10	11
Referencia	*	*	*	*	*	*	*	*	*	*	*	*
	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
	3	3	3	3	3	3	3	3	3	3	3	3
		5	5	5	5	5	5	5	5	5	5	5
					2	2	2	2	2	2	2	2
						4	4	4	4	4	4	4
									1	1	1	1
											6	6
												7

Figura 2: Exemplo de Memorização para uma nave de tamanho 10

Desta forma, a melhor solução utilizando a capacidade máxima da nave, para todos os povos, está na posição T[W+1]. Seu caminho é memorizado na tabela S na mesma posição.

Encontrada a melhor solução é necessário analisar se ela atende as restrições de distancia máxima e de caminhos entre os povos. Caso atenda, a solução ótima foi encontrada. Caso contrário deve-se analisar a solução anterior.

5 Análise de complexidade

A complexidade do algoritmo guloso se dá principalmente por sua rotina de ordenação, na qual é usado o algoritmo selecition sort que tem complexidade $O(P^2)$ tal que é o número de povos. A seleção dos povos ocorre em um loop que será executado P vezes. Complexidade geral: $O(\max(P^2, P)) = O(P^2)$.

A complexidade da programação dinâmica é dada pelo preenchimento das tabelas unidimensionais, de tamanho W+1. O anel mais interno leva um tempo fixo de O(P) que é chamado W vezes, logo a complexidade geral será de O(PW).

6 Análise dos resultados

A programação dinâmica se mostrou assertiva para encontrar a solução ótima em todos os casos testados enquanto a o algoritmo guloso encontrou boas aproximações. Para uma quantidade pequena de povos o algoritmo guloso se mostrou mais rápido enquanto para grandes quantidades de povos a programação dinâmica foi melhor. Para naves com pesos pequenos o algoritmo guloso também foi mais eficiente.

7 Máquina utilizada

Processador AMD Ryzen 5 2500U
Memória RAM 12GB
Sistema operacional Linux