Algoritmo Guloso

Darcinel Litaif Junior[1], João Cabral[2], Robert Oliveira[3], Romilso Silva[4], Victor Farias[5]

[1]Fundação, Centro de Análise Pesquisa e Inovação Tecnológica, Faculdade Fucapi (Instituto de Ensino Superior Fucapi) – Manaus– AM – Brazil

darcinel.litaiff.junior@gmail.com, joaocabral1232@gmail.com, robertholiveira15@gmail.com, romilsolive73@gmail.com, victor.alvesf7@gmail.com

Abstract. *The goal of the greedy algorithm is to choose the option that best suits that moment in order to satisfy the proposed objective, but every decision can have a consequence, since it does not reevaluate the decisions made. One of the problems used to use this algorithm is the problem of the change, where it aims to give the change to a person with a smaller number of bills or coins. The result obtained in this problem will depend greatly on the values reported, that is, the more data are informed to the algorithm, the greater the chance of obtaining an optimal solution. Therefore, we can conclude that the greedy algorithm can be quite efficient and easy to implement, depending on the amount of information until a certain moment, it can achieve an optimal solution.*

Resumo. *O objetivo do algoritmo guloso é escolher a opção que mais se adequa para aquele momento a fim de satisfazer o objetivo proposto, mas a cada decisão tomada pode haver uma consequência, pois o mesmo não reavalia as decisões tomadas. Um dos problemas utilizado para se utilizar este algoritmo é o problema do troco, onde este tem como objetivo dar o troco à uma pessoa com menor número de cédulas ou moedas possíveis. O resultado obtido nesse problema irá depender muito dos valores informados, ou seja, quanto mais dados forem informados ao algoritmo, maior a chance de se obter uma solução ótima. Portanto, podemos concluir que o algoritmo guloso pode ser de bastante eficiência e de fácil implementação, dependo da quantidade de informações até um determinado momento, ele pode alcançar uma solução ótima.*

1. Introdução

Grande parte dos problemas computacionais estão relacionados a busca por uma solução atendendo certas propriedades nesse sentido a otimização combinatória é uma área que engloba grande quantidade de problemas e que busca por soluções que façam melhor uso dos recursos envolvidos. Nestes problemas, não basta apenas se ter qualquer tipo de solução dentre as possíveis, mas sim uma solução que traga uma otimização considerável levando em conta os objetivos de interesse. Os objetivos mais comuns dos problemas de otimização combinatória, consistem de forma geral em otimizar o tempo

de determinadas ações e atividades, para que dessa forma os prejuízos e gastos sejam minimizados aumentando assim os lucros.

Os problemas de otimização combinatória podem ser de minimização ou de maximização. Em ambos os casos, se tem uma função aplicada a um domínio finito, por este motivo o domínio da função é em geral grande e algoritmos que verificam cada elemento deste domínio se tornam impraticáveis de serem utilizados. Desta forma, surge a necessidade de usar técnicas mais elaboradas para encontrar soluções ótimas, que pode ser de valor mínimo, caso o problema seja de minimização, ou máximo, caso o problema seja de maximização. Contudo, existem problemas importantes que necessitam de soluções, mesmo estas não sejam de valor ótimo, mas que de preferência tenham um valor aproximado do ótimo. As soluções são obtidos após o processamento computacional, e diante da complexidade dos problemas computacionais, surge a necessidade de buscar soluções que otimizam o processo proposto a solucionar tais problemas. Existem diversas soluções para se otimizar uma solução de um determinado problema, sendo uma delas os algoritmos gulosos. Os algoritmos gulosos estão associados a técnicas simples, intuitivas e, de forma ampla, é um algoritmo fácil de ser implementado. Dessa forma, é uma das técnicas mais utilizadas no projeto de algoritmos para problemas relacionados à otimização combinatória. A idéia básica do projeto de um algoritmo guloso é, iterativamente selecionar um elemento que melhor contribua para a construção de uma solução.

Neste texto apresentamos alguns tópicos relacionados ao algoritmos guloso. O texto está organizado da seguinte forma. Na Seção 2 a seguir, apresenta-se o background relacionado ao algoritmo. Na Seção 3 apresenta-se trabalhos relacionados ao algoritmo nos quais avaliam a sua utilização e desempenho em comparação a outros algoritmos. Nas seção 4 discutem-se a explicação do algoritmo algoritmo guloso. A seção 5 é dedicada a uma discussão dos resultados obtidos sobre os testes realizados e a observação de comportamento do algoritmo guloso. A seção 6 apresenta o relatório com o código do algoritmo. Finalmente, na Seção 7 são feitas algumas considerações finais sobre o algoritmo guloso.

2. Background

Existem inúmeras casos de problemas que a primeira vista parecem ser de simples solução, mas que na verdade exigem um nível elevado de dificuldade. A dificuldade está relacionada a agrupar todas as informações e processá-las de forma a gerar como buscam minimizar ou maximizar uma função através da escolha dos valores de variáveis reais ou inteiras dentro de um conjunto possível visando encontrar uma solução ótima para um dado problema, uma solução eficiente. Para lidar com esses problemas, foram surgindo algoritmos para facilitar a tomada de decisão para que dessa forma os algoritmos indiquem uma solução adequada para que um determinado objetivo seja atingido com sucesso. Tais algoritmos que possuem como característica gerar uma solução eficaz são chamados de algoritmos de otimização, pois a partir do seguimento de uma sequência de etapas conseguem indicar uma solução adequada entre as possibilidades existentes para problemas. Técnicas de otimização buscam minimizar ou maximizar uma função através da escolha dos valores de variáveis reais ou inteiras dentro de um conjunto possível visando encontrar uma solução ótima para um dado problema.

Um exemplo de estratégias existentes para que seja possível resolver problemas de otimização são os algoritmos gulosos ou greedy algorithms onde de acordo com Silveira (1999) a cada passo o algoritmo escolhe a opção que parece ser a melhor naquele determinado momento com o intuito de satisfazer o objetivo proposto. O nome guloso vem justamente do fato do algoritmo selecionar sempre a melhor opção no momento, escolhendo sempre o melhor pedaço, sem se preocupar com as consequências futuras. De acordo com Feofiloff (2015) um algoritmo guloso é aquele que escolhe a cada iteração o objeto mais apetitoso, e Koerich (2006) escolhe a melhor alternativa local esperando que isso leve a uma solução ótima global, que pode ser alcançada escolhendo a escolha ideal em cada etapa.

De maneira geral, o algoritmo guloso está relacionado a otimização, ou seja, está ligado a uma sequência de passos, onde em cada passo existem escolhas a serem feitas que resultarão em uma solução que pode ser ótima ou não. O motivo pelo qual os algoritmos gulosos nem sempre apresentam soluções ótimas é devido ao ao fato de que a tomada de decisão tem como base apenas a informação disponível em um determinado momento, dessa forma o algoritmo não reavalia as decisões tomadas, independentemente das consequências. Devido ao fatores citados, não se faz necessário uma avaliação para uma adequação das alternativas levando em consideração as decisões tomadas anteriormente a fim de que não sejam alteradas. Entretanto, de forma ampla o algoritmo guloso é fácil de se implementar e são eficientes.

3. Trabalhos relacionados

O artigo de Sorroche (2003) apresenta um sistema para sugestão de horário de matrícula de alunos, implementado com o objetivo de fornecer diversas possibilidades de turmas e horários para os alunos escolherem o que melhor se adequa a ele. Para isto foram implementados dois algoritmos de grafos e comparados os seus desempenhos. No sistema, o aluno escolhe a disciplina desejada e automaticamente é mostrada uma grade com os horários semanais da disciplina. Se houver coincidência de horário o mesmo é mostrado com uma cor diferente e uma mensagem ao aluno é emitida, caso contrário, a disciplina pode ser incluída pelo aluno em sua grade. Neste artigo foram implementados dois algoritmos, baseados na Teoria dos Grafos, para efeito de comparação; o Algoritmo Guloso (Szwarcfiter, 1984), é um método baseado no trabalho de Schwarz que implementou o modelo proposto por Bron e Kerbosh em Bron (1973). Em relação aos resultados obtidos, o algoritmo de Bron e Kerbosh apresenta um tempo menor de resposta, especialmente para grafos maiores. No algoritmo Guloso constatou-se que a verificação feita após a geração de uma sugestão para determinar se o conjunto formado é maximal, consome um tempo razoável. No entanto, que ambos os algoritmos se mostram aceitáveis para a geração das sugestões de matrícula, e também que os dois algoritmos, obtiveram um tempo médio por sugestão de menos de um segundo, o que considerando se tratar de uma aplicação web. Os dois algoritmos utilizados na implementação do sistema mostraram-se adequados, destacando-se o fato de que o algoritmo de Bron e Kerbosh apresentou melhor desempenho nos testes realizados.

Cravo (2006) apresentou um algoritmo guloso e uma heurística GRASP alcançando excelentes resultados para as instâncias utilizadas. O problema da rotulação cartográfica de pontos consiste em rotular o pontos de um mapa evitando as sobreposições dos

rótulos. Este trabalho apresenta uma heurística gulosa para esse problema, baseada no grafo de conflitos produzido. Este trabalho surgiu devido a identificação do crescimento das aplicações Desktop e voltadas para WEB, que permitem mostrar e tomar decisões baseadas em mapas georreferenciados. Surgiu então a necessidade de desenvolver algoritmos eficientes que permitam posicionar os textos em um mapa, evitando conflitos e, conseqüentemente, mapas "poluídos" visualmente. Como conclusão, este trabalho apresenta uma heurística gulosa, baseada em grafos de conflitos, que produz bons resultados quando aplicada em algumas instâncias presentes na literatura. Este trabalho apresentou uma heurística gulosa para o problema da rotulação cartográfica de pontos. Os resultados foram bem satisfatórios, superando heurísticas e metaheurísticas conhecidas, com um baixo tempo computacional, mesmo para as maiores instâncias, e ainda rotulações de boa qualidade, superando vários algoritmos, como a Busca Tabu, de Yamamoto et al. (2002), e os resultados médios do Algoritmo Genético Construtivo (AGC), de Yamamoto e Lorena (2005). . Os tempos computacionais também foram baixos, fortalecendo a sua importância.

O artigo de C. dos Santos (2019) apresenta um modelo de programação inteira mista para o planejamento de redes celulares de terceira geração, considerando a localização de estações rádio-base, controle de potência e múltiplos serviços. O problema resultante é NP-difícil e, por este motivo, para resolver instâncias de médio e grande porte em tempo hábil, tornou-se necessária a utilização de heurísticas onde foram propostos dois diferentes algoritmos, uma heurística gulosa e um algoritmo genético. Em relação aos resultados obtidos, os experimentos realizados no contexto deste estudo indicam que ambos os métodos implementados são capazes de solucionar rapidamente as três classes de instâncias de diferentes tamanhos. Os resultados comparativos mostraram que o algoritmo Genético realizou um planejamento mais eficaz da rede, sob a métrica relativa ao custo de instalação das estações rádio-base no entanto o algoritmo Guloso, apresentou resultados próximos ao do Genético, num tempo computacional bem inferior ao deste, se mostrando muito útil e relevante para o contexto deste trabalho.

4. Explicação do método

Uma forma de exemplificar o comportamento do algoritmo guloso pode ser feitas por meio de uma de suas aplicações como o problema do troco mínimo. O problema tem como proposta trocar uma determinada quantidade de dinheiro no menor número de moedas possíveis. É uma técnica para problemas de otimização, onde se espera uma resposta, buscando uma solução ótima. Basicamente, dado um valor que pode ser chamado de troco se busca representar esse determinado valor com o menor número de cédulas ou moedas possível, utilizando assim algoritmos gulosos para se obter esta solução.

Para possibilitar um melhor entendimento de como os algoritmos gulosos trabalham vamos dar um exemplo. Supondo que se tem a disposição moedas com valores de R$ 0,1, R$ 0,5, R$ 0,10, R$ 0,25, R$ 0,50 e R$ 1,0. O problema consiste em criar um algoritmo para que se possa obter como resposta uma solução, neste caso um troco com o menor número de moedas possível .Suponha que é preciso "dar um troco" de R$0,75. A melhor solução, isto é, o menor número de moedas possível para obter o valor, consiste em 2 moedas: 1 de valor R$ 0,50 e outra de valor R$ 0,25. De maneira geral, a

ação de um algoritmo guloso em cada etapa é selecionar a moeda de maior valor possível, que atenda as exigências de forma a não passar da quantia necessária. Neste primeiro exemplo o resultado obtido foi uma solução ótima, ou seja, a solução mais eficiente.

Mas, nem sempre o algoritmo guloso irá retornar uma solução eficiente para os problemas, pois de acordo com Cancela (2005) existe a "esperança" de se chegar à solução ótima porque o Método Guloso não garante o retorno de uma solução ótima, e também segundo Blum (2001) o método pode gerar algoritmos exatos ou de forma mais abrangente algoritmos heurísticos, onde a otimalidade não é garantida.

Supondo que se tem a disposição moedas fictícias com valores de R$ 0,1, R$ 0,7 e R$ 0,10 no qual se faz necessário retornar um troco de R$0,14. A solução ótima para obter o valor, consiste em 2 moedas de valor R$ 0,07, no entanto a solução dada pelo algoritmo é correta, mas não é eficiente, no qual consiste em 4 moedas: 1 de valor R$ 0,10 e 4 no valor de R$ 0,01. Isso se deve ao fato de que é muito difícil com valores de moedas não fictícias, e também tendo em disposição uma quantidade razoável de cada moeda o algoritmo guloso dar como resposta para o problema uma solução não ótima. Por outro lado, quando o problema utiliza como informação moedas fictícias, ou uma quantidade limitada de cada moeda disponível, o algoritmo guloso pode vir a não chegar em uma solução ótima, ou vir a chegar à solução alguma mesmo ela existindo, e em casos extremos entrar em loop infinito.

As características principais do algoritmo guloso que o difere em comparação com os outros algoritmos de otimização está relacionado ao aspecto onde em cada etapa ele escolhe o maior valor possível, onde no exemplo relacionado ao problemo do troco mínimo que possui como informação moedas reais e fictícias o maior valor utilizado foram respectivamente a moeda de R$1 e a moeda de R$0,10. Outra característica relacionada ao algoritmo guloso está em não reavaliar as decisões tomadas, ou seja, a partir do momento em que uma moeda foi escolhida como a melhor opção em um determinado momento, ela não será retirada sob nenhuma circunstância do conjunto solução que o algoritmo irá propor.

5. Resultados obtidos

Foram escolhidas 2 bases de dados, a primeira está relacionada a base de dados com notas reais, e a segundo está relacionada a base de dados com notas fictícias. Ambos foram utilizados nos códigos mostradas na seção 6 deste artigo.

Tabela 1. Tabela de base de dados utilizadas

Base de dados notas reais	Base de dados notas fictícias

R$ 10 R$ 5 R$ 2 R$ 1	R$ 10 R$ 7 R$ 1

Para exemplificar como foi feita a interação com o algoritmo e também mostrar como os testes foram realizados a figura 1 mostra as soluções geradas utilizando a base de dados com notas fictícias.

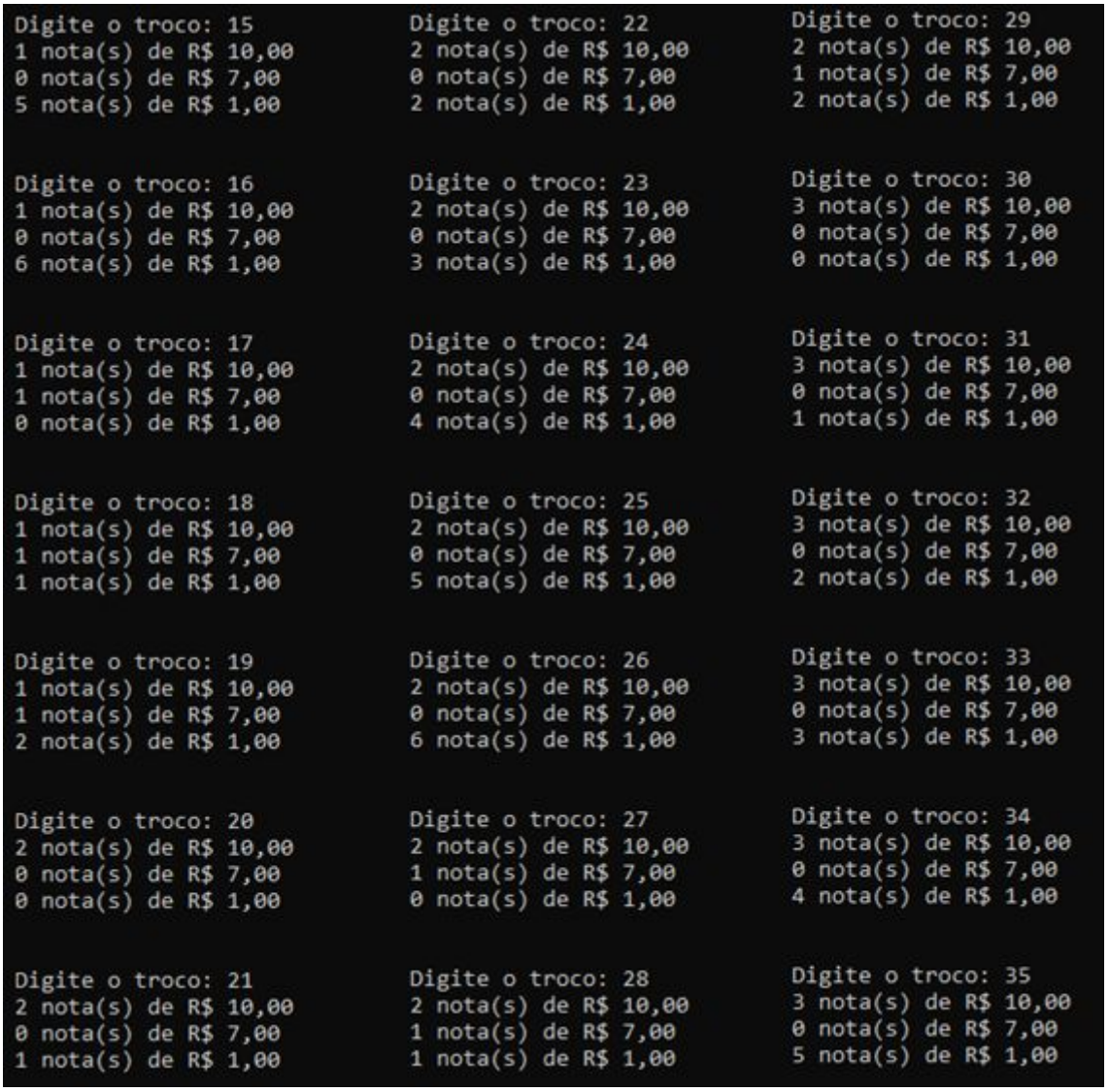

Fig. 1. Interação do algoritmo guloso com base de dados de notas fictícias

A tabela 2 mostra uma comparação das soluções obtidas com a base de dados fictícia tanto pelo algoritmo guloso com pelo algoritmo de programação dinâmica. É possível perceber que dos 21 troco utilizados, 13 apresentam solução ótima global, e possuem a

mesma quantidade de notas usadas em comparação com o algoritmo de programação dinâmica. No entanto 8 troco relacionados ao algoritmo guloso apresentaram soluções muito distantes do ideal, onde em alguns casos chegaram a utilizar o dobro de notas como solução em comparação com a quantidade de notas utilizadas pelo algoritmo de programação dinâmica.

Tabela 2. Testes realizados com a base de dados com notas fictícias

	Algoritmo Guloso	Solução Ótima Global (algoritmo de Programação Dinâmica)
Troco a ser dado (Entrada) utilizando a base de dados de notas fictícias conforme a tabela 1	Quantidade de notas usadas para o troco	Quantidade de notas usadas para o troco
R$ 15	6	3
R$ 16	7	4
R$ 17	2	2
R$ 18	3	3
R$ 19	4	4
R$ 20	2	2
R$ 21	3	3
R$ 22	4	4
R$ 23	5	5
R$ 24	6	3
R$ 25	7	4
R$ 26	8	5

R$ 27	3	3
R$ 28	4	4
R$ 29	5	5
R$ 30	3	3
R$ 31	4	4
R$ 32	5	5
R$ 33	6	6
R$ 34	7	4
R$ 35	8	5

Para exemplificar como foi feita a interação com o algoritmo e também mostrar como os testes foram realizados a figura 2 mostra as soluções geradas utilizando a base de dados com notas reais.

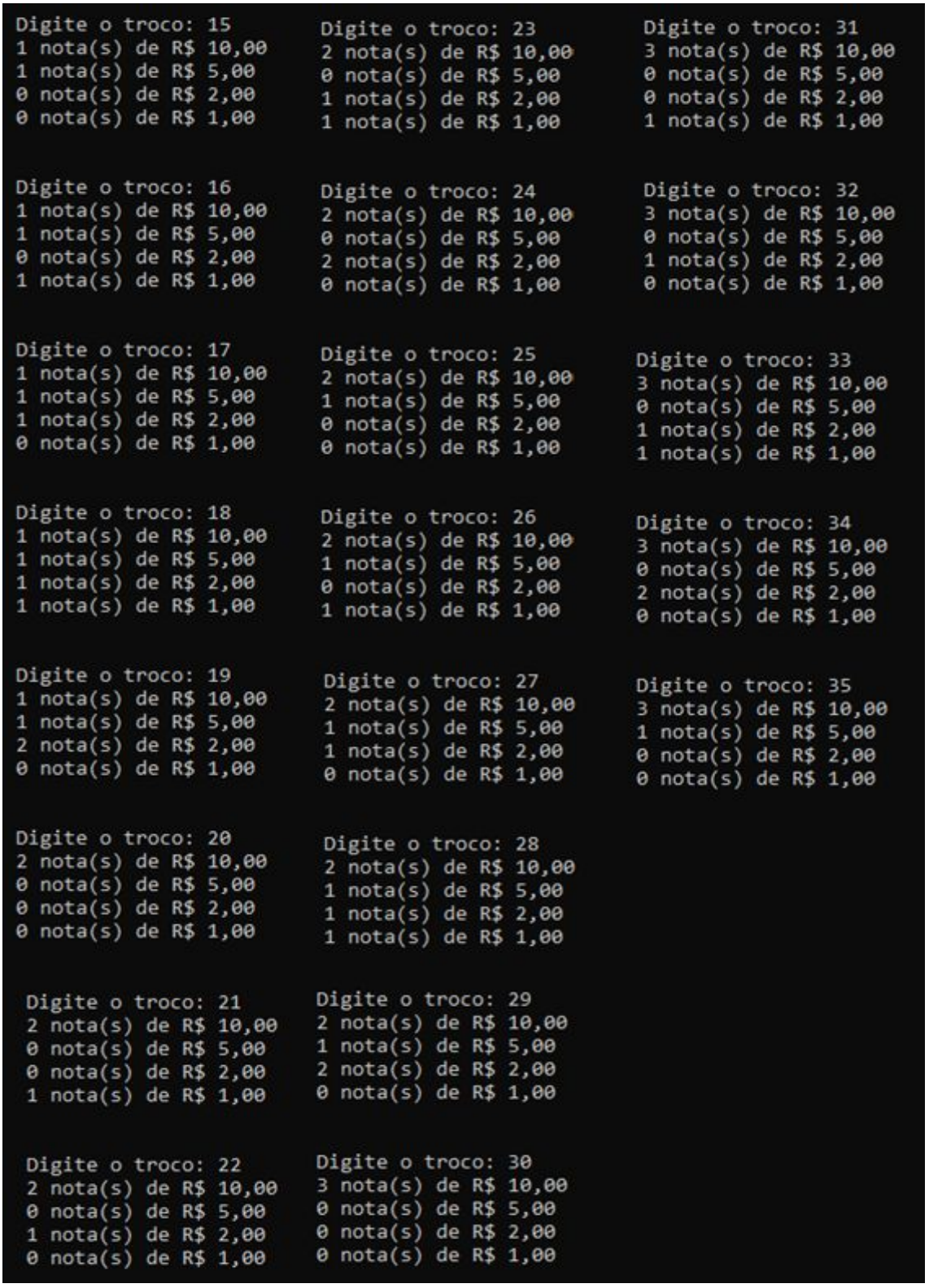

Fig. 2. Interação do algoritmo guloso com base de dados de notas reais

A tabela 3 mostra uma comparação das soluções obtidas com a base de dados fictícia tanto pelo algoritmo guloso com pelo algoritmo de programação dinâmica. É possível perceber que todos os 21 troco utilizados apresentam solução ótima global, e possuem a

mesma quantidade de notas usadas em comparação com o algoritmo de programação dinâmica.

Tabela 3. Testes realizados com a base de dados com notas reais

Solução Gulosa	Solução apresentada	Solução Ótima Global (algoritmo de Programação Dinâmica)
Troco a ser dado (Entrada) utilizando a base de dados de notas reais conforme a tabela 1	Quantidade de notas usadas para o troco	Quantidade de notas usadas para o troco
R$ 15	2	2
R$ 16	3	3
R$ 17	3	3
R$ 18	4	4
R$ 19	4	4
R$ 20	2	2
R$ 21	3	3
R$ 22	3	3
R$ 23	4	4
R$ 24	4	4
R$ 25	3	3
R$ 26	4	4
R$ 27	4	4
R$ 28	5	5

R$ 29	5	5
R$ 30	3	3
R$ 31	4	4
R$ 32	4	4
R$ 33	5	5
R$ 34	5	5
R$ 35	4	4

6. Relatório com o código do algoritmo

Nesta seção são mostrados os códigos do troco mínimo utilizando algoritmo guloso e programação dinâmica para que seja possível comparar os dados gerados pelo algoritmo guloso utilizando base de dados com notas reais e fictícias a fim de comparar se a soluções do algoritmo guloso é uma solução ótimas global, e se não forem, comparar a diferença de notas com a solução gerado pela programação dinâmica que produz uma solução ótima global.

Código em C problema do troco (algoritmo guloso)

```
#include <stdio.h>
#include <math.h>


int main(void){
        //Parte de pergar os valores do dados.txt


        char url[] = "dados.txt";
        float dado;
        int i=0, quantidade = 0;
        FILE *arq;
        arq = fopen(url, "r");


        if(arq == NULL){
        printf("Erro ao abrir o arquivo\n");
```

```
return 0;
}

while (!feof(arq)){
if(fscanf(arq,"%f", &dado)){
        quantidade = quantidade + 1;
}
}
fclose(arq);
arq = fopen(url, "r");

int vetor[quantidade-1];

while (!feof(arq)){
if(fscanf(arq,"%f", &dado)){
        vetor[i] = dado;
        i = i + 1;
}
}
fclose(arq);

//Inicio do algoritmo em si

int troco, a = 1;
i = 0;
for(;;){

printf("\n Digite o troco: ");
scanf("%d", &troco);

for(i=0; i<quantidade-1; i++){
printf("%d nota(s) de R$ %d,00\n", troco/vetor[i], vetor[i]);
  troco = troco % vetor[i];
```

```
        }
}
        return 0;
}
```

Código em Python problema do troco com notas fictícias (programação dinâmica)

```
import sys

# m is size of coins array (number of different coins)
def minCoins(coins, m, V):

    # base case
    if (V == 0):
        return 0

    # Initialize result
    res = sys.maxsize

    # Try every coin that has smaller value than V
    for i in range(0, m):
        if (coins[i] <= V):
            sub_res = minCoins(coins, m, V-coins[i])

            # Check for INT_MAX to avoid overflow and see if
            # result can minimized
            if (sub_res != sys.maxsize and sub_res + 1 < res):
                res = sub_res + 1

    return res
# Driver program to test above function
coins = [10, 7, 1]
m = len(coins)
```

```
V = 15
print("15 : ",minCoins(coins, m, V))
V = 16
print("16 : ",minCoins(coins, m, V))
V = 17
print("17 : ",minCoins(coins, m, V))
V = 18
print("18 : ",minCoins(coins, m, V))
V = 19
print("19 : ",minCoins(coins, m, V))
V = 20
print("20 : ",minCoins(coins, m, V))
V = 21
print("21: ",minCoins(coins, m, V))
V = 22
print("22 : ",minCoins(coins, m, V))
V = 23
print("23 : ",minCoins(coins, m, V))
V = 24
print("24 : ",minCoins(coins, m, V))
V = 25
print("25 : ",minCoins(coins, m, V))
V = 26
print("26 : ",minCoins(coins, m, V))
V = 27
print("27 : ",minCoins(coins, m, V))
V = 28
print("28 : ",minCoins(coins, m, V))
V = 29
print("29 : ",minCoins(coins, m, V))

V = 30
```

```
print("30: ",minCoins(coins, m, V))
V = 31
print("31: ",minCoins(coins, m, V))
V = 32
print("32: ",minCoins(coins, m, V))
V = 33
print("33 : ",minCoins(coins, m, V))
V = 34
print("34 : ",minCoins(coins, m, V))
V = 35
print("35 : ",minCoins(coins, m, V))
```

Código em Python problema do troco com notas reais (programação dinâmica)

```
# A Naive recursive python program to find minimum of coins
# to make a given change V

import sys

# m is size of coins array (number of different coins)
def minCoins(coins, m, V):

    # base case
    if (V == 0):
        return 0

    # Initialize result
    res = sys.maxsize

    # Try every coin that has smaller value than V
    for i in range(0, m):
        if (coins[i] <= V):
            sub_res = minCoins(coins, m, V-coins[i])
```

```
            # Check for INT_MAX to avoid overflow and see if
            # result can minimized
            if (sub_res != sys.maxsize and sub_res + 1 < res):
                res = sub_res + 1

    return res

# Driver program to test above function
coins = [10, 5, 2, 1]
m = len(coins)

V = 15
print("15 : ",minCoins(coins, m, V))
V = 16
print("16 : ",minCoins(coins, m, V))
V = 17
print("17 : ",minCoins(coins, m, V))
V = 18
print("18 : ",minCoins(coins, m, V))
V = 19
print("19 : ",minCoins(coins, m, V))
V = 20
print("20 : ",minCoins(coins, m, V))
V = 21
print("21: ",minCoins(coins, m, V))
V = 22
print("22 : ",minCoins(coins, m, V))
V = 23
print("23 : ",minCoins(coins, m, V))
V = 24
print("24 : ",minCoins(coins, m, V))
V = 25
print("25 : ",minCoins(coins, m, V))
```

```
V = 26
print("26 : ",minCoins(coins, m, V))
V = 27
print("27 : ",minCoins(coins, m, V))
V = 28
print("28 : ",minCoins(coins, m, V))
V = 29
print("29 : ",minCoins(coins, m, V))

V = 30
print("30: ",minCoins(coins, m, V))
V = 31
print("31: ",minCoins(coins, m, V))
V = 32
print("32: ",minCoins(coins, m, V))
V = 33
print("33 : ",minCoins(coins, m, V))
V = 34
print("34 : ",minCoins(coins, m, V))
V = 35
print("35 : ",minCoins(coins, m, V))
```

7. Conclusão

Como foi mostrado nos resultados obtidos, o algoritmo guloso pode ser utilizado para tipos específicos de problemas, como nos casos de teste utilizados. O algoritmo guloso se mostrou extremamente eficiente utilizando a base de dados com notas reais, se equiparando as soluções geradas pelo algoritmo de programação dinâmica, onde ambos geraram soluções ótimas globais. No entanto quando o algoritmo guloso trabalha com outros tipos de informações, como as notas fictícias, ele se mostra muito pouco eficiente.

Foi possível identificar que o algoritmo tem suas vantagens e desvantagens dependendo do escopo definido do problema ou do tipo de dados informados, logo a sua vantagem é a fácil implementação, a execução rápida e estando no seu estado ideal pode gerar uma solução ótima. Na sua desvantagem irá depender dos

dados informados, resolve problemas específicos mas geram soluções não eficientes devido ao fato de suas decisões serem irrevogáveis..

Referências

BLUM, C.; ROLI, A. Metaheuristics in Combinatorial Optimisation: Overview and Conceptual Comparison. Technical Report TR/IRIDIA/2001-13, IRIDIA, Université Libre de Bruxelles, Belgium, 2001.

BRON, COEN; KERBOSCH, JOEP. Finding all cliques of an undirected graph. The

Communications of the ACM, Eindhoven, v. 16, n. 9, p. 575-577, set. 1973.

CANCELA, H.; ROBLEDO, F.; RUBINO, G. A GRASP Algorithm for Designing a Wide Area Network Backbone. Research Report LIMOS/2005, Faculatad de Ingeniería, Uruguay, 2005.

C. LAGE DOS SANTOS, KATIA & A. CABRAL, GLEICY & MATEUS, GERALDO. (2019). Planejamento de Redes Celulares de Terceira Geração Utilizando Algoritmos Genéticos e Heurísticas Gulosas.

CRAVO, G.L.; RIBEIRO, M.G. LORENA L.A.N. Heurística gulosa para o problema da rotulação cartográfica de pontos. 2006. Revista Educação Tecnológica, Ano 2, Número 1 Abr/Set 2006– Departamento de Ciência da Computação e Informática Faculdade Aracruz - Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 2006.

FL´AVIO K MIYAZAWA AND CID C DE SOUZA. Introdução a otimização combinatória. In Jornadas de Atualização em Informática - Congresso da Sociedade Brasileira de Computação - JAI-SBC. SBC, 2015.

FEOFILOFF, P. "Algoritmos gulosos", 2015.

KOERICH, A. "Estratégia Gulosa, Técnicas de Projeto de Algoritmos", 2006.

MORAIS, Camila Mendonça. Algoritmo guloso. 2014. 65 f. Dissertação (Programa de Pós-Graduação em Matemática (PROFMAT)) - Universidade Federal Rural de Pernambuco, Recife.

ROCHA, ANDERSON; DORINI, LEYZA BALDO. Algoritmos gulosos: definições e aplicações.

SILVEIRA, C. M. D. Análise de Métodos Heurísticos de Características Gulosa. Dissertação de Mestrado, Universidade Federal do Rio Grande do Sul, Instituto de Informática, Porto Alegre, 1999.

SORROCHE, R. LOPES, M. C. Análise comparativa de algoritmos de grafos para um sistema de auxílio à matrícula de alunos. In: Seminco – Seminário de Computação, XII. Itajaí, 2003.

SZWARCFITER, JAYME LUIZ. Grafos e algoritmos computacionais. Rio de Janeiro: Campus, 1984

YAMAMOTO M.; CAMARA G.; LORENA L.A.N. Tabu search heuristic for point-feature cartographic label placement. GeoInformatica and International Journal on Advances of Computer Science for Geographic Information Systems, Kluwer Academic Publisher, Netherlands, v. 6, n. 1, p. 77-90, 2002.

YAMAMOTO M.; LORENA L.A.N. A constructive genetic approach to pointfeature cartographic label placement. In: IBARAKI, T., NONOBE, K.; YAGIURA, M. (Ed.). Metaheuristics: progress as real problem solvers. Kluwer Academic Publishers, 2005. p. 285-300