ECT2303 – Linguagem de Programação

Avaliação Unidade III

1 Introdução

Todos os anos, o Ministério da Educação e Cultura (MEC) e o INEP (Instituto Nacional de Estudos e Pesquisas) realizam o Exame Nacional do Ensino Médio (ENEM). Neste ano de 2015, 7,7 milhões de candidatos se inscreveram. Muitas vezes, no entanto, não percebemos o quão complexa pode ser a organização deste evento. Por exemplo, você já imaginou como é feita a definição do local das provas?

Se olharmos para este problema de uma maneira macro, podemos verificar que há duas decisões a serem tomadas:

- De todos os locais possíveis para a aplicação da prova, quais devem ser utilizados?
- Conhecidos os locais de prova, em qual destes cada um dos candidatos deverá prestar exame?

Em um mundo perfeito, cada candidato realizaria a prova no local mais próximo de sua residência. Porém, nem sempre este local tem capacidade para atender a todos que moram nas proximidades, ou sequer pode ser usado para a realização da prova dados os custos necessários à sua utilização.

De forma a ilustrar melhor este problema, vamos analisar um exemplo fictício com tamanho reduzido. Para tal, considere a Figura 1. Nesta figura, os elementos azuis representam possíveis locais para a realização de prova. Porém, há dinheiro suficiente para que apenas 3 locais sejam utilizados. Nesta figura, há também 10 quadrados vermelhos, cada um representando um candidato inscrito no exame. ***Em qual local cada candidato deverá prestar o exame de forma que o tempo de deslocamento de todos os candidatos em relação ao local de prova, somados, seja mínimo?***

Considere as informações apresentadas nas Tabelas 1 e 2 para responder a esta pergunta. A Tabela 1 indica qual a capacidade de cada local. Isto é, quantos candidatos podem utilizá-lo simultaneamente. A Tabela 2, por sua vez, mostra qual o tempo médio de deslocamento de cada um dos candidatos para cada um dos possíveis locais.

Se você tentou encontrar uma solução para este problema, deve ter notado que esta tarefa não é simples. Isto porque, como dito no início deste texto, há dois problemas envolvidos:

1. Quais locais devem ser utilizados.
2. Em qual local cada candidato realizará a prova.

Na verdade, mesmo para este exemplo, que é reduzido, resolver estes dois problemas é algo não tão trivial quanto parece, pois há 9.330 formas de alocar os candidatos aos 3 locais de prova, caso ignoremos a restrição de capacidade e consideremos os 3 locais como conhecidos.

Considerando-se este número de soluções, poderíamos então pensar o seguinte: como aprendemos a programar, poderíamos utilizar um computador que enumerasse todas as 9.330 possibilidades de solução, escolhendo aquela que é mais adequada.

Figura 1: Cenário para decisão

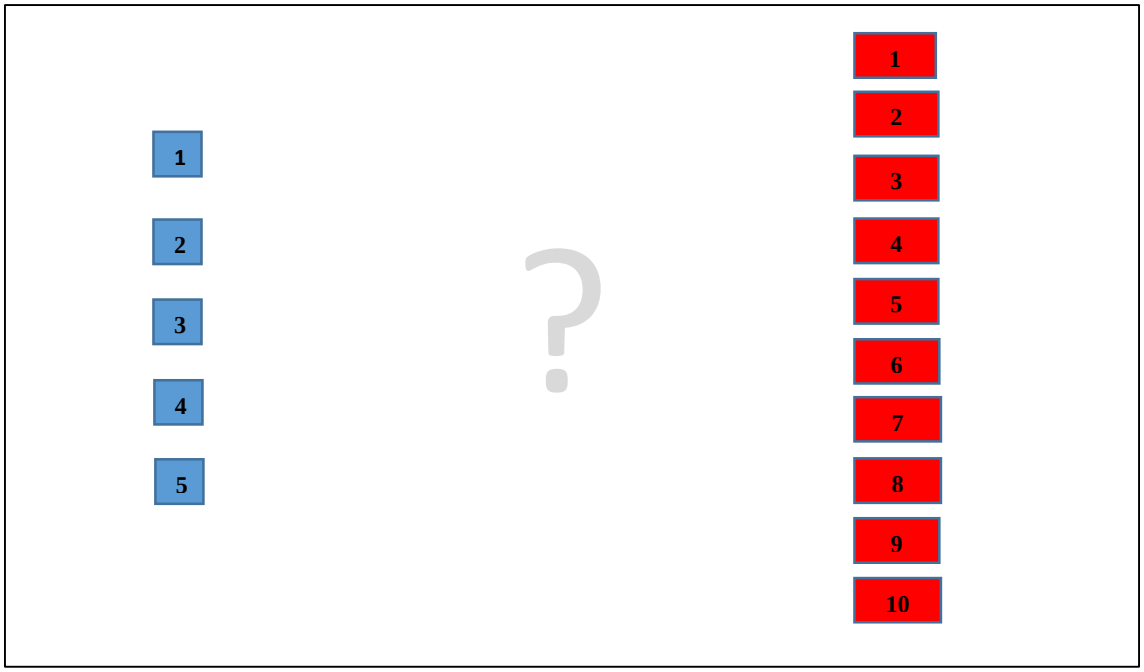


Tabela 1: Capacidade de cada local

Local 1	Local 2	Local 3	Local 4	Local 5
5	5	4	5	6

Tabela 2: Tempo médio de deslocamento de cada candidato em relação a cada local de prova

	Local 1	Local 2	Local 3	Local 4	Local 5
Candidato 1	40	41	47	54	26
Candidato 2	17	32	45	14	18
Candidato 3	31	45	31	27	41
Candidato 4	16	23	22	34	37
Candidato 5	35	17	28	22	40
Candidato 6	34	7	27	13	24
Candidato 7	22	48	11	16	47
Candidato 8	34	28	26	24	39
Candidato 9	42	53	29	20	25
Candidato 10	18	31	46	35	11

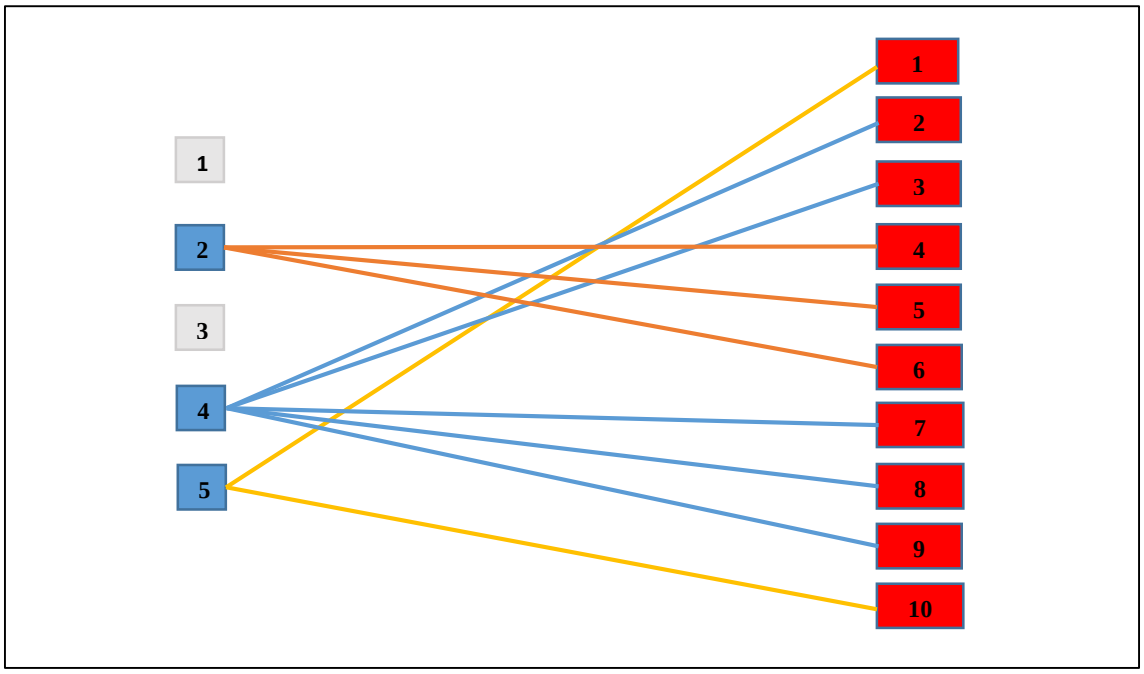
Isto funcionaria?

A resposta a esta pergunta pode ser a seguinte: “depende”. Para o caso mostrado, as 9.330 combinações seriam testadas rapidamente, mesmo que utilizássemos um simples *notebook* para isso. Porém, imagine agora que é necessário alocar 100 candidatos a 3 locais de prova já conhecidos: há aproximadamente 10^{68} formas de se fazer isso. Um computador, mesmo com alta capacidade de processamento levaria anos, ou mesmo séculos, para resolver este problema pelo processo de enumeração.

A esta altura, você já deve estar intrigado (ou não ☺), se perguntando como é que a organização do exame lida com esse problema. Como dito, só este ano, o exame teve 7,7 milhões de inscrições, com provas realizadas em milhares de lugares por todo o Brasil. Pensar no total de combinações possíveis entre candidatos e locais de prova, para este caso, é algo quase surreal.

Vamos agora voltar ao nosso exemplo, dado na Figura 1. Para este exemplo, a Figura 2 mostra a melhor solução existente para este problema, também chamada de **solução ótima**. Para esta solução, se somarmos o tempo médio de deslocamento de todos os candidatos em relação ao local ao qual foram designados, obteremos o valor 185. Chamamos este valor de **custo da solução**. Como esta solução é a solução "ótima", sabe-se que não existe outra solução para este problema com um custo menor. Isto é, não existe outra solução cuja soma dos tempos médios de deslocamento seja inferior a 185. Pode haver, no entanto, outra solução de mesmo custo.

Figura 2: Solução ótima para o exemplo fictício



Está curioso sobre como esta solução foi obtida? Os próximos capítulos deste texto lhe ajudarão a entender um pouco melhor sobre como podemos aplicar o conhecimento que adquirimos na disciplina de Linguagem de Programação para chegarmos a esta mesma resposta.

2 Resolução de Problemas de Otimização Combinatória

O problema mostrado no capítulo introdutório deste texto é conhecido na literatura como o problema das p-medianas capacitado (PMC). Neste problema, há m possíveis localizações para um serviço (facilidades) e n pontos de demanda (clientes). A solução do problema consiste em encontrar p facilidades que atendam à demanda dos clientes, considerando-se minimizar algum critério de custo e respeitando a capacidade de atendimento destas facilidades.

Para o nosso caso, o ENEM, as facilidades são os locais de prova e os clientes são os candidatos do exame.

O problema das p-medianas é classificado como NP-difícil. Isto significa que não se conhece um algoritmo de tempo polinomial que o resolva de forma **ótima.** Em outras palavras, pode-se dizer que, dependendo do tamanho da instância considerada (número de facilidades e clientes), é inviável utilizar algoritmos que forneçam a resposta ótima deste problema.

Um exemplo de algoritmo que resolve este problema de forma **exata** (ótima) é o algoritmo de *branch-and-bound*. Este algoritmo foi utilizado para resolver o exemplo utilizado no Capítulo 1. O software IBM ILOG CPLEX possui uma implementação deste algoritmo que pode ser utilizado por meio de diversas linguagens de programação, como por exemplo, C, C++, Java e Phyton, além de um ambiente gráfico com linguagem própria (OPL). Linguagens de modelagem como GAMS e AMPL também são suportadas. Para utilizar estas ferramentas, é necessário que se tenha um conhecimento mínimo de programação, assim como conhecimento matemático para formular um problema de otimização. O CPLEX é fornecido gratuitamente para fins de pesquisa.

Mas, como dito anteriormente, o problema das p-medianas é NP-difícil, e isto quer dizer que nem sempre algoritmos exatos serão capazes de resolvê-lo em um tempo aceitável. Isto mesmo! Nem o poderoso CPLEX da IBM dá conta de resolver certos problemas ou certas instâncias.

Sabendo disto, que que nem sempre é possível se obter a resposta ótima para um problema em um tempo aceitável, muitas vezes os tomadores de decisão se contentam em obter uma solução de boa qualidade para o problema. Esta solução não necessariamente é a ótima.

Para obter estas soluções de boa qualidade, pesquisadores e profissionais do mercado recorrem a uma família de métodos chamados de heurísticas. Uma definição possível para um método heurístico é a seguinte: uma heurística é um algoritmo que fornece uma solução para um problema. Note que esta definição é muito ampla. Sendo assim, qualquer ideia que gere uma solução válida para o problema pode ser classificada como uma heurística.

Sua vez

Sabendo o que é uma heurística, tente imaginar alguma que forneça uma solução para o problema apresentado. Gaste ao menos 5 minutos pensando em um modo de resolver o problema antes de prosseguir.

Talvez você tenha pensado em:

- Gerar soluções de forma completamente aleatória;
- Escolher aleatoriamente três locais de prova cuja capacidade somada atenda a todos os candidatos. Então, você aloca cada um dos candidatos ao local mais próximo de sua residência. Se este local estiver com a capacidade esgotada, você aloca o candidato ao secundo local mais próximo de sua residência, e assim sucessivamente;

Se você pensou em qualquer outra forma de resolver o problema, parabéns. Você pensou em uma heurística. Lembre-se, uma heurística é um algoritmo qualquer que fornece uma solução para um problema.

Mas o que você deve saber é que, nem sempre uma heurística fornece uma solução de boa qualidade. Na verdade, a solução fornecida pode estar muito distante da solução ótima.

Considerando esta realidade, os pesquisadores conceberam uma família complementar de métodos, chamados de metaheurísticas.

Metaheurísticas são métodos que preveem a aplicação de estratégias de busca mais especializadas às heurísticas, permitindo que estas encontrem soluções de boa qualidade para o problema em questão. A partir do uso de metaheurísticas, ainda, a solução ótima de um problema pode ser encontrada. Porém, uma metaheurística não fornece a prova matemática de que a solução dada é a solução ótima, isto é, a melhor solução possível. Os métodos exatos, diferentemente, fornecem esta prova.

A vantagem em se utilizar metaheurísticas advém do fato de que na maioria das vezes a ideia por trás destes métodos é simples, assim como sua implementação. Há também uma enorme economia de tempo e recursos computacionais ao se utilizar estes métodos. Há, desta forma, a possibilidade de se obter soluções de boa qualidade para problemas complexos em muito menos tempo e com recursos computacionais muito mais limitados em relação aos métodos exatos.

Na literatura, são encontradas diversas heurísticas e metaheurísticas, como por exemplo:

- Colônia de formigas;
- Têmpera Simulada;
- Busca em Vizinhança Variada;
- Busca Tabu;
- Otimização por Nuvem de Partículas;
- Algoritmos Genéticos;
- Outros;

Muitas das ideias por trás da forma como estes métodos encontram soluções para os problemas de otimização são baseadas em observações de fenômenos físicos ou naturais. Por exemplo, Algoritmos Genéticos se baseiam na Teoria da Evolução das Espécies, especialmente o mecanismo de seleção natural. Colônia de Formigas, por sua vez, é baseada na forma como as formigas encontrar alimento. Já algoritmos baseados em Têmpera Simulada, são baseados no processo de resfriamento de metais. Caso tenha ficado curioso, faça uma pesquisa sobre estes métodos.

Outra metaheurística conhecida na literatura é a **Busca Local Iterada (BLI)**, mais conhecida do inglês *Iterated Local Search - ILS*. O Capítulo a seguir descreverá esta abordagem, a qual utilizaremos em nosso trabalho.

3 Busca Local Iterada

Para descrever o algoritmo BLI são necessários poucos elementos. Porém, para cada problema a ser resolvido, estes elementos devem ser adaptados. Isto é, a ideia por trás do método é genérica, porém seus elementos precisam ser customizados para o problema que se quer tratar. Veja o Algoritmo 1, no qual a BLI é apresentada em sua forma mais genérica. Para o algoritmo apresentado, temos as seguintes subrotinas:

1. *gera_solução_inicial*: obter uma solução inicial par ao problema. Para este passo, qualquer heurística que gere uma solução válida pode ser usada;
2. *busca_local*: consiste em, a partir de uma solução existente, obter uma solução de melhor custo utilizando alguma estratégia de movimentação;

3. *perturbação*: a partir de uma solução existente, o processo de perturbação deverá gerar uma nova solução. Preferencialmente, este processo deve ser feito com base em algum grau de aleatoriedade.

A BLI basicamente repete estes três procedimentos enquanto algum critério de parada não for satisfeito. Este critério, por exemplo, pode ser o tempo total de execução do algoritmo ou o número de iterações.

Algoritmo 1: Busca Local Iterada

1. s = gera_solucao_inicial();
2. s^* = busca_local(s);
3. Enquanto critério de parada não satisfeito
   3.1. s = perturba(s^*);
   3.2. s = busca_local(s);
   3.3. se custo de s melhor que custo de s^* então
      3.3.1. $s^* = s$;
   3.4. Fim Se;
4. Fim Equanto

No próximo Capítulo deste texto será dado o roteiro de desenvolvimento deste projeto, no qual cada elemento do BLI é formulado em maiores detalhes para a resolução do problema das p-medianas, assim como outros elementos do programa que você deverá apresentar como solução. Para facilitar sua tarefa e maximizar sua nota, siga este roteiro.

4 Um algoritmo BLI para o problema das p-medianas: caso ENEM

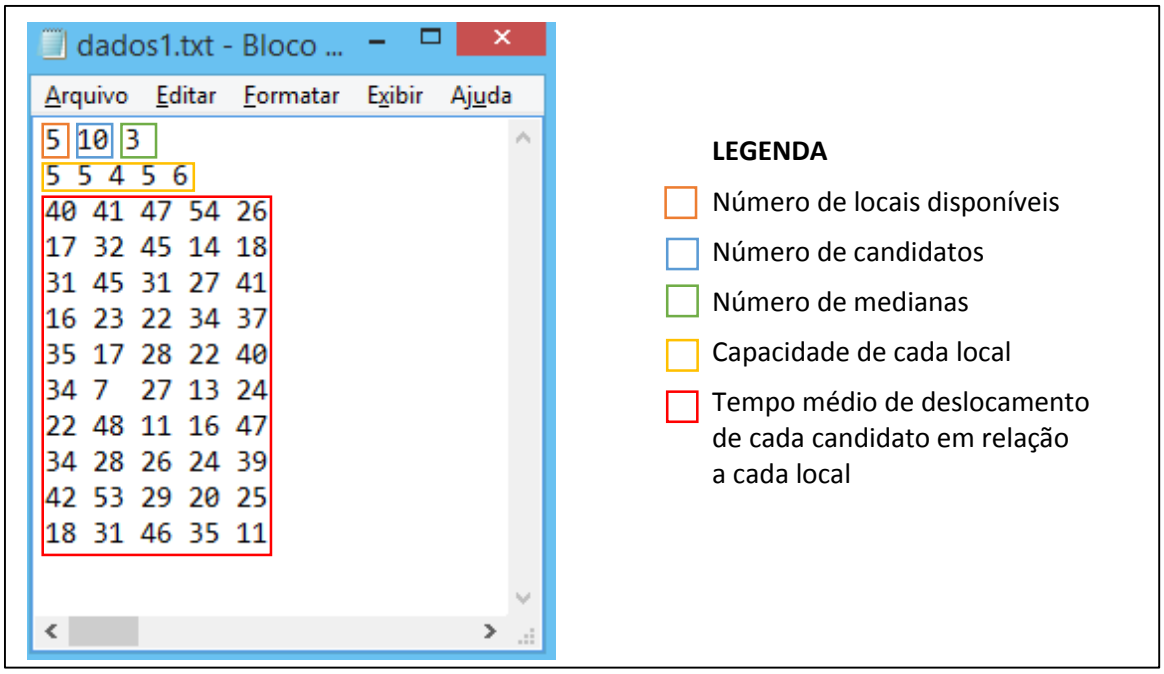
Passo 1: adquirindo os dados de entrada

Os dados que você deverá considerar como instância de problema a ser resolvido estão armazenados em arquivos de texto (extensão txt). Desta forma, você deverá criar uma função que receba como parâmetro o nome do arquivo a ser aberto e leia todos os dados nele contidos, armazenando-os em um tipo estruturado criado por você. Sua estrutura necessitará dos seguintes campos:

- um campo para armazenar a quantidade de facilidades disponíveis (valor de m);
- um campo para armazenar a quantidade de clientes (valor de n);
- um campo para armazenar o número de facilidades que devem ser utilizadas na solução (valor de p).
- um campo para armazenar a capacidade de cada uma das facilidades. Isto é, a quantidade máxima de candidatos que podem fazer a prova em cada local;
- um campo para armazenar o tempo médio de deslocamento de cada candidato em relação a cada local de prova;

É importante que você analise cuidadosamente qual será o tipo de cada um destes campos para que erros não ocorram. Os arquivos de dados a serem lidos estão no formato mostrado na Figura 3.

Figura 3: Formato do arquivo de dados



Neste arquivo, o primeiro valor encontrado é um inteiro, que representa o número total de locais de prova disponíveis (valor de m). O segundo valor corresponde ao número total de candidatos (valor de n). O terceiro valor corresponde a quantos locais de prova podem ser utilizados (valor de p). A segunda linha do arquivo contém um *array* com m valores, um para cada local disponível de prova, representando sua capacidade de atendimento. Por fim, da terceira linha em diante, há um $array$ bidimensional, para o qual cada linha denota o tempo de deslocamento de cada um dos n candidatos em relação aos m locas de prova.

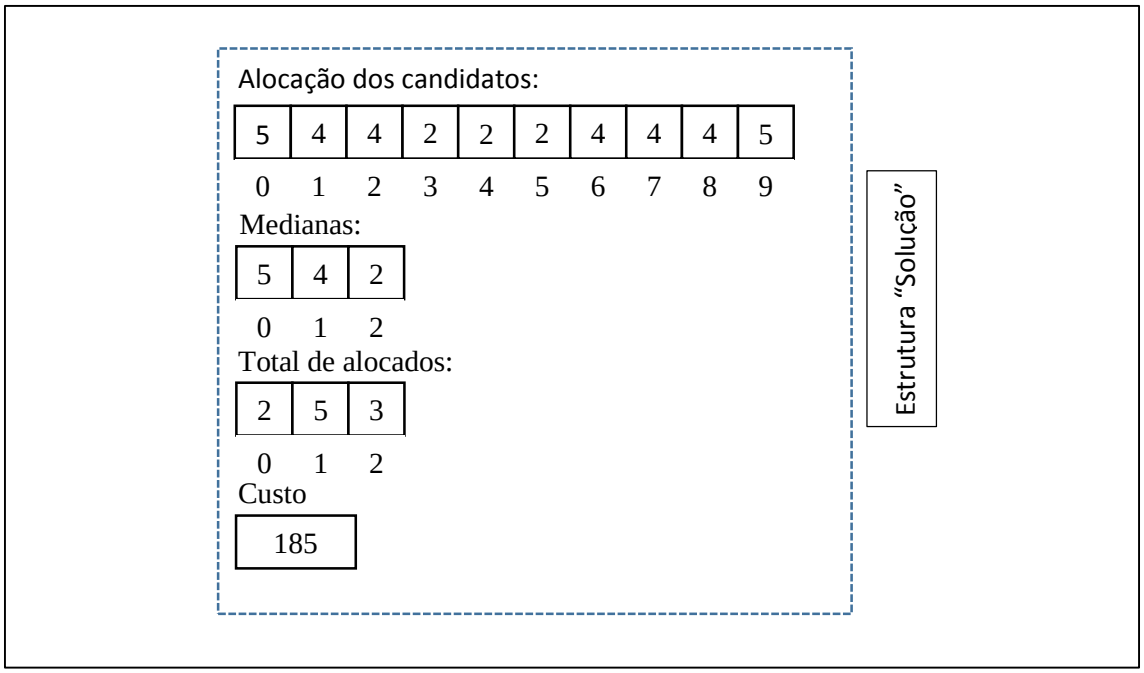
Passo 2: representando uma solução

É muito importante que uma estrutura de dados adequada seja definida para que você possa codificar seu algoritmo de forma concisa e prática. É sugerido que você crie um tipo estruturado para isto. Este tipo estruturado pode conter os seguintes campos:

- um *array* para representar a alocação dos candidatos em relação a cada local de prova;
- um *array* que representa quais são as medianas atualmente utilizadas;
- um array que representa a contagem de quantos candidatos estão alocados a cada mediana;
- um campo para armazenar o custo da solução. Lembre-se, para o nosso problema o custo corresponde ao tempo de deslocamento de todos os candidatos somados, considerando a alocação destes às medianas;

Para facilitar a compreensão desta estrutura, considere o exemplo apresentado na Figura 4, no qual os campos da estrutura contêm os valores da solução apresentada na Figura 2.

Figura 4: Representação de uma solução em uma estrutura



Passo 3: Obtendo uma solução inicial para o problema

Para resolver este passo, você deverá criar uma função que gere uma solução inicial para o problema. Um algoritmo possível para isso seria: escolha p locais de prova de forma aleatória, de modo que a capacidade destes locais somados seja suficiente para atender ao número total de candidatos. Após isto, aloque os candidatos a estes três locais. Defina um critério de alocação, como por exemplo:

- Aloque o candidato ao local mais próximo dele. Se o local já estiver cheio, aloque-o ao segundo local mais próximo, e assim sucessivamente;
- Aloque os candidatos de forma aleatória, respeitando a capacidade dos locais escolhidos;

Passo 4: Avaliando a solução gerada

É interessante que você tenha uma função que recebe como parâmetro a estrutura que representa a solução e calcula qual o custo correspondente a esta solução, atualizando o valor do respectivo campo. Lembre-se, para o problema que estamos tratando, o custo corresponde à soma dos tempos de deslocamento de cada candidato em relação ao local de prova ao qual este foi designado.

Passo 5: Perturbando uma solução

Neste passo, você deverá criar uma função que gera uma perturbação em uma solução. Uma perturbação nada mais é do que introduzir alguma modificação à solução que você tem em dado momento.

De forma a exemplificar esta operação, suponha que a solução atual tenha como medianas os locais 5, 4 e 2. Poderíamos escolher um destes locais aleatoriamente para deixar a solução, substituindo-o por outro local escolhido aleatoriamente. Feito isso, refazemos a alocação dos

clientes seguindo algum critério, como descrito no Passo 3. Obviamente a perturbação deverá considerar que os locais de prova escolhidos devem ter capacidade para atender a todos os candidatos.

Passo 6: busca local

A busca local consiste em realizar uma série de movimentações nos elementos da solução, buscando-se obter uma solução de melhor custo. Uma estratégia de busca local para o problema considerado poderia ser a seguinte: para cada par de candidatos, verificamos se a troca (permuta) de local de prova destes candidatos gera uma solução de melhor custo.

Imagine, por exemplo, que o candidato 1 faz prova no local 3 e o candidato 2 faz prova no local 4. Estas associações, como sabemos, tem um custo. Sendo assim, verificamos se associar o candidato 1 ao local de prova 4 e o associar o candidato 2 ao local 3 gera um custo menor. Se sim, fazemos esta alteração na solução. Para ser eficaz, a busca local deve exaurir todas as possibilidades. Isto é, verificar a possibilidade de troca de cada candidato com todos os demais.

Passo 7: salvando a solução obtida em um arquivo

Neste passo, você deve criar uma função que salve os valores da solução obtida em um arquivo, mostrando não só o custo da solução, mas também quais são as medianas e qual a alocação dos candidatos.

Passo 8: função *main*

Implemente a função *main* de um programa que execute a BLI durante um certo intervalo de tempo, informado pelo usuário (30 segundos, por exemplo). O usuário deverá informar também o nome do arquivo no qual estão contidos os dados do problema, bem como o nome do arquivo onde o resultado deverá ser salvo.

5 Critérios de avaliação

Para fins de pontuação máxima, seu programa deverá, no mínimo:

- Utilizar um tipo estruturado para representar uma solução;
- Utilizar um tipo estruturado para armazenar os dados lidos do arquivo de dados de entrada;
- Uma função que faz a leitura dos dados de entrada a partir do arquivo;
- Uma função para gerar uma solução inicial;
- Uma função que gera perturbações em uma solução;
- Uma função que faz a busca local em uma solução;
- Uma função que salva o resultado do algoritmo em um arquivo;
- A função *main*, que implementa a BLI a partir das demais funções criadas e a executa por uma certa quantidade de tempo, definida pelo usuário;

Além de estar de acordo com as seguintes definições:

- Não deverá utilizar variáveis ou constantes globais;
- O código deverá estar bem **indentado**. Veja o vídeo a seguir para um tutorial sobre este tópico: https://www.youtube.com/watch?v=840OgXEqWfU;
- O programa deverá compilar e executar sem erros. Se o código não compilar, o trabalho não será avaliado;
- O programa deverá obter uma solução para o problema, não necessariamente a ótima;

- O programa deverá, obrigatoriamente, pedir ao usuário os seguintes valores:
    - Nome do arquivo que contém os dados de entrada;
    - Tempo total, em segundos, a ser considerado para a execução do algoritmo de BLI;
    - Nome do arquivo de saída a ser gerado;

Este trabalho é individual. O código-fonte de seu programa deverá ser enviado dentro da data e horário previstos. Não haverá prorrogação de prazo, independentemente de eventuais problemas no SIGAA nos últimos minutos de envio. Caso o sistema esteja indisponível, o trabalho deverá ser enviado para o email santi.everton@gmail.com.