UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
DEPARTAMENTO DE INFORMÁTICA TEÓRICA
INF05010 Otimização Combinatória
Prof. Dr. Henrique Becker

Relatório

Nome: André Gustavo Dessoy Hübner **Matrícula:** 00315569
Nome: Emanuel Ruschel Rauter **Matrícula:** 00578247
Nome: Joana Alexia de Campos Vargas **Matrícula:** 00270178

Este trabalho aborda a aplicação da meta-heurística de Busca Local Iterada ao problema da Peregrinação.

Escolha de representação da solução

A solução do menor caminho foi representada através de um vetor de inteiros com os índices do templo, onde a ordem em que o índice do templo aparece indica quando o templo é visitado no caminho. Ex.: [1 3 2] -> (Começa no templo 1, vai para o templo 3 e por último vai pro templo 2).

Manter a solução num vetor facilita o processo de swap que é utilizado na busca local e é uma forma com pouco uso de memória (em relação a, por exemplo, uma lista de adjacências) e que facilita o cálculo de distância total.

Construção da solução inicial

A solução inicial é construída de maneira randomizada, construindo do zero um vetor de caminho tentando inserir em cada iteração um dos templos que esteja livre.

Primeiramente, um vetor de todos os templos é criado, e a partir de então, enquanto não restar nenhum templo válido livre, tenta-se inserir um possível templo no próximo espaço do caminho. Para a verificação de se um templo é possível, busca-se por todos os seus pré-requisitos; se algum deles ainda não estiver no caminho, este não é um templo válido. Apenas um destes templos é adicionado por vez, sendo o mesmo então filtrado do vetor de templos livres de forma que este eventualmente se esvazie.

Para o cálculo da distância inicial, fez-se o cálculo da distância cartesiana entre o ponto i do caminho para seu destino, ponto i+1 no caminho (multiplicando depois por 100 para considerar dois pontos após vírgula) e repetindo até somar o tamanho de todas as arestas.

Principais estruturas de dados

Vetores de inteiros: Armazenam os caminhos dos templos, sendo usados para guardar o melhor caminho obtido e também para guardar o caminho atual após busca local ou perturbação.

Vetor de coordenadas: Vetor composto por vetores que contém as coordenadas x e y de cada templo, é usado para calcular a distância entre 2 templos

Vetores auxiliares: Usados para armazenar os índices de todos os templos que são pré-requisitos de um certo templo. Coloca todos num mesmo vetor para criar um local para verificar os pré-requisitos de todos templos de uma instância

Vizinhança e a estratégia de escolha de vizinhos

O tipo de vizinhança escolhido é o swap de 2 vértices, onde invertemos a posição de 2 vértices dentro do caminho escolhido e calculamos de forma incremental a diferença entre a nova distância e a antiga distância.

A estratégia de escolha de vizinhos é se deslocar para o primeiro vizinho com distância do caminho menor até que se chegue ao ótimo local (não há vizinho melhor na vizinhança).

Processo de recombinação e factibilização

O processo de recombinação não ocorre nesta meta-heurística devido ao fato de a Busca Iterada não ser um algoritmo genético e utilizar apenas uma solução.

Como a perturbação é a troca aleatória de K templos gerando somente soluções válidas, o processo de factibilização não é necessário.

Parâmetros do método

O método recebe quatro parâmetros: O caminho para o arquivo que contém a instância, o número máximo de iterações permitidas, a seed de 1 a 5 que vai ser utilizada para padronizar randomização e, por último recebe o parâmetro K que define quantos vértices a perturbação tentará realocar. Este último possui um valor default de 2 para o caso em que não seja informado nenhum valor.

Critério de parada

O critério de parada adotado na implementação da Busca Local Iterada para o problema da Peregrinação é por número de iterações, encerrando o processo quando se atinge o número máximo de iterações definido na chamada do programa.

Execuções das instâncias

Para todos os testes, as medições de tempo estão em segundos.

Instance	Iteração	Seed	K	Distância final
01.txt	52309	1	2	25500
02.txt	17409	1	5	19700
03.txt	100	2	50	1295900
04.txt	100	3	50	1746700
05.txt	100000	5	20	16831
06.txt	100000	5	3	893285
07.txt	1000	1	2	3320
08.txt	725	1	2	451939
09.txt	78900	4	2	81388
10.txt	100	1	1	2889660

Tabela 1 - Iterações em meta-heurística

Comparações preliminares

CPU: 13th Gen Intel(R) Core(TM) i7-13650HX (20) @ 4.90 GHz
RAM: 15.33 GiB
SO: Fedora Linux 42 (Workstation Edition) x86_64
Instância 10.txt (400 vértices) - Seed 1

Iterações	Tempo	K	Distância inicial	Distância final
10000	59.28	1	16631961	2009750
100000	588.31	1	16631961	1970622
10000	111.84	2	16631961	1995617
100000	1113.43	2	16631961	1994425
10000	205.24	4	16631961	1990562
100000	2061.2	4	16631961	1990562

Tabela 2 - Iterações sobre instância 10

Levando em conta estas execuções aprofundadas sobre a instância 10 é possível ver que o incremento de K implicou um aumento importante no tempo decorrido, bem próximo inclusive do esperado, pois a complexidade da perturbação é dependente de K e um aumento de 1 para 4 ocasionou um tempo aproximadamente 4x maior também. Ou seja, K pode ser um gargalo importante no fator tempo.

Por outro lado, pelo menos para os casos de 10000 iterações, variando apenas o K para esta instância, apresentaram uma diminuição da distância final (função objetivo) conforme K aumentava, o que de fato seria esperado para um número relevante de iterações. Isto, no entanto, não se repetiu para os casos de 100000, demonstrando por outro lado que não necessariamente esta variação da distância final é decorrente de K, até porque a alteração não é tão significativa.

De qualquer forma, não se espera que perturbações sempre ajudem uma instância a escapar a um outro basin de atração mais favorável que o que cairia sem ela. Assim como nesta tabela, outras execuções experimentais também validaram a capacidade desta estratégia de melhorar a função objetivo em certos casos.

CPU: Intel(R) Core(TM) i7-6600U CPU @ 2.60GHz 2.81 GHz
RAM: 8,00 GB (utilizável: 7,88 GB)
SO: Windows 10 Pro Versão 22H2
Instância 01.txt (100 vértices) - Seed 1

Iterações	K	Distância final
623	2	31300
52309	2	25500
623	5	32900
52309	5	26700
623	10	32700
52309	10	25700

Tabela 3 - Iterações sobre instância 01

Comparando com a instância anterior, é possível ver algumas diferenças nas performances do programa. Por óbvio, aumentar o número de iterações melhora o resultado final, pois a meta-heurística tem mais tempo de percorrer o plano de soluções. Por outro lado, a menor perturbação obteve uma performance melhor na solução comparada às maiores, isso se dá ao fato da instância 01 ser bem menor que a instância 10 (a instância 01 tem 100 templos, enquanto a instância 10 tem 400) o que significa que em 10 o K com valor 2 é pequeno demais e perturbação não altera o suficiente a solução para fugir de ótimos locais, na instância 01 esse valor de K funciona bem para ajudá-lo a escapar desses ótimos locais.