INF1771 - Inteligência Artificial - (2018.2)

Professora: Renatha Capua

Relatório – Trabalho 1 – Minimum Latency Problem

Grupo:

Rodrigo Pumar

Bruno Pedrazza

1. Introdução

O problema MLP- Minimum Latency Problem – Problema de latência mínima de um percurso. É um variante do problema do cacheiro viajante, em que ao invés de tentar minimizar a distância percorrida no percurso, minimizamos a latência total dos nós visitados, ou seja, minimizamos o tempo de espera dos nós clientes.

2. Implementação e entrega

Foi implementado em JAVA e em pair programming em sua plenitude.

A entrega foi feita zipando o projeto do eclipse.

O zip entregado, deve ser extraído e aberto o projeto pelo eclipse. O projeto já possui as instancias nos locais certos para a execução do código fonte.

3. Metodologia

Para solução do problema, foi implementado dois algoritmos:

3.1. Dijkstra's adaptado para contabilizar latência

Foi implementado um algoritmo guloso que é uma implementação do algoritmo Dijkstra's com a adaptação de guardar a latência de cada nó visitado, durante sua visita.

Pela natureza do algoritmo guloso, o percurso guloso foi sempre o mesmo devido a previsibilidade de andar sempre para o nó mais próximo.

3.2. Algoritmo Genético com PMX crossover e mutação de troca

A metaheurística selecionada foi um algoritmo genético, que implementa como método de recombinação Crossover Mapeado Parcialmente (PMX) . Foi usado um vetor de reposicionamento para lidar com as colisões dos genes fora do segmento cortado durante o PMX.

Depois da recombinação foi aplicada mutações de com probabilidades variadas dependendo em qual geração estamos, com mais probabilidade nas ultimas gerações. As mutações eram do tipo de troca de genes, com o número de genes trocados variando aleatoriamente também e dependendo do tamanho do caminho.

A primeira geração contem caminhos aleatórios acrescentado, porem foi acrescentado o caminho da solução gulosa, visto que sabemos que ela é comparativamente melhor que os primeiros indivíduos aleatórios, para que comecemos com pelo menos um pai bom.

A função heurística considerada foi a própria latência total do caminho, e usamos o paradigma de que não sabíamos o ótimo/melhor latência alcançável (OPT/BKS), pois normalmente em problemas reais não saberíamos e por isso não usamos esse OPT/BKS na função heurística.

A seleção dos pais que iriam procriar foi implementada ordenando os melhores pais, e sempre pegando os X melhores pais, porem como foi aprendido em aula que é bom ter pais não necessariamente bons, foi adicionado a listagem de pais Y filhos com índice que não estavam entre esses melhores pais, assim adicionando sempre um numero fixo de filhos não ótimos e assim mantendo a variabilidade nos indivíduos em cada geração de maneira bem simples.

4. Resultados

Instancia	BKS/OPT		Nossa Solução Algoritmo Genético (40 gerações)		Nosso Guloso Dijkstra's	
	Tempo de execução (ms) (paper)	Latência Total	Tempo de execução (ms)	Latência Total	Tempo de execução (ms)	Latência Total
brazil58	550	512361	308	592365.0	3.86	600705.0
dantzig42	170	12528	98	12967.0	0.28	13514.0
gr120	9540	363454	190	389187.0	1.26	390569.0
gr48	310	102378	106	113992.0	0.23	113992.0
pa561	1155320	658870	779	779036.0	15.99	779400.0

O nosso algoritmo genético teve melhoras significativas contra o guloso, porem mesmo com o aumento do número de gerações, não conseguimos nos mover mais perto do próximo do BKS/OPT que justificasse o aumento do tempo de execução. Não sabemos se o algoritmo funciona perfeitamente para sair de máximos locais, visto não fizemos estatística da geração de filhos.

5. Dificuldades

Tentamos adaptar o algoritmo guloso, para que previsse X nós adiante e verifica-se se o caminho guloso ainda era o melhor e mudasse caso contrário, mas devido a dificuldades de implementação e a limitação de tempo, abandonamos essa tentativa, visto que era necessário a implementação do genético ainda.

6. Conclusão

A implementação do algoritmo genético foi bastante iterativa, usando conceitos aprendidos em aula. A implementação deu bastante margem para criatividade e iteração numérica nos valores para maximizar a eficiência do resultado.

Em problemas reais em que não se sabe qual o melhor possível, acreditamos que essa analise da performance e corretude do algoritmo se tornam bem mais complicados, pois ajudou muito saber qual o máximo ótimo, como parâmetro para guiar e ajustar o código e os parâmetros.