Talita Halboth Cunha Fernandes
GRR20165399
talitahalboth@gmail.com

Trabalho Prático 1
Inteliência Artificial 2020/2021

4 de janeiro de 2021

1 A*

1.1 Algoritmo

O algoritmo A* é relativamente simples. Uma fila de prioridades que ordena com relação ao custo do caminho atual (g) + custo da heurística (h) é inicializada contendo apenas um nó como estado inicial do problema. A primeira coisa é testar se o nó inicial é o nó objetivo, em caso positivo, retornamos este nó. Então, enquanto a fila não está vazia:

1. Retira-se o primeiro nó *cur* da fila de prioridade
2. Criamos um array com todos os vizinhos de *cur*.
3. Para cada vizinho de *cur* :
   (a) Encontramos o custo *tentative* do início até o vizinho atual.
   (b) Calculamos o custo heuristico a partir do vizinho atual até o estado final.
   (c) Criamos um novo nó com esses valores e o estado do vizinho
      i. Se este nó é o nó de objetivo, retorna o caminho encontrado por backtracking.
      i. Inserimos o novo nó na fila de prioridade.

2 Algoritmo Genético

2.1 Estrutura do Algoritmo Genético

O algoritmo genético inicialmente gera uma população aleatória, e, a partir daí, gera novas gerações a partir da geração anteior até que uma das condições a seguir seja alcançada:

- O número de gerações chegue a 2000
- Ocorra convergência do melhor valor de *fitness* encontrado

2.2 Nova Geração

A nova geração é gerada da seguinte forma: 2 pais são escolhidos da população atual. É realizada operação de crossover entre os pais, de forma que eles gerem 2 filhos. Cada um dos filhos pode ou não sofrer uma mutação. Estes passos são repetidos $\mathcal{N}$ vezes, sendo $\mathcal{N}$ o número de indivíduos da população. Por fim, são escolhidos $\mathcal{N}$ indivíduos dentre os filhos e os pais para compor a nova população.

2.2.1 Seleção de Pais

A seleção de pais é feita a partir de torneios. São escolhidos $\mathcal{K} = (numero\ de\ individuos\ na\ populaco)/10$ indivíduos da população aleatoriamente (os competidores) e são realizados ”torneios”entre eles. Torneios são realizados escolhendo-se dois indivíduos dentre os competidores, e o indivíduo com o maior fitness ”vence”o torneio e continua entre os competidores, enquanto o outro é removido do grupo. Isto é repetido até sobrar apensar um indivíduo entre os competidores, que é escolhido como o próximo pai.

2.2.2 Crossover

O crossover é feito usando-se o algoritmo Edge Assembly Crossover (EAX), descrito no arquivo suplementar incluído no trabalho. Inicialmente são gerados Ciclos AB a partir dos pais A e B. Cada ciclo encontrado é selecionado com probabilidade 0.5 para compor um grupo D. A partir de D é gerada uma solução intermediária. Esta solução intermediária é separada em ciclos, o que foi alcançado usando-se um *Disjoint Set* e um algoritmo de *Union Find*. Por fim, é realizada a mutação da solução intermediária, unindo-se cada ciclo encontrado, até chegar a um grafo contendo um único ciclo.

2.2.3 Mutação

A operação de mutação usada é uma simples troca de 2 elementos da permutação aleatoriamente, baseando-se em um valor de *mutation rate*.

2.2.4 Seleção da próxima geração

A seleção da próxima geração é baseada na aptidão de cada indivíduo. Os $\mathcal{N}$ indivíduos com menor custo de caminho são selecionados para compor a próxima população.

2.3 Escolha de Operadores

Foram codificados vários operadores para cada passo da geração de novos indivíduos. Eles são descritos a seguir. Todos os operadores que não estão na versão final podem ser encontrados no arquivo ”utils.py”.

2.3.1 Seleção de Pais

Inicialmente, havia 2 operadores para a seleção de pais. O operador 0 realiza seleção proporcional baseado na aptidão de cada indivíduo: indivíduos com maior *fitness* têm mais chance de ser selecionados. O operador 1 é o de seleção por torneios, que é o operador usado na versão final.

2.3.2 Crossover

Inicialmente, havia 3 operadores de crossover. O operador 0 simplesmente copiava a primeira metade de um dos pais para o filho, e a metade seguinte era montada a partir da ordem das cidades no outro pai.

O operador 2 é o operador pmx (Partially Mapped Crossover) descrito no arquivo suplementar fornecido, e o operador 3 é o EAX, utilizado na vesão final do algoritmo.

2.3.3 Nova Geração

Inicialmente, codifiquei 3 operadores para seleção da nova geração. O operador 0 é o operador de seleção baseado na aptidão, que é o operador usado na versão final. O operador 1 simplesmente escolhia os primeiros $\mathcal{N}$ gerados e retornava. O operador 2 realizada seleção proporcional baseado no *fitness* de cada indivíduo: indivíduos com maior *fitness* têm mais chance de ser selecionados.

2.3.4 Mutação

Foram feitos testes com *mutation rate* de 0.0, 0.01, 0.1, 0.2 e 0.5. O melhor resultado encontrado foi com *mutation rate* de 0.01.

2.3.5 Avaliação de Operadores

Para avaliar quais os operadores que geram os melhores resultados, foram realizados testes entre todas as combinações (Exceto as variações do *mutation rate*), rodando cada combinação cinco vezes. Os resultados são apresentados na Figura 1. O eixo X representa cada instância testada, e o eixo Y representa a porcentagem da diferença do custo do caminho obtido em relação ao custo ótimo para cada instância $(((obtido - ótimo)/ótimo) * 100)$. As labels de cada curva representam os parâmetros utilizados, sendo que o primeiro é a escolha de seleção de pais, o segundo é o operador de crossover e o terceiro é o operador de escolha da próxima geração. O gráfico está ordenado em relação à média da porcentagens de diferença entre o valor ótimo e o valor obtido pelo algoritmo. Podemos ver claramente que o operador de crossover 2 (EAX) obteve os melhores resultados. Também é possivel notal que o operador de seleção de nova geração 0 (baseado na aptidão) obteve resultados superiores aos outros. Isso fica mais visível ao calcular a média das % de diferenças para cara Operador. No caso do operados de crossover, as médias são apresentadas na tabela 1. Já em relação ao operador de escolha da próxima geração, os resultados são mostrados na tabela 2. Os operadores de seleção de pais não tiveram grandes diferenças entre si. A escolha do *mutation rate* foi feita realizando-se testes utilizando seleção de pais por torneio, crossover por EAX e seleção de próxima geração por apditão. A taxa utilizada na versão final do algoritmo foi de 0.01. Os resultados da diferença em % para o ótimo foram muito próximos. Desta forma escolheu-se essa por ela ser próxima do resultado ótimo, mas com um tempo mais rápido que outras taxas.

2.4 Critério de Convergência

Foi adotado um critério de convergência para parar a computação em casos que o algoritmo não consegue encontrar permutações com melhor fitness depois de muitas gerações. Isto é feito para casos em que, por exemplo, o algoritmo alcance o custo ótimo e continue procurando. Esse critério é útil principalmente por que o operador de crossover (EAX) tem uma alta complexidade, na faixa de $O(\mathcal{N}^2)$, sendo $\mathcal{N}$ o tamanho da permutação (número de cidades na entrada), mas ele consegue encontrar indivíduos com custo muito próximo de ótimo rapidamente.
O cálculo realizado para checagem de convergência compara o número da geração atual com a geração onde o melhor *fitness* encontrado mudou pela última vez, multiplicado por 5. Assim, cada vez que o melhor *fitness* é atualizado, o algoritmo terá mais 4 x esse "tempo"para encontrar um *fitness* melhor. O critério foi escolhido observando-se a frequência média em que o melhor *fitness* é encontrando.

3 Avaliação e comparação de cada método

3.1 Avaliação

3.1.1 A*

O algoritmo A* sempre encontra o caminho ótimo, porém ele demanda muita memória e computação, fazendo com que seja inviável sua utilização em casos muito grandes. Por ser ótimo, não faz sentido avaliar o custo que o algoritmo encontrou. Dessa forma, a informação mais relevante é o tempo que o algoritmo levou para cada instância. Estes valores podem ser encontrados na tabela

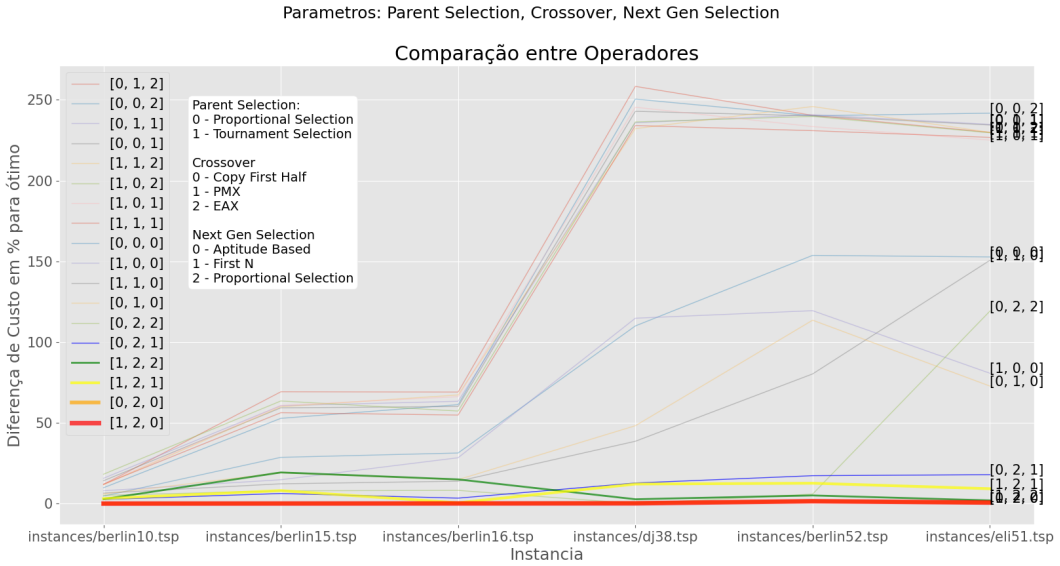
3.1.2 Algoritmo Genético

O algoritmo genético é uma busca heurística, portanto, não há certeza de que se alcançará um resultado ótimo. Desta forma, é importante avaliar tanto o restuldado encontrado quanto o tempo necessário para realizar os cálculos. A figura mostra a diferença em % da média encontrada para cada instância, rodando a instância 10 vezes, e a figura mostra a média de tempo necessário para rodar cada instância.

3.2 Comparação

Ambos os algoritmos podem ser usados para resolver o problema do caixeiro viajante, mas eles são relevantes em situações diferentes. Para casos muito grandes, o A* pode não ser ideal, por necessitar de muita memória e muito tempo para fazer as computações. Neste caso, o algoritmo genético é uma solução melhor. Apesar de nem sempre encontrar o melhor resultado, em geral ele chega bastante perto mesmo para problemas maiores em um tempo razoável, o que é suficiente para algumas aplicações.

Figura 1: Comparação da porcentagem de erro entre resultados obtidos e resultados ótimos para todos os parametros.



Operador	Média
0	117.82
1	110.21
2	8.43

Tabela 1: Operadores de Crossover

Operador	Média
0	39.67
1	96.25
2	100.53

Tabela 2: Operadores de Seleção da Próxima Geração

Instância	Tempo
TSP_berlin10	0.14 s
TSP_berlin15	7.77 s
TSP_berlin16	34.17 s

Tabela 3: Tempo gasto pelo algoritmo A* para rodar cada instância

Figura 2: Comparação da porcentagem de erro entre resultados obtidos e resultados ótimos

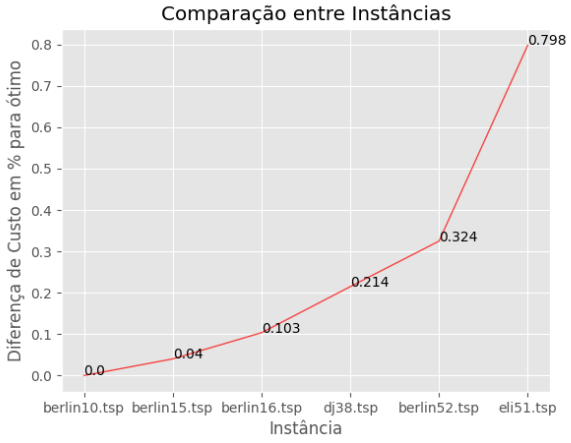


Figura 3: Comparação de média tempo