Análise Comparativa de Configurações de um Algoritmo Genético para o Problema do Caixeiro Viajante

Gabriel Ast dos Santos[1]

[1] Universidade Tuiuti do Paraná (UTP)
Curitiba – PR – Brazil

gabriel.santos12@tuiuti.edu.br

Resumo. *Este trabalho investiga o impacto de diferentes configurações de parâmetros na performance de um Algoritmo Genético (AG) aplicado ao Problema do Caixeiro Viajante (TSP). O TSP, um clássico problema de otimização combinatória NP-difícil, busca a rota de menor custo que visita um conjunto de cidades. A eficiência dos algoritmos genéticos para resolver o TSP é altamente dependente de seus parâmetros, como a taxa de mutação, o método de crossover, o número de gerações e a estratégia de seleção. A metodologia envolve a execução sistemática do AG com variações nesses parâmetros para instâncias fixas do TSP, avaliando como cada configuração afeta a qualidade da solução final (menor distância) e o tempo de convergência. Este estudo visa fornecer diretrizes sobre como a otimização de hiperparâmetros pode aprimorar a eficácia de abordagens evolutivas para problemas de otimização complexos.*

1. Introdução

O Problema do Caixeiro Viajante (TSP) é um dos desafios mais emblemáticos da otimização combinatória. Dada uma lista de cidades e as distâncias entre cada par, o objetivo é encontrar a rota mais curta possível que visite cada cidade exatamente uma vez e retorne à cidade de origem. A sua complexidade, que cresce fatorialmente com o número de cidades ($O(n!)$), torna a busca por soluções ótimas através de métodos exaustivos computacionalmente inviável para instâncias de tamanho moderado. Nesse contexto, a Inteligência Artificial (IA), e mais especificamente os Algoritmos Genéticos (AGs), oferecem uma abordagem heurística poderosa. Inspirados na teoria da evolução biológica, os AGs são capazes de encontrar soluções de alta qualidade para o TSP em um tempo de execução razoável. No entanto, o sucesso de um AG não é garantido e depende fortemente da configuração de seus operadores e parâmetros, como as taxas de mutação e crossover, o método de seleção e o número de gerações. A escolha inadequada desses parâmetros pode levar a uma convergência prematura para ótimos locais ou a uma exploração ineficiente do espaço de busca. Portanto, este trabalho se propõe a realizar uma análise comparativa do impacto de diferentes configurações de um algoritmo genético na resolução do TSP.

2. Metodologia

Para analisar o impacto das diferentes configurações do AG, foi utilizada a implementação desenvolvida em Python. A estrutura do AG foi encapsulada em uma classe, permitindo a fácil modificação de seus parâmetros para os testes experimentais.

2.1. Representação e Avaliação

Cada solução (cromossomo) é representada por uma lista contendo a permutação dos identificadores das cidades. A qualidade de cada rota é avaliada por sua distância total. A distância entre duas cidades é calculada usando a distância Euclidiana.

Código 1: Cálculo da distância Euclidiana

```
def calculate_distance(city1, city2):
    """Calculate Euclidean distance"""
    _, x1, y1 = city1
    _, x2, y2 = city2
    return math.sqrt((x2 - x1)**2 + (y2 - y1)**2)
```

A função de aptidão de uma rota é o inverso de sua distância total, garantindo que rotas mais curtas recebam um valor de aptidão maior.

Código 2: Cálculo da aptidão de uma rota

```
def calculate_fitness(self, route):
    """Calculate fitness of a route"""
    dist = total_distance(route, self.cities)
    return 1 / dist if dist > 0 else float('inf')
```

2.2. Operadores Genéticos

Os operadores genéticos são o cerne do processo evolutivo. Foram analisados diferentes métodos de seleção, crossover e mutação. A seleção por torneio define quais indivíduos serão os pais da próxima geração.

Código 3: Implementação da seleção por torneio

```
# Part of select_parents function
if self.selection_method == 'tournament':
    for _ in range(len(fitness_results)):
        tournament = random.sample(
            list(fitness_results.keys()), self.tournament_size)
        best_in_tournament = min(
            tournament, key=lambda x: -fitness_results[x])
        selection_results.append(best_in_tournament)
```

O operador de crossover, como o Order Crossover (OX), combina o material genético de dois pais.

Código 4: Implementação do Order Crossover (OX)

```
def order_crossover(self, parent1, parent2):
    size = len(parent1)
    child = [None] * size
    start, end = sorted(random.sample(range(size), 2))

    # Copy segment from parent1
    child[start:end+1] = parent1[start:end+1]

    # Fill remaining from parent2
    parent2_pos = end + 1
    child_pos = end + 1

    while None in child:
        if parent2[parent2_pos % size] not in child:
            child[child_pos % size] = parent2[parent2_pos % size
                ]

        child_pos += 1
        parent2_pos += 1

    return child, # Other child omitted
```

Por fim, a mutação por troca (swap) introduz diversidade na população.

Código 5: Implementação da mutação por troca (swap)

```
def mutate(self, route):
    """Perform mutation (swap mutation)"""
    for i in range(len(route)):
        if random.random() < self.mutation_rate:
            j = random.randint(0, len(route) - 1)
            route[i], route[j] = route[j], route[i]
    return route
```

2.3. Configuração do Experimento

Os experimentos foram conduzidos utilizando um conjunto de parâmetros padrão como base, conforme detalhado na Tabela 1. A partir dessa configuração, cada parâmetro foi variado individualmente para observar seu impacto no desempenho do algoritmo.

Tabela 1. Parâmetros padrão do Algoritmo Genético.

Parâmetro	Valor Padrão	Descrição
`pop_size`	100	Tamanho da população
`generations`	200	Número de gerações
`crossover_rate`	0.8	Taxa de crossover
`mutation_rate`	0.1	Taxa de mutação
`elite_size`	20	Tamanho do elitismo
`tournament_size`	5	Tamanho do torneio para seleção
`selection`	tournament	Método de seleção
`crossover`	pmx	Método de crossover

3. Resultados

Nesta seção, são apresentados os resultados dos experimentos realizados. Para cada configuração testada, o algoritmo foi executado 5 vezes em uma instância de 30 cidades, e os resultados apresentados correspondem à média aritmética dessas execuções, garantindo maior robustez estatística aos dados. As tabelas a seguir mostram os resultados obtidos ao variar sistematicamente os parâmetros do AG.

Tabela 2. Resultados com diferentes Taxas de Mutação.

Taxa de Mutação	Média das Distâncias	Tempo médio(ms)
0.01 (1%)	609.48	846.36
0.10 (10%)	657.89	878.59
0.25 (25%)	1047.76	999.14

Tabela 3. Resultados com diferentes Taxas de Crossover.

Taxa de Crossover	Média das Distâncias	Tempo médio(ms)
0.60 (60%)	681.57	841.06
0.80 (80%)	718.76	889.19
0.95 (95%)	673.43	941.78

Tabela 4. Resultados com diferente Número de Gerações.

Nº de Gerações	Média das Distâncias	Tempo médio(ms)
100	758.26	468.79
200	706.53	953.38
500	604.11	2248.65

Tabela 5. Resultados com diferentes Métodos de Seleção.

Método de Seleção	Média das Distâncias	Tempo médio(ms)
Roleta	1149.86	964.32
Torneio (t=2)	1054.25	963.47
Torneio (t=5)	669.14	910.76
Torneio (t=10)	606.73	947.09

Tabela 6. Resultados com diferentes Tamanhos de População.

Tamanho da População	Média das Distâncias	Tempo médio(ms)
50	709.14	430.34
100	706.53	953.38
300	838.47	3095.13

Tabela 7. Resultados com diferentes Tamanhos do Elitismo.

Tamanho do Elitismo	Média das Distâncias	Tempo médio(ms)
5	879.26	1056.09
20	706.53	953.38
50	654.08	808.94

4. Discussão

A análise dos resultados revela a interação complexa entre os parâmetros do Algoritmo Genético e seu impacto no equilíbrio entre *exploração* (busca por novas regiões no espaço de soluções) e *intensificação* (aprimoramento de soluções promissoras já encontradas).

A taxa de mutação (Tabela 2) demonstrou ser um fator crucial para a estabilidade do algoritmo. Uma taxa baixa de 1% foi ótima, indicando que a mutação deve ser um operador secundário, usado para introduzir diversidade e evitar a convergência prematura para ótimos locais. Taxas mais altas prejudicaram drasticamente o desempenho, destruindo soluções promissoras com mais frequência do que as aprimorando.

Em contrapartida, uma alta taxa de crossover de 95% (Tabela 3) se mostrou benéfica. Isso sugere que a recombinação de boas soluções é o principal motor de progresso do AG neste cenário. A combinação de uma alta taxa de crossover com uma baixa taxa de mutação favorece a intensificação, explorando intensivamente as características das melhores soluções da população.

A pressão seletiva, ajustada pelo método de seleção (Tabela 5), também foi determinante. A seleção por torneio com tamanho 10 superou todas as outras configurações. Um torneio maior aumenta a probabilidade de que indivíduos mais aptos sejam selecionados para a próxima geração. Isso, combinado com um elitismo robusto de 50 indivíduos (Tabela 7), cria uma forte força de intensificação, garantindo que as melhores soluções não apenas sobrevivam, mas também se propaguem rapidamente pela população.

O número de gerações (Tabela 4) confirmou o esperado: mais tempo de evolução leva a melhores resultados, com 500 gerações apresentando a melhor distância ao custo

de um maior tempo computacional. Curiosamente, o tamanho da população (Tabela :populacao) não seguiu a mesma lógica; uma população de 100 indivíduos foi superior a uma de 300. Isso pode indicar que uma população excessivamente grande, para o número de gerações testado, torna a convergência mais lenta e aumenta o custo computacional sem ganhos proporcionais de qualidade.

Em síntese, a configuração de melhor desempenho encontrada favorece uma forte intensificação (elitismo alto, torneio agressivo, crossover frequente) com uma leve exploração (mutação rara), executada por um número elevado de gerações em uma população de tamanho moderado. A análise evidencia que não há "bala de prata", mas sim um ajuste cuidadoso de parâmetros que direciona a busca evolutiva de forma eficaz para o problema em questão.

A título de curiosidade e para validar a análise, foi realizada uma execução final combinando os parâmetros que individualmente apresentaram os melhores resultados: população de 100, 500 gerações, taxa de crossover de 95%, taxa de mutação de 1%, elitismo de 50 e seleção por torneio com tamanho 10. Os resultados desta configuração otimizada são apresentados na Tabela 8.

Tabela 8. Resultado da configuração otimizada.

Execução	Distância Total	Tempo (ms)
Execução 1	588.79	2121.05
Execução 2	563.42	2134.66
Execução 3	569.19	2088.63
Média	**573.80**	**2114.78**

Como se pode observar, a combinação dos melhores parâmetros resultou na menor distância média encontrada em todos os experimentos (573.80), reforçando as conclusões sobre a eficácia de uma estratégia focada na intensificação para este problema.

5. Conclusão

Este trabalho demonstrou que a configuração de parâmetros de um Algoritmo Genético influencia profundamente sua capacidade de encontrar soluções de alta qualidade para o TSP. Os resultados indicam que não existe uma configuração universalmente ótima; o desempenho é um produto do equilíbrio delicado entre exploração e intensificação. A experiência obtida valida que a calibragem cuidadosa dos hiperparâmetros é uma etapa fundamental no desenvolvimento de soluções baseadas em algoritmos evolutivos. Para problemas de otimização combinatória complexos como o TSP, a performance não reside apenas na estrutura do algoritmo, mas também na sua sintonização fina. Como trabalhos futuros, sugere-se a automação do processo de ajuste de hiperparâmetros, utilizando técnicas de meta-otimização, e a aplicação da metodologia a outras classes de problemas de otimização combinatória.