GRASP com Path Relinking para o Problema do Caixeiro Viajante Simétrico

Felipe Sasdelli

Contents

1 Introdução

Este artigo apresenta a implementação de um algoritmo GRASP (Greedy Randomized Adaptive Search Procedures) com Path Relinking para resolver o Problema do Caixeiro Viajante Simétrico (TSP, do inglês Symmetric Traveling Salesman Problem). O TSP é um dos problemas mais estudados em otimização combinatória e consiste em encontrar o menor caminho que passa por todas as cidades de um conjunto exatamente uma vez, retornando à cidade de origem.

O GRASP é uma meta-heurística que constrói soluções iniciais através de uma fase construtiva e, em seguida, melhora essas soluções utilizando uma busca local. O Path Relinking é uma técnica que explora trajetórias entre soluções de elite para encontrar soluções ainda melhores.

2 Apresentação do Problema

O Problema do Caixeiro Viajante (TSP) pode ser descrito formalmente como um grafo completo $G = (V, E)$, onde V é o conjunto de cidades e E é o conjunto de arestas que conectam cada par de cidades. Cada aresta tem um peso d_{ij} que representa a distância entre as cidades

i e j. O objetivo é encontrar o circuito Hamiltoniano de menor custo, que passa por todas as cidades uma vez e retorna à cidade de origem.

Para o TSP simétrico, a distância entre duas cidades é a mesma em ambas as direções, ou seja, $d_{ij} = d_{ji}$.

3 Metodologia

3.1 Representação de Solução

Uma solução para o TSP é representada como uma permutação das cidades, onde a ordem das cidades na permutação indica a sequência em que elas serão visitadas. Por exemplo, se uma solução é representada pela permutação $[1,3,2,4]$, significa que o caixeiro começa na cidade 1, depois vai para a cidade 3, em seguida para a cidade 2, e finalmente para a cidade 4, retornando à cidade 1.

3.2 Função Objetivo

A função objetivo é minimizar a soma das distâncias entre cidades consecutivas na permutação, mais a distância entre a última cidade da permutação e a primeira. Formalmente, a função objetivo para uma solução S é dada por:

$$f(S) = \sum_{i=1}^{n-1} d_{S_i S_{i+1}} + d_{S_n S_1}$$

onde n é o número de cidades e S_i é a cidade na posição i da permutação S.

```rust
fn eval(&mut self, instance: &Instance) {
    if self.path.len() != instance.num_cities {
        panic!("Path length does not match the number of cities in the instance");
    }
    self.total_distance = 0;

    for i in 0..self.path.len() - 1 {
        let from = self.path[i];
        let to = self.path[i + 1];
        self.total_distance += instance.distances[from][to]
    }

    let last = *self.path.last().unwrap();
    let first = self.path[0];
    self.total_distance += instance.distances[last][first]
}
```

4 Método Construtivo

A fase construtiva do GRASP começa com uma solução parcial e a expande sequencialmente, escolhendo a próxima cidade a ser adicionada a partir de uma lista restrita de candidatos (RCL - Restricted Candidate List). Os candidatos na RCL são selecionados com base em um critério de qualidade, como a menor distância até a próxima cidade. A escolha final entre os candidatos é feita de maneira aleatória.

Após a construção da solução inicial, uma busca local é aplicada para encontrar um mínimo local. A busca local utilizada neste trabalho é a 2-opt, que tenta melhorar a solução invertendo pares de arestas.

4.1 Implementação em Rust

A implementação do algoritmo foi feita em Rust, uma linguagem de programação que oferece alto desempenho e segurança. A seguir, o código-fonte principal do método construtivo:

```rust
fn constructive_phase(instance: &Instance) -> Solution {
    let mut solution = Solution::new(instance.num_cities);
    let mut remaining: Vec<usize> = (0..instance.num_cities).collect();

    let start_city = remaining.remove(rand::random::<usize>() % remaining.len());
    solution.path.push(start_city);

    while !remaining.is_empty() {
        let last_city = *solution.path.last().unwrap();
        let mut candidates: Vec<(usize, i32)> = remaining
            .iter()
            .map(|&city| (city, instance.distances[last_city][city]))
            .collect();
        candidates.sort_by_key(|&(_, dist)| dist);

        let next_city = candidates[rand::random::<usize>() % candidates.len()].0;
        remaining.retain(|&x| x != next_city);
        solution.path.push(next_city);
    }

    solution
}
```

5 Conclusão

Este trabalho implementa uma variação do GRASP com Path Relinking para o Problema do Caixeiro Viajante Simétrico. A implementação foi feita em Rust, visando obter alta eficiência e aproveitar o paralelismo oferecido pela linguagem. Os experimentos futuros irão comparar diferentes técnicas de Path Relinking para o GRASP, como Dynamic e Static Path Relinking, além de comparar o GRASP com e sem Path Relinking.