Trabalho Algoritmos Aproximativos

Carlos André Feldmann Júnior

Foi proposto implementar o algoritmo TSP(Travel Salesperson Problem) utilizando força bruta e um algoritmo aproximátivo.

Solução Técnica

Foi utilizado a linguagem de programação Rust, inicialmente tentei fazer todas as funções de criação de grafo na mão, mas devido ao tempo de entrega acabei cedendo e utilizando bibliotecas de grafos(`petgraph`).

Algoritmos

Brute Force

Calcula todos os caminhos(todas permutações do vetor de cidades), retornando a solução com o menor caminho possível.

```rust
// src/bruteforce.rs
let permutations = (0..cities)
    .collect::<Vec<usize> > ()
    .into_iter()
    .permutations(cities);
for solution in permutations {

    // Timeout mechanism
    if ! keep_executing.load(Relaxed) {
        let mut forced_solution = best_solution.unwrap();
        (&mut forced_solution).force_stopped();
        return forced_solution;
    }
    // Store the best solution
    let distance = matrix.calculate_distance( & solution);
    if best_solution.is_none() || distance < best_solution.as_ref().unwrap()
        best_solution = Some(matrix.build_solution(solution));
    }
}
```

Christofides

Aproximativo com fator de 3/2.

O algoritmo é composto pelos seguintes passos:

- Criar uma MST(Minimum Spanning Tree) do grafo das cidades
- Pegar nodos de grau impar, e relacionar em pares com a menor distancia possível.

- Navegar a arvore em DSF
- Remover caminhos duplicados

```
//christofides.rs
// Convert the matrix to a graph
let g = < TSPMatrix as Into<Graph<usize, usize, Undirected> > >::into(matri›
// Get the MST
let mut mst = UnGraph::<_, _ >::from_elements(min_spanning_tree( & g));
// Create the the paths between odd edges
Self::add_odd_edges( & mut mst, & matrix);
// Deep search first to build the solution
let mut solution = vec![];
let mut dfs = Dfs::new( & mst, n(0));

while let Some(visited) = dfs.next(&mst) {
// Ignore visited cities
    if ! solution.contains( & visited.index()) {
        solution.push(visited.index())
    }
}
```

Benchmarks

Algorithm	Dataset	Execution Time	Solution	Error
Brute Force	1	4.878639725s	253	0%
Brute Force	2	75.457µs	1248	0%
Brute Force	3	20min (Timeout)	1194	0%
Brute Force	4	20min (Timeout)	22304	218%(15291)
Brute Force	5	20min (Timeout)	40872	48.1%(13269)
Christofides	1	110.109µs	269	6.3%(16)
Christofides	2	36.585µs	1272	1.9%(24)
Christofides	3	20.876µs	1424	19.3%(230)
Christofides	4	169.485µs	8452	20.5%(1439)
Christofides	5	51.367µs	34277	24.2%(6674)

Bruteforce

É executado em tempo exponencial(N!), por isso nos primeiros datasets obteve um resultado satisfatório, porém ao se aumentar o número de cidades fica inviável executa-lo em tempo real. Um detalhe importante, no dataset 3 foi encontrado o melhor resultado antes de forçar a execução.

Christofides

Foi executado em tempo polinomial, não variando muito conforme o tamanho da entrada. Os dados apresentados incluem ruido, não foi feito uma média de várias execuções.
Observa-se que nenhum valor aproximado ultrapassou o limite de 50% de erro do algoritmo.

Execução

Instalar o toolchain do ecosistema de Rust em https://rustup.rs/
Rodar o comando `cargo run --release`