# Paralelizando metaheurística em Python para solução do *Team Orienteering Problem*

**Tiago Funk**[1], **Adriano Fiorese**[1]

[1]Departamento de Ciências da Computação – (DCC)
Universidade do Estado de Santa Catarina – (UDESC)
Caixa Postal 631 – 89.219-710 – Joinville – SC – Brasil

tiagoff.tf@gmail.com, adriano.fiorese@udesc.br

***Resumo.*** *Este trabalho realiza uma análise envolvendo uma implementação paralalelizada da metaheurística Greedy Randomized Adaptive Search Procedures para tratamento do Team Orienteering Problem, utilizando a linguagem Python. Experimento comparando as versões não paralelizada e paralelizada mostra que a versão paralelizada obteve os mesmos resultados, porém é escalável, convergindo em tempo consideravelmente menor.*

## 1. Introdução

O *Team Orienteering Problem* (TOP) é um problema de alocação de rotas para uma frota. As características desse problema preconizam que cada membro da frota receba uma rota que deve ser cumprida em um tempo limite. Cada rota não pode repetir locais de visitas e nem pode visitar locais de outras rotas. Cada local possui um recompensa que é coletada quando a visita é realizada, porém não é necessário que todos os locais sejam visitados, uma vez que o objetivo é maximizar a recompensa total da equipe [Chao et al. 1996].

Metaheurísticas são algoritmos genéricos que são utilizados em problemas de otimização. Eles têm como proposta resolverem problemas computacionalmente exigentes de forma mais rápida quando comparados a similares exatos. Porém, metaheurísticas canônicas em geral não são desenvolvidas tendo em vista a computação distribuída ou paralela. Criar um ambiente fácil e simples de implementação paralela pode se mostrar um diferencial, pois permite a utilização de todo o recurso do computador para executar o algoritmo. Devido a complexidade dos problemas abordados pelas metaheurísticas, a utilização de técnicas de programação paralelizada torna-se atrativa para problemas complexos tornando mais rápida a resolução do problema.

Assim, este trabalho realiza uma implementação da metaheurística *Greedy Randomized Adaptive Search Procedures* (GRASP) usando a linguagem Python de forma paralelizada, utilizando, para tanto, o pacote *multiprocessing*, evitando o Global Interpreter Lock (GIL) usando subprocessos em vez de *threads*. A contribuição deste trabalho reside no estudo da utilização da linguagem Python para a paralelização da metaheurística GRASP avaliando se esta implementação oferece a robustez que a implementação não paralelizada traz, bem como disponibilizando tal aparato para a tomada de decisão quanto a escolha da mais adequada rota visando a maximização do resultado da solução do problema TOP.

## 2. Solução Proposta

O algoritmo GRASP é dividido em dois passos importantes: a fase de geração e fase de busca local. Na fase de geração são criadas soluções novas inserindo vértices na solução candidata. Cada vértice é selecionado considerando um valor e a escolha é probabilística. Vértices com valores mais altos tem mais chance de serem escolhidos. Esse valor é calculado considerando o valor de recompensa ganha ao ser visitado e a distância entre o vértice e o último vértice visitado na rota. O processo continua enquanto for possível inserir novos vértices nas rotas. Na fase de busca local, são realizadas operações que tem a intenção de melhorar a solução. São quatro operações ao todo. *i*.Remoção probabilística: uma quantidade de vértices são escolhidos aleatoriamente para serem removidos da solução. *ii*.*Exchange*: troca dois vértices de posição na solução. A troca que mais diminuir a custo total da rota é selecionada. *iii*.Melhor adição: Aqui são inseridos vértices na rota. A adição que oferecer mais recompensa à rota e não ofender as restrições da solução é selecionada. *iv*.Melhor troca: este operador vai trocar vértices não utilizados por vértices utilizados na rota. A troca que mais oferecer mais recompensa e/ou diminuir o custo da rota vai ser selecionado.

A técnica de paralelização do algoritmo se beneficia do fato de que o GRASP é uma metaheuristica *multi-start*, ou seja, cria e aplica operações de busca local várias vezes e não considera soluções anteriores. Assim, é possível atribuir ao algoritmo como critério de parada o número de iterações. Dessa forma, para paralelizar, a forma mais simples é dividir o número de iterações entre os núcleos do processador.

Para realizar o experimento, foram utilizadas 16 instâncias. O algoritmo foi implementado na linguagem Python 3.6.9, e executado no sistema operacional Ubuntu 18.04.5 LTS com processador Intel Core i5-8265U 1.60GHz de 8 núcleos e 8 Gb de RAM. Cada instância foi executada 30 vezes, com coleta de tempo de execução e qualidade de solução. Foi utilizada a ferramenta *irace* para calibração de parâmetros.

## 3. Considerações Finais

Foram coletados dois dados importantes do experimento. O primeiro é a qualidade da solução e o segundo é o tempo de execução. O objetivo é comparar se a versão paralelizada do algoritmo executa em menos tempo e se obtém o mesmo resultado. Houve a execução do teste estatístico ANOVA com 95% de nível de confiança para verificar se a diferença é significativa entre a qualidade da solução dos dois algoritmos. O resultado demonstrou que a qualidade da solução é a mesma entre os dois algoritmos. Um teste ANOVA com nível de confiança de 95% também foi executado para verificar a significância estatística da comparação entre as duas abordagens no quesito tempo de execução. O resultado indica que não representam tempos de execução equivalentes, como esperado. Para exemplificar a redução de tempo, as execuções médias da versão não paralelizada foi de 577 ms, enquanto o tempo médio da versão paralelizada foi de 100 ms. Esses testes comprovam que o algoritmo paralelizado obteve um resultado equivalente no quesito qualidade de solução, mas em menor tempo comparado ao equivalente não paralelizado.

## Referências

Chao, I.-M., Golden, B. L., and Wasil, E. A. (1996). The team orienteering problem. *European journal of operational research*, 88(3):464–474.