Apresentação Metaheurísticas

Arthur Ferraz

Sumário

- Buscas locais
  - Tabu Search
  - Simulated Anneling
- Buscas populacionais
  - Ant Colony
  - Differential Evolution
  - Particle Swarm
  - Genetic Algorithm

Buscas Locais

Tabu Search

IMPLEMENTAÇÃO

- Implementação dividida em múltiplas funções:
- Função objective responsável por comparar a performance de dois caminhos no tsp e retornar true ou false.
- Funções de impressão imprimirBloqueio e imprimirTipo foram usadas para ter feedback durante o desenvolvimento do codigo
- createMoves preenche e retorna uma lista com todos os movimentos para cada combinação de valores i e j válidos.
- BuscaLocal percorre toda a lista moves, aplicando-os à permutação atual sol para que o melhor vizinho de sol seja escolhido.
- TabuSearch define um tempo de bloqueio para os movimentos utilizados e realiza K iterações de busca local, imprimindo os resultados de cada uma e retorna o melhor.

Teste

```
include("tabuSearch.jl");

tsp=readTSPLIB(:berlin52);

tabuSearch(tsp);

tsp=readTSPLIB(:kroA200);

tabuSearch(tsp);
```

Simulated Annealing

IMPLEMENTAÇÃO

Função única;

- Precisa dos parâmetros de temperatura inicial t0, constante de variação da temperatura alfa e dois valores pra controle de loops;
- Utiliza-se de dois loops em cadeia:
  - Loop externo: varia a temperatura a cada iteração
  - Loop interno: com temperatura constante, busca soluções entre diferentes vizinhos.
- Utiliza roleta para decidir qual movimento aplicar;
- Soluções:
  - Caso encontre alguma solução melhor ou igual à atual, atualiza;
  - Caso contrário, roda uma probabilidade para aceitar ou não essa solução pior, que diminuirá à medida que a temperatura tende à zero.

Teste

```
include("SimAnel.jl");

tsp=readTSPLIB(:berlin52);

simAnel!(tsp, 100, 0.1, 30, 1000);

tsp=readTSPLIB(:kroA200);

simAnel!(tsp, 100, 0.1, 30, 1000);
```

Buscas Populacionais

Ant Colony

IMPLEMENTAÇÃO

Função única;

- Em cada iteração, cria uma rota por busca local com cada formiga da população;
- Sorteia uma cidade origem, calcula a probabilidade de a formiga visitar cada uma das outras cidades e utiliza uma roleta pra decidir a próxima cidade da rota
- Todas as rotas realizadas liberam feromonios no caminho que servem para reforçar as rotas mais utilizadas
- A melhor formiga deixa o dobro de feromonio

Teste

```
include("AntColony.jl");
tsp=readTSPLIB(:berlin52);
antColony(tsp, 100, 10);
tsp=readTSPLIB(:kroA200);
antColony(tsp, 15, 10);
```

Differential Evolution

IMPLEMENTAÇÃO

Função única;

- Cria população aleatória;
- Para cada indivíduo em cada iteração:
  - Escolhe 3 indivíduos distintos aleatóriamente;
  - Cria um indivíduo mutado com esses 3;
  - Realiza um crossover entre o mutado e o atual;
  - Substitui o atual pelo crossover, se tiver melhor função objetivo.

Teste

```
include("DifferentialEvolution.jl")

name = "rosenbrock";

diffEvo(f[name], l[name], u[name], CR=0.01);

name = "sphere";

diffEvo(f[name], l[name], u[name], CR=0.01);
```

Particle Swarm

IMPLEMENTAÇÃO

Função única;

- Cria população aleatória, onde cada indivíduo possui posição, velocidade e melhor posição histórica
- Para cada indivíduo em cada iteração:
  - Atualiza melhor performace indivídual e o melhor global;
  - Atualiza a velocidade com parâmetros aleatórios;
  - Movimenta o indivíduo a essa velocidade.

Teste

```
include("ParticleSwarm.jl")

name = "rosenbrock";

partSwarm(f[name], l[name], u[name]);

name = "sphere";

partSwarm(f[name], l[name], u[name]);
```

Genetic Algorithm

IMPLEMENTAÇÃO

Implementação dividida em múltiplas funções:

- Para cada indivíduo de cada iteração:
  - Selecione dois genitores distintos;
  - Gere filhos entre eles:
    - Dada uma probabilidade, ao passar, use um crossover entre ambos;
    - Caso não passe, os filhos serão copias identicas deles;
  - Atualiza o indivíduo atual para o melhor entre o genitor 1 e o genitor 2.
  - Dada uma probabilidade, realiza ou não uma mutação no indivíduo atual;
- Para cada iteração, mantém o melhor indivíduo da geração anterior.

Teste

```
include("Genetic.jl");

tsp=readTSPLIB(:berlin52);

genetic(tsp, N=500, K=200, CR=0.1);

tsp=readTSPLIB(:kroA200);

genetic(tsp, N=250, K=1000, CR=0.1);
```