Universidade Federal do Rio Grande do Norte
Departamento de Informática e Matemática Aplicada

Disciplina: DIM0406 — Algoritmos Avançados
Docente: Sílvia Maria Diniz Monteiro Maia
Discente: Felipe Cortez de Sá

GRASP-VNS aplicado ao problema de Steiner com rotulação mínima

1 Introdução

Neste relatório são apresentados os algoritmos *Greedy Randomized Adaptative Search Procedure* e *Variable Neighbourhood Search* para solução do problema da árvore de Steiner com rotulação mínima. Os princípios de cada técnica são descritos, assim como a maneira em que foram utilizados para resolver especificamente o problema. É realizada uma análise de complexidade em tempo para as implementações, identificados os gargalos. Explica-se como foram gerados os casos de teste e os seus parâmetros e são apresentados resultados comparando o resultado das execuções para cada técnica, incluindo o algoritmo exato desenvolvido na segunda unidade. EXPERIMENTOS COMPUTACIONAIS PARA DEFINIÇÃO DO ALGORITMO! Além disso, é feito um adendo para o relatório do primeiro trabalho, apresentando os resultados e conclusão previamente faltantes.

2 Metaheurísticas utilizadas

2.1 GRASP

O *Greedy Randomized Adaptative Search Procedure* é comumente utilizado em problemas de otimização combinatória. A cada iteração, é realizada uma fase de construção, em que se gera uma solução para o problema e posteriormente uma fase de busca local, procurando um mínimo local na vizinhança da solução gerada. Se a melhor solução global é encontrada na iteração, atualiza-se a variável que contém a melhor solução. É um algoritmo de inicialização múltipla, ou seja, as duas fases são repetidas até o critério de parada ser satisfeito, podendo ser o número de iterações ou o tempo de execução, por exemplo. Na fase de construção, é criada uma lista de candidatos restritos, possuindo os candidatos cujos elementos adicionados minimizam os custos incrementais. O elemento é selecionado aleatoriamente da RCL.

```
grasp(limite) {
    col* = {}
    int no_improv = 0

    while(no_improv < limite) {
        col = {}
        construct(col)
        local(col) ou vns(col)
        if(card(col) < card(col*)) {
```

```
            col* = col
            no_improv = 0
        } else {
            ++no_improv
        }
    }
}

construct() {
    if(iteration > 2) {
        rcl = {1, 1, ... , 1}
        col[random()] = 1
    } else {
        rcl = {0, 0, ... , 0}
    }

    while(comp(c) > 1) {
        rcl = argmin comp(c)
        col = col ∪ rcl[random()]
    }
}
```

Listing 1: Pseudocódigo para GRASP

2.2 VNS

O *Variable Neighbourhood Search* faz uso de múltiplas estruturas de vizinhança, explorando comumente espaços cada vez mais distantes e maiores, portanto mais custosos. Para fugir de mínimos locais, o algoritmo possui uma fase de agitação, em que a solução encontrada pode ser trocada por uma pior a fim de diversificar a busca, explotando melhor o espaço.

```
vns(col, kmax)
col2 = col
k = 1
while(k <= kmax) {
    shaking(col2, k)
    local(col2)
    if(card(col2) < card(col)) {
        col = col2
        k = 1
    } else {
        ++k
    }
}

shaking(col, k) {
    col2 = col
    for(i in 1..k) {
        if(i <= card(col)) {
            col[random(cores utilizadas em col)] = 0
        } else {
            col[random(cores nao utilizadas em col)] = 1
        }
    }

    while(comp(c) > 1) {
        melhores = argmin comp(c)
        col2[random(melhores)] = 1
    }
}
```

Listing 2: Pseudocódigo para VNS

3 Metaheurística aplicada ao problema

3.1 GRASP

4 Complexidade

5 Casos teste utilizados

Os casos teste utilizados são gerados automaticamente por um programa **generate.c** de acordo com parâmetros de entrada. Os parâmetros são **SIZE**, a quantidade de nós do grafo, **COLORS**, o número de rótulos, **DENSITY**, a proporção de arestas para cada nó, e **BASIC**, a quantidade de nós básicos. **DENSITY** funciona percorrendo a matriz de adjacência que representa o grafo e de acordo com a probabilidade definida (sendo 0 e 100 equivalentes a 0% e 100%, respectivamente) adicionando ou não uma aresta de rotulação aleatória ligando dois nós. O arquivo gerado é então passado para o programa principal.

A fim de comparar os resultados com o trabalho realizado por Cerulli, [4], os parâmetros dos testes são os mesmos, isto é, tem-se uma combinação entre **SIZE** $\in \{50, 100\}$, **COLORS** $\in \{0.25n, 0.5n, n, 1.25n\}$, **DENSITY** $\in \{0.2, 0.5, 0.8\}$ e **BASIC** $\in \{0.2n, 0.4n\}$. Cada caso teste é executado dez vezes diferentes e são apresentadas a média, melhor e pior casos e mediana.

O código que gera os arquivos de caso teste para os parâmetros desejados está em **generate.py**.

6 Resultados

7 Correções do primeiro trabalho

7.1 Técnica utilizada

7.2 Resultados

7.3 Conclusão

Referências

[1] S. Consoli, K. Darby-Dowman, N. Mladenovic, J.A. Moreno-Perez. *Variable neighbourhood search for the minimum labelling Steiner tree problem.* Annals of Operations Research, 2009.

https://www.researchgate.net/publication/225327721_Variable_neighbourhood_search_for_the_minimum_labelling_Steiner_tree_problem

[2] *Graphviz — Graph Visualization Software.*

http://www.graphviz.org/

[3] Glover, F., Kochenberger, G. A. et al. *Handbook of Metaheuristics*. Kluwer Academic Publishers

[4] R. Cerulli, A. Fink, M. Gentili e S. Voß. *Extensions of the minimum labelling spanning tree problem*. Journal of Telecommunications and Information Technology, 2006.

https://www.researchgate.net/publication/228668519_Extensions_of_the_minimum_labelling_spanning_tree_problem

[5] *Stack Overflow — Execution time of a C program.*

http://stackoverflow.com/questions/5248915/execution-time-of-c-program