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, identificando 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. 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.

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.

3 Metaheurística aplicada ao problema

3.1 GRASP

No problema, a fase de construção gera uma lista de candidatos restritos calculando argmin(comp(c)) para cada cor não utilizada, onde comp(c) é o número de componentes conexos do grafo com a coloração c que incluem pelo menos um nó básico. Após a segunda repetição, o primeiro rótulo a ser adicionado é totalmente aleatório, ou seja, a lista de candidatos é inicializada como $1, 1, ..., 1$, explotando melhor o espaço de busca em vez de escolher sempre rótulos que minimizam comp(c).

Após a fase de construção, é feita uma busca local, que consiste em tentar remover cores da solução e verificar se ainda obtém-se um grafo conexo, configurando outra solução válida.

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

    while(no_improv < limite) {
        col = {}
        construct(col)
        local(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
    }

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

Listing 1: Pseudocódigo para GRASP

3.2 VNS

Quando o algoritmo é executado independentemente, a configuração das cores inicial é totalmente aleatória, e a fase de agitação remove e adiciona cores dependendo da vizinhança.

```
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

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 Experimentos comparativos

8 Conclusões

9 Correções do primeiro trabalho

9.1 Técnica utilizada

O *branch-and-bound* é um algoritmo de otimização que explora o espaço de busca de maneira mais eficiente que uma enumeração total de soluções possíveis por força-bruta. Atualizando o limite inferior continuamente, é possível eliminar a exploração de regiões não-promissoras do espaço de busca.

9.2 Resultados

Opa.

9.3 Conclusão

9.4 Considerações adicionais

O código referente ao algoritmo exato foi modificado para aceitar entradas de um caso de teste, foi comentado mais extensivamente e agora é cronometrado para possibilitar a análise de resultados.

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

n	l	d	b	Exato	GRASP	VNS
50	12	20	10	2.79	2.79	3.11
50	12	20	20	3.79	3.79	1.18
50	12	50	10	2.14	2.14	0.74
50	12	50	20	2.15	2.15	1.02
50	12	80	10	0.62	0.62	0.26
50	12	80	20	0.99	0.99	0.22
50	25	20	10	2.83	2.83	6.97
50	25	20	20	5.81	5.81	6.17
50	25	50	10	3.79	3.79	3.22
50	25	50	20	5.70	5.70	4.83
50	25	80	10	1.35	1.35	1.14
50	25	80	20	3.37	3.37	0.99
50	50	20	10	7.13	7.13	12.03
50	50	20	20	17.33	17.33	24.27
50	50	50	10	9.62	9.62	9.23
50	50	50	20	14.04	14.04	8.75
50	50	80	10	3.49	3.49	1.66
50	50	80	20	5.63	5.63	3.03
50	62	20	10	9.90	9.90	19.35
50	62	20	20	29.64	29.64	28.15
50	62	50	10	6.64	6.64	14.36
50	62	50	20	10.53	10.53	16.71
50	62	80	10	7.70	7.70	6.64
50	62	80	20	8.15	8.15	7.57
100	25	20	20	16.60	16.60	21.70
100	25	20	40	19.53	19.53	15.79
100	25	50	20	16.91	16.91	5.51
100	25	50	40	16.77	16.77	5.20
100	25	80	20	7.94	7.94	1.26
100	25	80	40	9.59	9.59	4.28
100	50	20	20	32.87	32.87	48.95
100	50	20	40	48.67	48.67	108.81
100	50	50	20	29.56	29.56	28.43
100	50	50	40	37.94	37.94	37.15
100	50	80	20	21.23	21.23	6.22
100	50	80	40	22.80	22.80	7.70
100	100	20	20	75.55	75.55	268.45
100	100	20	40	145.81	145.81	399.00
100	100	50	20	113.30	113.30	145.98
100	100	50	40	111.12	111.12	184.37
100	100	80	20	60.98	60.98	58.55
100	100	80	40	61.79	61.79	56.35
100	125	20	20	170.28	170.28	226.11
100	125	20	40	166.12	166.12	342.48
100	125	50	20	105.08	105.08	154.44
100	125	50	40	155.19	155.19	184.92
100	125	80	20	89.28	89.28	89.35
100	125	80	40	111.02	111.02	74.04

Tabela 1: Opa