Heurística VNS para o problema do corte de rotulação mínima

Augusto Cesar Bordini Braga

Instituto de Computação – Universidade Federal Fluminense (UFF) – Niterói – RJ - Brasil

gutocnet@ic.uff.br

Resumo: O problema do corte de rotulação mínima (MLC em inglês, de Minimum Labelling Cut) é de natureza de otimização combinatória na área de algoritmos em grafos. Dado um grafo conexo não direcionado com as arestas rotuladas (ou coloridas), o objetivo é achar um corte de arestas no qual o número de rótulos distintos associados à estas arestas seja o menor possível.

1. Introdução

O problema MLC tem como entrada um grafo conexo não-direcionado, com arestas rotuladas (ou coloridas), com cada rótulo podendo estar associado a zero ou mais arestas, entretanto, cada aresta está associada somente a 1 rótulo. O objetivo é encontrar um corte (conjunto desconectante de arestas) deste grafo com o menor número possível de rótulos associados. Pode se dizer, também, que se deseja encontrar o maior número de rótulos que mantenham o grafo desconectado.

Cito como exemplo de aplicação prática um sistema de planejamento de transportes de uma cidade. Podemos modelar os vértices como sendo os bairros atendidos e os rótulos como sendo as empresas de ônibus que prestam o serviço. Desta maneira, podemos obter o número mínimo de empresas que precisam deixar de operar para que não seja possível ir para todos os bairros da cidade de ônibus.

Definição: Problema do corte de rotulação mínima. (Minimum labelling Cut Problem):

Dado: Um grafo simples conexo G = (V, E, L), em que V é o conjunto de vértices, E é o conjunto de arestas, e L é o conjunto dos rótulos.

Objetivo: Encontrar um corte C de G tal que min $|L_C|$, onde L_C é o conjunto de rótulos ausentes em C.

O grafo (a) é um exemplo de entrada, (b) seria o grafo (a) após o corte de menor número de rótulos associados e (c) a solução MLC ótima. Ver figura 1.

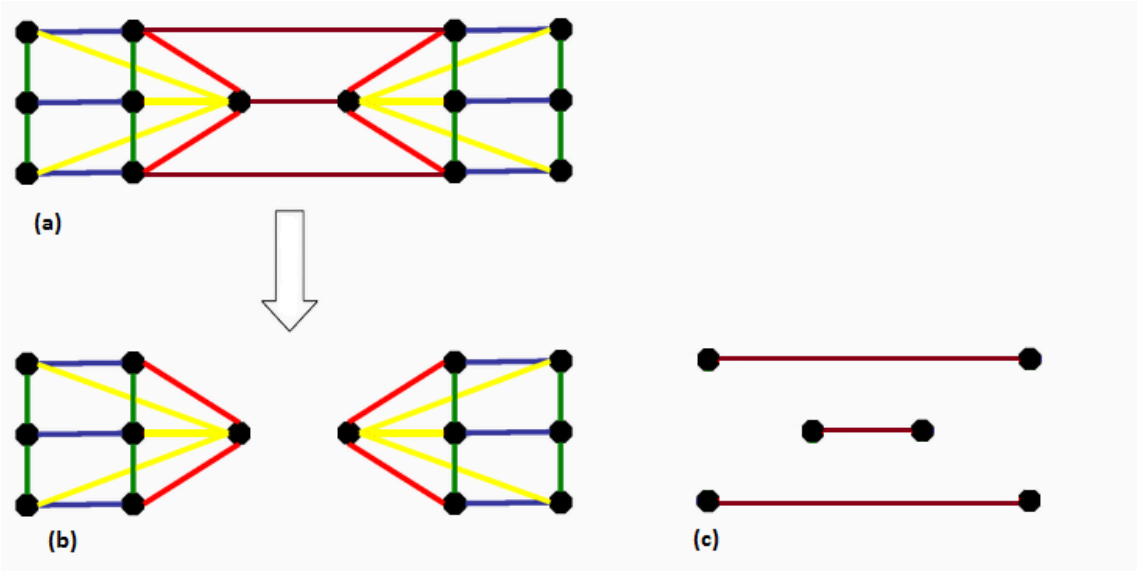

Figura 1

Este problema é o 'dual' do problema Minimum Labelling Spanning Tree (MLST), o qual busca encontrar o menor número de rótulos que mantém o grafo conexo.

O algoritmo MLC busca maximizar o número de rótulos dentro do grafo C que mantenham o grafo desconexo. Ao final das iterações, os rótulos ausentes de C são a solução esperada.

A estrutura deste trabalho é a seguinte: na seção 2 são descritos os procedimentos e funções utilizados. Na seção 3 são descritos alguns detalhes de implementação, assim como algumas abordagens testadas e o algoritmo em si, o qual é uma meta-heurística VNS com abordagens complementares. A seção 4 contém a análise computacional. As conclusões estão na seção 5.

2. Procedimentos e funções

Nesta seção são descritos os procedimentos e funções utilizados pelo algoritmo.

2.1 Função de Boltzmann

Para a construção das soluções, a inclusão de um rótulo (candidato à nova solução) que gere o maior número de componentes conexos (método guloso), geralmente deixa o algoritmo preso em ótimos locais. Esta é a abordagem determinística do MVCA usado por Krumke e Wirth[3] e outros autores no problema MLST. Para fugir dos ótimos locais é utilizada a função de Boltzmann, a qual introduz probabilidade na escolha de um novo rótulo candidato a integrar uma nova solução. Esta tática foi utilizada por Consoli et al.[1] no problema MLST e, além disso, faz parte da meta-heurística conhecida como Simulated Annelling.

A função de Boltzmann é utilizada na construção das soluções do espaço complementar, da solução inicial e na busca local.

A probabilidade de admissão de um componente pior em uma solução parcial é avaliada de acordo com o critério habitual (número de componentes conexos gerados) normalizado pela

função de Boltzmann (exp (Δ /T)), em que o parâmetro T, referido como *temperatura*, controla a dinâmica da função. Inicialmente, o valor de T é grande, permitindo assim que rótulos piores sejam aceitos, e é gradualmente reduzido pela seguinte lei geométrica de resfriamento (figura 2):

$$T_{k+1} = \alpha \cdot T_k \quad \text{where} \quad \begin{cases} T_0 = |Best_C|, \\ \alpha = 1/|Best_C| \in [0, 1], \end{cases}$$

Figura 2

- *BestC* é a atual melhor solução e |*BestC*| seu número de rótulos.

Esta fórmula, também conhecida por *lei de resfriamento,* além de prover rápida convergência para o problema, resulta em um bom equilíbrio entre as capacidades de intensificação e diversificação. Além disso, esta não requer qualquer intervenção do usuário em relação a configuração de seus parâmetros, pois é guiada automaticamente pelo tamanho da melhor solução *BestC*.

Portanto todo o processo de otimização é capaz de reagir em resposta ao comportamento do algoritmo de busca e adaptar suas configurações “online” de acordo com a instância do problema em avaliação.

A inclusão probabilística de novos rótulos tem como propósito permitir que uma quantidade menor de rótulos “promissores” seja incluída a cada iteração nas soluções parciais. Os valores de probabilidade assinalados a cada rótulo são diretamente proporcionais ao número de componentes conexos que eles produzem caso incluídos à solução parcial. Desta maneira, em cada passo, rótulos com um maior número de componentes conexos resultantes terão maior probabilidade de serem escolhidos em detrimento daqueles que possuem um número menor de componentes conexos.

Além disso, a progressiva redução do parâmetro de temperatura na lei de resfriamento produz, passo a passo, um aumento da diversidade em probabilidades. Isso significa que a diferença entre as probabilidades de 2 rótulos com número diferentes de componentes conexos é mais alta a medida que o algoritmo prossegue. Conforme as iterações do algoritmo vão acontecendo, a probabilidade de escolha de um rótulo que resulte numa solução com número pequeno de componentes conexos tende a zero, fazendo que a busca se pareça com o MVCA.

2.2 Geração da solução inicial

A construção da solução inicial consiste na escolha iterativa rótulo a rótulo. Para cada inclusão é calculada a conectividade gerada por cada rótulo caso o mesmo fosse incluído na solução. Um rótulo é escolhido probabilisticamente para entrar na solução, sendo estas probabilidades normalizadas pela função de Boltzmann e seguindo a lei do resfriamento. O procedimento finaliza quando não existem mais rótulos possíveis de adicionar à solução atual sem que a mesma se torne um grafo conexo.

Algoritmo 1

Definições:

Conectividade(BestC) = número de componentes conexos de BestC.

Todos = Todos os rótulos do problema.

Δ = Conectividade (BestC $\cup$ {v}) - Conectividade (BestC $\cup$ {s})

T => $T_{|BestC|+1} = T_{|BestC|} * \alpha$

T_0 = 1 e α = 1/|BestC|

Início

BestC ← 0

Repete

Seja s $\in$ (Todos - BestC), o rótulo que minimiza Conectividade (BestC $\cup$ {s})

Para cada v $\in$ (Todos – BestC) **faça**

Calcule a probabilidade P(v) normalizada pela função de Boltzmann: exp(Δ /T)

Selecione aleatoriamente um rótulo não usado u $\in$ (Todos– BestC) de acordo com as probabilidades P(.) tal que Conectividade (C+u) > 1

Caso exista {u} tal que Conectividade (BestC+u) > 1 então:

Adicione u à solução BestC: BestC ← BestC $\cup$ {u}

Atualize Conectividade (BestC)

Caso contrário, sai do loop.

Até "sai loop" = Verdadeiro

Fim

2.3 Espaço Complementar

O VNS complementar é uma busca local baseada no VNS que visa o aumento do fator da diversidade do VNS básico.

Definição:

Dado: Uma solução C de um grafo simples, conexo e rotulado G=(V,E,L)

Defina: Seja (L-C) o espaço complementar de C, isto é, o conjunto de todos os rótulos que não estão contidos na solução C.

O processo iterativo de extração de uma nova solução do espaço complementar da melhor solução global, ajuda o algoritmo a fugir de soluções ótimas locais, haja vista que o espaço complementar possui uma zona de busca muito diferente da solução corrente. Ver figura 3.

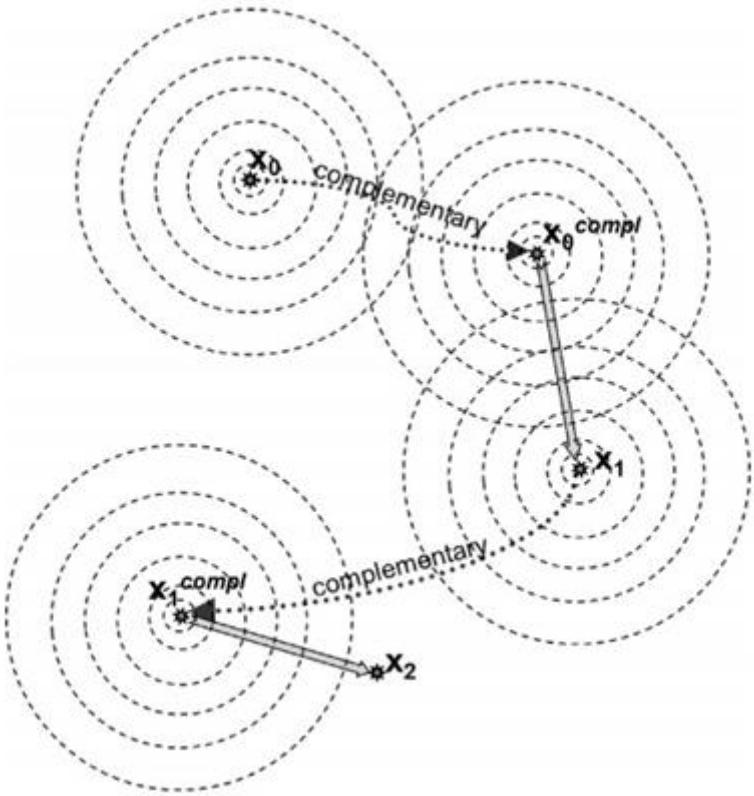

Figura 3

Este processo produz um pico imediato da capacidade de diversificação de todo o processo de busca local. Para obter uma solução de espaço complementar, este usa o método probabilístico (função de Boltzmann) como método construtivo aplicado ao subgrafo de G com rótulos em (L-C). O procedimento espaço complementar é sempre aplicado em relação à melhor solução corrente.

A primeira parte deste procedimento finaliza em 2 situações:
- uma solução parcial é obtida e o conjunto de cores não utilizadas contido no espaço complementar é vazio (L-C=Ø)

Ou

- Após n iterações, todos os rótulos restantes em *L-C* gerariam um grafo conexo caso adicionados a esta nova solução.

A segunda parte consiste em continuar o processo iterativo de adição de rótulos à solução atual com os rótulos presentes na melhor solução até o presente momento. Este processo finaliza quando não restarem rótulos que mantenham o grafo desconexo.

Segue descrição do procedimento de extração de uma nova solução do Espaço Complementar:

Algoritmo 2

Definições:

EspCompl = Rótulos no espaço complementar = {Todos os rótulos - rótulos da Solução Best}

Todos = {Todos os rótulos}

Conectividade (C) = número de componentes conexos de C.

Δ = Conectividade (C $\cup$ {v}) - Conectividade (C $\cup$ {s})

T => $T_{|C|+1} = T_{|C|} * \alpha$

T_0 = |BestC| e α = 1/|BestC|

Início

C ← 0

Enquanto (Conectividade (C) > 1) e (EspCompl - C ≠ Ø) **faça**

Seja s $\in$ (EspCompl - C), o rótulo que minimiza Conectividade (C $\cup$ {s})

Para cada v $\in$ (EspCompl – C) **faça**

Calcule a probabilidade P(v) normalizada pela função de Boltzmann: exp(Δ /T)

Selecione aleatoriamente um rótulo não usado u $\in$ (EspCompl – C) de acordo com as probabilidades P(.) tal que Conectividade (C+u) > 1

Caso exista {u} tal que Conectividade (C+u) > 1 então:

Adicione u à solução C: C ← C $\cup$ {u}

Atualize Conectividade (C)

Retire {u} de EspCompl

Caso contrário, sai do loop.

Enquanto (Conectividade (C) > 1) **faça**

Seja s $\in$ (Todos - C), o rótulo que minimiza Conectividade (C $\cup$ {s})

Para cada v $\in$ (Todos - C) **faça**

Calcule a probabilidade P(v) normalizada pela função de Boltzmann: exp(Δ /T)

Selecione aleatoriamente um rótulo não usado u $\in$ (Todos - C) de acordo com as probabilidades P(.) tal que Conectividade (C+u) > 1

Caso exista então:

Adicione u à solução C: C ← C $\cup$ {u}

Atualize Conectividade (C)

Caso contrário, sai do loop.

Fim

2.4 Procedimento de perturbação

Esta fase consiste em gerar uma perturbação na solução corrente para diversificar a gama de soluções e, desta maneira, tentar escapar de um ótimo local. A perturbação consiste em adicionar um rótulo ou retirar um rótulo aleatoriamente. O número de vezes que a solução corrente será perturbada depende do tamanho da vizinhança K. Por exemplo, com K=3 podem ser incluídos 2 rótulos e retirado 1. A vizinhança varia do tamanho de K igual a 1 até KMAX.

O valor de KMAX é auto regulável ao longo da execução do algoritmo. Isto tem como objetivo criar um balanço entre a diversificação e intensificação do processo de busca de novas soluções. O valor de KMAX inicia com número de rótulos do problema diminuído do número de rótulos da melhor solução corrente dividido por 2. Ao final da iteração principal do algoritmo, este valor é incrementado caso a melhor solução corrente não tenha sido melhorada e, decrementado, caso contrário. O valor de teto para KMAX é igual ao |Número de rótulos| - |Número de rótulos da melhor solução|.

Algoritmo 3

Início
Para i ← 1 até K **faça**
  Se random(n) < 0.5 **então** Retira aleatoriamente um rótulo de C'
  Caso contrário Adiciona aleatoriamente um rótulo a C'
Fim

Estas operações podem deixar inviável a solução C' (grafo conexo). O procedimento denominado *Corrige Solução* fará a correção.

2.5 Procedimento Corrige Solução

Este procedimento consiste em retirar aleatoriamente rótulos da solução corrente C' até que a mesma deixe de ser um grafo conexo.

Algoritmo 4
Definições:
Conectividade (C') = número de componentes conexos de C'.
Início
Enquanto (Conectividade (C') = 1) **faça**
  Retira aleatoriamente um rótulo presente em C'
  Calcule novo Conectividade (C')
Fim

2.6 Procedimento de busca local

Caso o número de componentes conexos seja diferente de 1, serão adicionados probabilisticamente rótulos até que não seja possível adicionar um rótulo sem que o mesmo deixe o grafo conexo. Assim como no procedimento do espaço complementar, esta etapa também usa a função de Boltzmann para adicionar probabilisticamente um rótulo.

Algoritmo 5
Definições:
L = Lista de rótulos
Conectividade (C') = número de componentes conexos de C'.
Δ = Conectividade (C' $\cup$ {v}) - Conectividade (C' $\cup$ {s})
T => $T_{|C|+1} = T_{|C|} * \alpha$
T_0 = |BestC| e α = 1/|BestC|

Início
Enquanto (Conectividade (C') > 1) **faça**
Seja s $\in$ (L – C'), o rótulo que minimiza Conectividade (C' $\cup$ {s})
Para cada v $\in$ (L – C') **faça**
Calcule a probabilidade P(v) normalizada pela função de Boltzmann: exp(Δ /T)

Selecione aleatoriamente um rótulo não usado u $\in$ (L – C') de acordo com
as probabilidades P(.) tal que Conectividade (C'+u) > 1
Caso exista então:
Adicione u à solução C': C' $\leftarrow$ C' $\cup$ {u}
Atualize Conectividade (C')
Caso contrário, sai do loop.
Fim

3. Implementação e Propostas

Nesta seção estão descritas todas propostas e implementações testadas.

3.1 Cálculo de conectividade - Union Find

Inicialmente foi utilizado o método de busca em largura (BFS) para o teste de conectividade do grafo. No decorrer dos testes ficou claro que tal procedimento é muito pesado. Como o teste de conectividade é a função mais executada no problema MLC, se fez necessário encontrar um método mais rápido. Por este motivo foi estudado o procedimento Union Find, o qual se mostrou mais adequado para a função.

Foi escolhido o Union Find Weighted + Path Compression. Este é um algoritmo com complexidade O((M+N) lg*N), onde:

M é o número de arestas;
N é o número de vértices;
lg*N é o número de vezes necessárias para levar um número a 1 aplicando repetidamente a operação logarítmica, como ilustrado na tabela 1.

N	lg* N
1	0
2	1
4	2
16	3
65536	4
265536	5

Tabela 1

Em teoria este algoritmo não é linear, mas o é na prática, pois lg*N é uma constante neste universo, haja vista que o maior valor de N para as instâncias do problema é igual a 1000.

Foram implementadas algumas customizações para o problema MLC como, por exemplo, abortar o teste de conectividade tão logo seja detectado que a conectividade é igual a 1.

3.2 Algoritmo proposto

Este algoritmo consiste basicamente de uma solução inicial gerada iterativamente; em seguida as soluções candidatas, a cada iteração do loop principal, são geradas pelo algoritmo de espaço complementar. Para cada solução candidata gerada, são executados os procedimentos de perturbação e busca local por K vezes, no qual K é o tamanho da vizinhança e varia de 1 até KMAX. Tudo isto ocorre dentro de um tempo determinado de acordo com o número de vértices do grafo de entrada.

Algoritmo 6
Definições:
BestC = Melhor solução atual
C = Solução gerada pelo espaço complementar
C' = Solução a ser submetida aos processos de perturbação e busca local
KMAX = Tamanho máximo da vizinhança a ser explorada

Início
BestC ← Gera uma solução inicial
KMAX ← (|número de rótulos| - |BestC|)/ 2

Repete

C ← Extrai uma solução do espaço complementar de BestC
Enquanto (|C| > |BestC|) **faça**
BestC ← C
KMAX ← (|número de rótulos| - |BestC|) / 2
Extrai outra solução do espaço complementar de BestC

K ← 1
Enquanto K < KMAX **faça**
C' ← Perturbação de C (Vizinhança de K)
Se Conectividade (C') = 1 **então** CorrigeSolução
Busca Local em C'
Se |C'| > |C| **então**
C ← C'
Reinicia a vizinhança para K=1
Senão aumenta a vizinhança K:=K+1

Se |C| > |BestC| **então**
BestC ← C
KMAX ← KMAX - 1 (sendo que o piso é (|Número de rótulos| - |BestC|) / 2)
Senão
KMAX ← KMAX+1 (sendo que o teto é |Número de rótulos| - |BestC|)

Até que Condição de parada seja atendida.
Imprimir todos os rótulos que estão ausentes de BestC
Fim

As condições de parada são definidas a seguir:
Para instâncias com 20, 30, 40 e 50 vértices – tempo máximo de execução : 1 segundo.
Para instâncias com 100 vértices – tempo máximo de execução: 20 segundos.
Para instâncias com 200 vértices – tempo máximo de execução: 45 segundos.
Para instâncias com 400 vértices – tempo máximo de execução: 120 segundos.
Para instâncias com 500 vértices – tempo máximo de execução: 500 segundos.
Para instâncias com 1000 vértices – tempo máximo de execução: 4000 segundos.

4. Resultados Computacionais

Os experimentos foram executados em um computador com processador Intel Core I7 2,7 GHz com 32Gb de memória. O algoritmo foi feito, inicialmente, em Delphi (Windows 7) e também em Pascal (Linux Ubuntu). Ao contrário do esperado, a performance do Delphi foi 40% mais rápida, mesmo usando a opção *–O3* de otimização do compilador pascal pra Linux.

Ao formatar o código em pascal tradicional, que requer ordem na declaração de procedimentos e outros detalhes de código, e o transpor novamente pra Delphi, foi notada melhora da performance nesta última plataforma.

Após isso, o código foi transposto para C++ e compilado em pelo G++ do Linux e pelo C++ Builder do Windows. A performance do C++ no Linux foi cerca de 5% superior ao Delphi. A performance no C++ Builder foi a pior entre os 4 cenários.

Os testes iniciais mostraram que as instâncias de baixa densidade apresentam rápida convergência para o resultado. Já as instâncias de alta e média densidade possuem maior complexidade de processamento, pois por possuírem muitas arestas, a conectividade do grafo é maior do que na categoria de baixa densidade.

Foram usadas as instâncias do problema MLST, as quais consistem em 24 diferentes conjuntos de dados, cada um contendo 10 instâncias geradas aleatoriamente, possuindo alto (0,8), o médio (0,5) e baixo (0,2) valores de densidade d, totalizando 720 instâncias.

Para testar a performance e eficiência dos algoritmos apresentados, foi rodada uma avaliação experimental nas instâncias com diferentes números de vértices, densidade do grafo e número de rótulos.

Estas instâncias foram selecionadas com o número de vértices N (variando de 50 até 1000) e o número de rótulos L, enquanto o número de arestas, M, é obtida indiretamente a partir da densidade D. Para cada conjunto de dados, a qualidade da solução é avaliada como o valor médio da função objetivo (isto é, o número de rótulos ausentes da solução) entre as 10 instâncias do conjunto. O tempo máximo de CPU (max CPU-tempo) foi escolhido como condição de parada do algoritmo, determinados com respeito à dimensão do exemplo problema. Por se tratar de um problema novo, o valor max CPU-tempo ainda sofrerá ajustes.

Pela razão citada acima, não há outros experimentos/resultados para fazer comparação. Os tempos que aparecem nas tabelas é o tempo no qual a melhor solução foi encontrada. O valor ‘média’ se refere à média dos resultados das 10 instâncias para cada combinação vértice/cor/densidade.

Vértices	Cores	Densidade	Média	Milissegundos
50	12	0,8	9,8	0,77
		0,5	7,4	1,45
		0,2	2,5	0,91
	25	0,8	15,5	2,60
		0,5	9,9	3,24
		0,2	2,7	1,60
	50	0,8	21,3	6,92
		0,5	11,6	6,59
		0,2	2,8	4,26
	62	0,8	22,7	12,77
		0,5	12,1	6,84
		0,2	2,8	7,27

Tabela 2

Vértices	Cores	Densidade	Média	Milissegundos
100	25	0,8	21	4,57
		0,5	16,5	5,64
		0,2	6,2	4,80
	50	0,8	33,1	21,08
		0,5	22,2	39,24
		0,2	6,8	13,61
	100	0,8	45,2	103,66
		0,5	26,5	144,37
		0,2	7,2	53,05
	125	0,8	48,6	144,68
		0,5	27,1	154,67
		0,2	7,2	81,45

Tabela 3

Vértices	Cores	Densidade	Média	Milissegundos
200	50	0,8	43,3	52,04
		0,5	32,7	108,51
		0,2	13,2	81,07
	100	0,8	68,8	340,05
		0,5	45,4	378,52
		0,2	15	255,01
	200	0,8	93,8	1424,45
		0,5	54,1	1149,16
		0,2	15,9	1126,41
	250	0,8	99,4	2020,43
		0,5	56,5	2052,33
		0,2	16,1	1553,20

Tabela 4

Vértices	Cores	Densidade	Média	Milissegundos
400	100	0,8	88,7	581
		0,5	71,9	1087
		0,2	30,7	1285
	200	0,8	144,3	4041
		0,5	99,5	5112
		0,2	35	5654
	400	0,8	195,7	18840
		0,5	120,3	27101
		0,2	37,4	13771
	500	0,8	210,2	31952
		0,5	124,9	31868
		0,2	38,2	22947

Tabela 5

Vértices	Cores	Densidade	Média	Milissegundos
500	125	0,8	111,4	1547
		0,5	89,2	1589
		0,2	37,1	3110
	250	0,8	178,3	11207
		0,5	123,8	12113
		0,2	41,4	14348
	500	0,8	240,4	53284
		0,5	146,8	75466
		0,2	45	33940
	625	0,8	256,9	61486
		0,5	155,2	66054
		0,2	45,3	51554

Tabela 6

Vértices	Cores	Densidade	Média	Milissegundos
1000	250	0,8	228,8	24692
		0,5	197,2	59115
		0,2	113,8	85739
	500	0,8	375,4	210436
		0,5	284,3	186726
		0,2	133,4	266436
	1000	0,8	514,7	672845
		0,5	353,6	877898
		0,2	145,8	627834
	1250	0,8	552,6	1327128
		0,5	369,7	1470729
		0,2	147	944132

Tabela 7

5. Conclusões e trabalhos futuros

Este trabalho apresentou uma estratégia algorítmica para o problema MLC, a qual integra a meta-heurística VNS com o Simulated Annealing e, usa mecanismos de auto regulagem de parâmetros. Não existem outros resultados conhecidos para o problema.

Como próximos passos, será necessário desenvolver um método exato para poder comparar seus resultados com os obtidos por este estudo e, trabalhar na melhora do procedimento de detecção de conexidade. Além disso, penso em estudar a viabilidade de implantar o procedimento de trocar 1 rótulo por 2 (2-OPT). Será necessário utilizar uma máquina com maior poder de processamento para avaliar a redução dos tempos de cálculo.

Referências

[1] S. Consoli, N. Mladenovi´c, J.A. Moreno-Pérez, Solving the minimum labelling spanning tree problem by intelligent optimization, Applied Soft Computing 28 (2014) 440–452.

[2] Y. Xiong, B. Golden, E. Wasil, Improved heuristics for the minimum labellingspanning tree problem, IEEE Trans. Evol. Comput. 10 (2006) 700–703.

[3] S.O. Krumke, H.C. Wirth, On the minimum label spanning tree problem, Inform.Process. Lett. 66 (1998) 81–85.