UM ALGORITMO BASEADO EM *LATE ACCEPTANCE HILL-CLIMBING* PARA SOLUÇÃO DO PROBLEMA DE *LAYOUT* LINEAR DE CIRCUITOS

Guilherme M. Miranda*, Jonatas B. C. Chagas†, Marcone J. F. Souza†

*Departamento de Engenharia de Controle e Automação
Universidade Federal de Ouro Preto
CEP: 35.400-000, Ouro Preto, MG, Brasil

†Departamento de Ciência da Computação
Universidade Federal de Ouro Preto
CEP: 35.400-000, Ouro Preto, MG, Brasil

Emails: gui.miranda1996@gmail.com, jonatas.chagas@iceb.ufop.br, marcone@ufop.edu.br

Abstract— This work has its focus on a linear circuit layout problem, the Cutwidth Problem. This problem has a set of components of a circuit that communicate to connections previously defined. The objective is to create a linear sequence of components and minimize the number of wires between two elements of the circuit. To solve this problem, a heuristic algorithm based on the Late Acceptance Hill-Climbing (LAHC) metaheuristic is proposed. This algorithm starts from a random initial solution and makes use of a memory structure to explore the solution space. The advantage of this algorithm over other methods based on metaheuristics is that the LAHC has only two parameters to be calibrated, the size of the memory that holds the value of the last solutions generated and the stopping criterion. The algorithm was tested and compared with four algorithms of the literature. The preliminary results show that the method is competitive with the best algorithms of the literature, having its simplicity as advantage.

Keywords— Cutwidth, Metaheuristics, Optimization, Late Acceptance Hill-Climbing

Resumo— Este trabalho tem seu foco em um problema de layout linear de circuitos, o *Cutwidth Problem*. Neste problema há um conjunto de componentes de um circuito que comunicam-se por meio de ligações previamente definidas. O objetivo é sequenciar esses componentes em um layout linear de forma a minimizar a quantidade de ligações que passam entre cada par de componentes da sequência. Para resolvê-lo, propõe-se neste trabalho um algoritmo heurístico baseado na metaheurística *Late Acceptance Hill-Climbing* (LAHC), a qual parte de uma solução inicial aleatória e faz o uso de uma estrutura de memória para explorar o espaço de soluções. A vantagem desse algoritmo sobre outros métodos baseados em metaheurísticas é que o LAHC tem apenas dois parâmetros para serem calibrados, o tamanho da memória que guarda o valor das últimas soluções geradas e o critério de parada. O algoritmo foi testado e comparado com outros quatro algoritmos da literatura. Os resultados computacionais preliminares mostram que o método é competitivo com os melhores algoritmos da literatura, tendo como vantagem sua simplicidade.

Palavras-chave— Layout linear de circuitos, Metaheuristicas, Otimização, Late Acceptance Hill-Climbing

1 Introdução

O *Cutwidth Problem* ou Problema de Layout Linear de Circuitos (PLLC) também referenciado na literatura como *Minimum Cut Linear Arrangement* (Díaz et al., 1997), *Migration Scheduling* (Resende and Andrade, 2012) ou, uma generalização do problema para hipergrafos chamado de *Board Permutation Problem* (Cohoon and Sahni, 1987), trata da geração de um layout linear de um circuito de componentes que se conectam por fios. O PLLC resume-se a encontrar um arranjo linear de vértices, provenientes de um grafo, de forma que o máximo número de arestas ligando dois vértices consecutivos, seja minimizado.

Aplicações práticas do PLLC começaram a aparecer no meio acadêmico na década de 70, quando foi desenvolvido um modelo teórico para estabelecer o número de canais em um layout ótimo de circuito (Adolphson and Hu, 1973). Técnica para identificar *clusters* naturais (isto é, não arbitrários) em hipertextos é outra aplicação do PLLC (Botafogo, 1993). Essa técnica aprimorou a navegação (*browsing*) e a exibição (*display*) de informações. Existem várias outras aplicações para o *Cutwidth Problem*, como engenharia de proteína (Blin et al., 2008), migração de redes (Resende and Andrade, 2012), entre outras.

O PLLC pertence à classe de problemas NP-difíceis, mesmo para grafos pequenos que apresentam vértices de grau maior ou igual a três (Makedon et al., 1985). Assim, as principais abordagens presentes na literatura se baseiam em procedimento heurísticos, embora existam trabalhos que utilizam métodos exatos de forma eficiente para problemas com tamanhos reduzidos.

Entre os procedimentos heurísticos existentes destaca-se: uma junção do GRASP com Path Relinking (Andrade and Resende, 2007b), a combinação de GRASP com *Evolutionary Path Relinking* (Andrade and Resende, 2007a), metaheuristica baseada no método *Simulated Annealing* (Cohoon and Sahni, 1987), *Scatter Search* (Pantrigo et al., 2012), uma Busca Tabu, para auxiliar um algoritmo *Branch-and-bound*

(Palubeckis and Rubliauskas, 2012) e uma nova variante do *Variable Neighbourhood Search*, chamada *Variable Formulation Search* (Pardo et al., 2013).

Andrade and Resende (2007b) trataram duas versões de migração de rede e para resolvê-las apresentaram um algoritmo híbrido que combina GRASP com *Path-Relinking* para encontrar soluções com custos eficientes. Já em (Andrade and Resende, 2007a), a heurística híbrida proveniente de GRASP e *Evolutionary Path-relinking* foi testada para as mesmas aplicações anteriores; porém, os experimentos mostraram que GRASP com *Evolutionary Path-Relinking* encontram soluções mais rapidamente que GRASP com *Path-Relinking*.

Cohoon and Sahni (1987) desenvolveram um algoritmo *Simulated Annealing* baseado em diferentes métodos construtivos e métodos de buscas locais.

Pantrigo et al. (2012) também abordaram o problema tratado neste trabalho utilizando um procedimento metaheurístico. Os autores usaram *Scatter Search* para encontrar soluções para o problema. Esse algoritmo explora o espaço de soluções com um conjunto de pontos de referência. Além disso, ele é baseado no método construtivo GRASP e em uma busca local de movimento de inserção e métodos de combinação baseados em votos. Por meio de experimentos, os autores mostraram que essa heurística supera o *Simulated Annealing* (Cohoon and Sahni, 1987) e o *Evolutionary Path Relinking* (Andrade and Resende, 2007a).

Palubeckis and Rubliauskas (2012) apresentam um algoritmo *Branch-and-Bound* com um teste de dominância que permite reduzir drasticamente a redundância no processo de enumeração. O teste é feito utilizando um método de Busca Tabu.

Pardo et al. (2013) propõem uma nova variação do método *Variable Neighbourhood Search* chamado *Variable Formation Search*. Em problemas de otimização é comum que várias soluções diferentes possuam os valores da função objetivo iguais. Esse algoritmo utiliza formulações alternativas, para definir qual das soluções é mais promissora. Isso é feito criando uma perturbação (*shaking*), seguida de uma busca local e modificação da vizinhança do problema original.

O desenvolvimento de um método de solução eficiente para o PLLC é, portanto, de grande importância. Neste trabalho aborda-se o PLLC pelo método *Late Acceptance Hill-Climbing* – LAHC (Burke and Bykov, 2017). Os resultados do método desenvolvido foram comparados com aqueles dos métodos *Scatter Search* (Pantrigo et al., 2012), *Simulated Annealing* (Cohoon and Sahni, 1987) e GRASP (Andrade and Resende, 2007a).

O restante deste artigo está organizado como segue. Na Seção 2 descreve-se o problema abordado. A Seção 3 apresenta a metodologia abordada, incluindo a forma de representação adotada e de geração de uma solução inicial, a estrutura de vizinhança, a função de avaliação e a adaptação da técnica heurística LAHC ao problema. A Seção 4 mostra os resultados encontrados e a Seção 5 conclui o trabalho e aponta os trabalhos futuros.

2 Descrição do Problema

O PLLC é um problema que consiste em encontrar uma ordem linear de componentes, ligados por fios, de forma a minimizar ao máximo o número de fios entre os dispositivos adjacentes. Essa solução é de grande utilidade em muitas áreas, sendo uma delas a de construção de chips e materiais eletrônicos, podendo trazer, como consequência, a diminuição ainda maior dos tamanhos desses equipamentos.

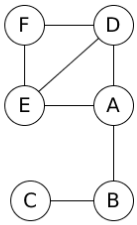

Figura 1: Grafo

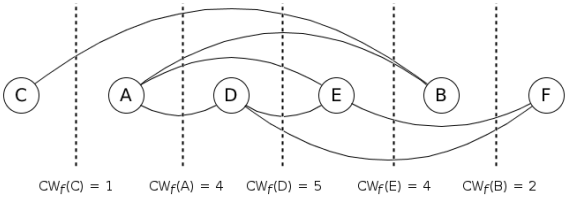

Figura 2: Solução 1

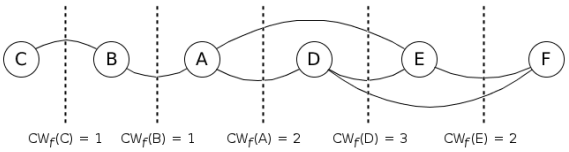

Figura 3: Solução 2

Para entendê-lo melhor, pode-se observar na Figura 1 um exemplo de um grafo com 5 vértices (A, B, C, D e E) e na Figura 2 a representação de uma solução s desse grafo, dada pelo arranjo linear (C, A, D, E, B, F).

O número de arestas que passam por dois vértices consecutivos varia a cada par, sendo que o maior número de arestas ocorre na ligação do vértice D com o vértice E, com 5 ligações. Diz-se, assim, que a função objetivo dessa solução s tem valor $f(s) = 5$, ou seja, o *cutwidth* dessa solução é 5.

Fazendo-se algumas alterações na sequência de vértices, pode-se chegar a uma solução com um valor melhor da função objetivo. Por exemplo, na Figura 3 tem-se uma solução com *cutwidth* de valor 3.

Em outras palavras, uma vez que as ligações entre os vértices são predefinidas pelo grafo, o problema a ser resolvido consiste em determinar a sequência linear dos vértices, de forma a minimizar o número máximo de ligações que passam por dois vértices consecutivos.

3 Metodologia

Este trabalho trata o PLLC pelo método *Late Acceptance Hill-CLimbing* – LAHC (Burke and Bykov, 2017). Na Seção 3.1 é descrita a forma de representação da solução do problema; na Seção 3.2, o procedimento de geração da solução inicial; na Seção 3.3, a estrutura de vizinhança usada para explorar o espaço de soluções do problema; na Seção 3.4 é mostrada como uma solução é avaliada e, na Seção 3.5, a adaptação do LAHC ao PLLC.

3.1 Representação de uma solução

Para formar um layout linear é preciso saber o número de componentes do circuito, quantas ligações são realizadas e qual componente é ligado a qual componente. A forma utilizada para representar o circuito foi criar um vetor de n posições, sendo n igual ao número de componentes no circuito, em que cada posição do vetor recebe um componente. Os componentes são inseridos na ordem da solução encontrada.

A solução 1 mostrada na Figura 2, por exemplo, é representada pela Figura 4:

C	A	D	E	B	F

Figura 4: Representação computacional da solução 1

3.2 Geração da Solução Inicial

A solução inicial é obtida de forma randômica, ou seja, cada componente é alocado em uma posição aleatória no *layout* linear.

3.3 Estrutura de Vizinhança

Para explorar o espaço de soluções do problema, utiliza-se o movimento de troca da posição de um componente i pela posição de outro componente j distinto. Esse movimento define a vizinhança $N(s)$ de uma dada solução s. O conjunto de todas as soluções s' geradas a partir de s por meio de movimentos de troca define a vizinhança $N^(T)_(s)$. Como pode ser observado na Figura 5, o componente A troca de posição com o componente C.

C	A	D	E	B	F

A	C	D	E	B	F

Figura 5: Exemplo de aplicação do movimento

3.4 Função de Avaliação

A função de avaliação considerada neste trabalho, e que deve ser minimizada, é o maior número de fios que passam por dois componentes consecutivos.

Para calcular a função objetivo, os dados do grafo e os dados da solução são carregados em uma espécie de dicionário, em que existem seções que representam cada componente e dentro de cada seção tem o conteúdo, representando as ligações que esse componente realiza. É criada, também, uma memória para guardar os itens que já foram analisados.

O cálculo é feito em partes. Escolhe-se uma seção do dicionário (um componente) e analisa-se o seu conteúdo (isto é, com quem ele tem ligação). Se o item tem ligação com outro e ele não está na memória, então é somado 1 ao valor da função objetivo dessa seção. Caso esse item esteja na memória, é subtraído o valor 1. Por fim, a função objetivo da solução é o maior valor entre todos os calculados.

3.5 Late Acceptance Hill-Climbing

Late Acceptance Hill-CLimbing (Burke and Bykov, 2017) é uma metaheuristica desenvolvida com base na heurística *Hill-Climbing* (HC).

HC é um método iterativo, ou seja, a solução corrente é utilizada para definir se a solução vizinha será aceita ou não. Por só aceitar soluções melhores, o HC fica preso no primeiro ótimo local encontrado.

Para não ficar preso nesse ótimo local e, então, expandir o alcance da busca, o LAHC compara a solução vizinha com alguma solução anterior e não apenas com a última, tal como no HC. Para cumprir esse objetivo, o LAHC utiliza uma memória que armazena o valor das últimas l soluções encontradas. Com essa estratégia, o algoritmo também aceita soluções de piora e, portanto, não fica preso em ótimos locais.

O pseudocódigo do LAHC implementado neste trabalho é o descrito pelo Algoritmo 1, conforme notação de Toffolo et al. (2018).

Para implementar o LAHC, dois parâmetros devem ser calibrados, o tamanho da memória e o critério de parada, e três argumentos são necessários, uma solução inicial s, o tamanho l da memória e o número máximo de soluções consecutivas rejeitadas *iterMax*.

Na linha 1 do Algoritmo 1 é criado um laço de repetição que percorre todas as l posições do vetor F, para então preencher todas as posições com o valor da solução inicial, $f(s)$. F é a memória que

Algoritmo 1 Late Acceptance Hill-Climbing

Entrada: Solução inicial s, tamanho l da memória e número máximo de rejeições consecutivas *IterMax*.

Saída: $s^\star$ (Melhor solução encontrada durante toda a execução do algoritmo).

LAHC(s, l, *IterMax*):

```
1: Para cada  p ∈ {0, ..., l − 1} faça
2:    F_p ← f(s)
3: Fim
4: p ← 0, s* ← s, iter ← 0
5: Enquanto iter < iterMax faça
6:    s' ← vizinho aleatório de s
7:    Se f(s') ≤ f(s) ou f(s') ≤ F_p então faça
8:       Se f(s') < f(s) então faça
9:          iter ← 0
10:      Fim
11:      s ← s'
12:      Se f(s') < f(s*) então faça
13:         s* ← s'
14:      Fim
15:   F_p ← f(s)
16:   p ← (p + 1) mod l
17:   iter ← iter + 1
18: Fim
19: Retorna s*
```

armazena os valores das l últimas soluções analisadas. Experimentalmente, l foi fixado em 100 vezes o valor de n, sendo n o número de componentes do circuito analisado. Na linha 4 é inicializada uma variável p para controlar a entrada e saída de informação da memória. Ainda na linha 4, a variável $s^\star$, que representa a melhor solução encontrada até então, recebe a solução inicial (conforme descrito na Seção 3.2) e a variável utilizada para controlar o número de iterações consecutivas sem melhora, *iter*, é atribuído o valor 0 (zero).

Após inicializar todas as variáveis necessárias, o método entra em um laço, dado pelas linhas 5 a 18, que é executado enquanto o critério de parada do método não for alcançado. O algoritmo é interrompido após *Itermax* soluções consecutivas não aceitas, ou seja, o algoritmo funcionará até que *Itermax* soluções consecutivas não obedecerem à condição imposta na linha 8.

Na linha 6 é gerada aleatoriamente uma solução vizinha s' da solução corrente s na estrutura de vizinhança considerada, isto é, naquela gerada pelo movimento de troca simples. Se a solução vizinha s' for melhor que a solução atual s ou então melhor que a solução armazenada na posição p da memória (F_p) então é verificado na linha 7 se essa solução vizinha melhora a solução corrente. Em caso positivo, isto é, se $f(s')$ for menor que $f(s)$, o contador de iterações consecutivas sem melhora na solução atual, dado pela variável *iter*, é zerado na linha 9. Independentemente de a solução s' ser melhor ou pior que s, a solução vizinha s' torna-se a solução corrente s na linha 11, sendo ela armazenada na posição p da memória na linha 15. Caso a solução vizinha s' também seja melhor do que a melhor solução gerada até então, dada por $s^\star$, ela torna-se a melhor solução encontrada até o momento, linha 13.

Na linha 16, a variável p é incrementada ao final de todo o ciclo. Observa-se que, quando p for igual l, a variável p recebe o valor 0 (zero). Dessa forma, as soluções aceitas no método serão alocadas na primeira posição da memória até a última. Quando chegar na última posição, a próxima solução aceita substituirá a solução presente na primeira posição. Por fim, o contador de iterações (*iter*) é incrementado na linha 17.

4 Resultados

O algoritmo desenvolvido foi implementado na linguagem C++ usando o compilador GNU GCC Compiler e testado em microcomputador i7 7500U, 2.7 GHz, com 8 GB de RAM sob o sistema operacional Windows 10.

Inicialmente, foram realizados testes para calibrar o critério de parada e o tamanho da memória do método LAHC. Foi implementado, como critério de parada, $itermax = 20000$ repetições consecutivas sem passar pelo teste de melhora atrasada. O tamanho da memória, isto é, $|F|$ foi definido como 100 vezes o número de vértices do problema.

Para avaliar o algoritmo foi utilizado o conjunto de instâncias chamado *Small*, apresentado por Martí et al. (2008). Esse conjunto contém 84 grafos que foram apresentados no contexto do *Bandwidth Reduction Problem*. O número de vértices varia de 16 a 24 e o número de arestas de 18 a 49.

Os experimentos foram realizados com as 84 instâncias, sendo que cada uma delas foi executada 10 vezes.

A Tabela 1 mostra os resultados dos experimentos nas 84 instâncias desse conjunto. Nesta tabela, a primeira coluna indica a instância; a segunda coluna (BKS), o valor da melhor solução conhecida para essa instância; a terceira coluna (BSLAHC), o valor da melhor solução encontrada com o método LAHC nas 10 execuções; a quarta coluna (MLAHC), a média dos resultados obtidos pelo LAHC nessas 10 execuções; a quinta coluna (GBS), o *gap* entre o valor da melhor solução encontrada pelo LAHC e o valor da melhor solução da literatura, no caso, os valores ótimos, conforme mostrado em Coudert (2016); na sexta coluna (GSM), o *gap* entre o resultado médio do LAHC e o valor da melhor solução da literatura. Na sétima e última coluna tem-se o tempo demandado, em segundos, para resolver cada instância.

Tabela 1: Comparação de resultados do LAHC × melhores resultados conhecidos das instâncias *Small*

Instância	BKS	BSLAHC	MLAHC	GBS(%)	GSM(%)	Tempo(s)
p17_16_24	7	7	7,0	0,0	0,0	0,75
p18_16_21	5	5	5,0	0,0	0,0	0,68
p19_16_19	4	4	4,1	0,0	2,5	0,72
p20_16_18	4	4	4,1	0,0	2,5	0,61
p21_17_20	4	4	4,4	0,0	10,0	0,72
p22_17_19	4	4	4,0	0,0	0,0	0,75
p23_17_23	5	5	5,4	0,0	8,0	0,85
p24_17_29	8	8	8,1	0,0	1,3	0,96
p25_17_20	4	4	4,4	0,0	10,0	0,73
p26_17_19	4	4	4,0	0,0	0,0	0,66
p27_17_19	4	4	4,0	0,0	0,0	0,78
p28_17_18	3	3	3,6	0,0	20,0	0,68
p29_17_18	3	3	3,0	0,0	0,0	0,89
p30_17_19	4	4	4,0	0,0	0,0	0,84
p31_18_21	3	4	4,0	33,3	33,3	0,79
p32_18_20	4	4	4,3	0,0	7,5	0,76
p33_18_21	4	4	4,1	0,0	2,5	0,86
p34_18_21	4	4	4,2	0,0	5,0	0,82
p35_18_19	3	3	3,9	0,0	30,0	0,76
p36_18_20	4	4	4,5	0,0	12,5	0,75
p37_18_20	4	5	5,0	25,0	25,0	0,70
p38_18_19	3	3	3,0	0,0	0,0	0,91
p39_18_19	3	3	3,8	0,0	26,7	0,71
p40_18_32	8	8	8,2	0,0	2,5	1,05
p41_19_20	3	3	3,9	0,0	30,0	0,76
p42_19_24	5	5	5,0	0,0	0,0	0,93
p43_19_22	4	4	4,2	0,0	5,0	0,91
p44_19_25	6	6	6,0	0,0	0,0	0,99
p45_19_25	5	5	5,4	0,0	8,0	1,05
p46_19_20	3	3	3,3	0,0	10,0	0,91
p47_19_21	4	4	4,0	0,0	0,0	0,83
p48_19_21	4	4	4,1	0,0	2,5	0,82
p49_19_22	4	4	4,3	0,0	7,5	0,88
p50_19_25	4	4	5,0	0,0	25,0	1,01
p51_20_28	6	6	6,0	0,0	0,0	1,20
p52_20_27	5	5	5,4	0,0	8,0	1,10
p53_20_22	4	4	4,1	0,0	2,5	0,91
p54_20_28	6	6	6,4	0,0	6,7	1,15
p55_20_24	4	4	4,9	0,0	22,5	1,00
p56_20_23	5	5	5,0	0,0	0,0	0,87
p57_20_24	4	4	4,6	0,0	15,0	1,04
p58_20_21	3	4	4,0	33,3	33,3	0,88
p59_20_23	4	4	4,0	0,0	0,0	0,93
p60_20_22	4	4	4,3	0,0	7,5	0,99
p61_21_22	4	4	4,2	0,0	5,0	0,88
p62_21_30	7	7	7,2	0,0	2,9	1,29
p63_21_42	12	12	12,0	0,0	0,0	1,75
p64_21_22	3	4	4,0	33,3	33,3	0,97
p65_21_24	4	4	4,5	0,0	12,5	1,16
p66_21_28	6	6	6,0	0,0	0,0	0,96
p67_21_22	3	4	4,1	33,3	36,7	0,90
p68_21_27	6	6	6,0	0,0	0,0	1,09
p69_21_23	5	5	5,1	0,0	2,0	0,79
p70_21_25	5	5	5,7	0,0	14,0	0,97
p71_22_29	5	5	5,6	0,0	12,0	1,33
p72_22_49	14	14	14,8	0,0	5,7	1,72
p73_22_29	5	5	5,1	0,0	2,0	1,46
p74_22_30	6	6	6,2	0,0	3,3	1,23
p75_22_25	5	5	5,0	0,0	0,0	1,03
p76_22_30	6	6	6,3	0,0	5,0	1,28
p77_22_37	8	8	8,4	0,0	5,0	1,41
p78_22_31	6	6	6,8	0,0	13,3	1,30
p79_22_29	5	5	5,0	0,0	0,0	1,25
p80_22_30	5	5	5,5	0,0	10,0	1,36
p81_23_46	13	13	13,1	0,0	0,8	1,88
p82_23_24	4	5	5,0	25,0	25,0	1,00
p83_23_24	4	4	4,9	0,0	22,5	0,95
p84_23_26	4	4	4,4	0,0	10,0	1,08
p85_23_26	4	4	4,5	0,0	12,5	1,14
p86_23_24	3	4	4,1	33,3	36,7	1,06
p87_23_30	6	6	6,5	0,0	8,3	1,25
p88_23_26	4	4	4,2	0,0	5,0	1,31
p89_23_27	5	5	5,2	0,0	4,0	1,28
p90_23_35	7	7	7,0	0,0	0,0	1,55
p91_24_33	6	6	6,9	0,0	15,0	1,42
p92_24_26	4	4	4,9	0,0	22,5	1,32
p93_24_27	4	4	5,2	0,0	30,0	1,24
p94_24_31	6	6	6,4	0,0	6,7	1,31
p95_24_27	4	4	4,9	0,0	22,5	1,28
p96_24_27	4	4	4,8	0,0	20,0	1,29
p97_24_26	4	5	5,2	25,0	30,0	1,04
p98_24_29	5	5	6,1	0,0	22,0	1,25
p99_24_27	5	5	5,0	0,0	0,0	1,34
p100_24_34	7	7	7,4	0,0	5,7	1,23

[1]Testes realizados em um computador Intel i7 7500U, 2.7 GHz, com 8 GB de RAM.

Na Tabela 1, o *gap* das soluções médias produ- zidas pelo algoritmo LAHC é calculado pela Equa-

ção (1):

$$GSM = 100 \times \frac{(MSLAHC - BKS)}{BKS} \qquad (1)$$

em que *MSLAHC* é o valor médio das soluções obtidas pelo algoritmo em 10 execuções em uma dada instância e *BKS* é o valor da melhor solução existente na literatura para a instância considerada.

Já o *gap* da melhor solução produzida pelo algoritmo LAHC é calculado pela Equação (2):

$$GBS = 100 \times \frac{BSLAHC - BKS}{BKS} \qquad (2)$$

em que *BSLAHC* é o valor da melhor solução obtida pelo algoritmo em 10 execuções em uma dada instância e *BKS* é o valor da melhor solução existente na literatura para a instância considerada.

A Tabela 2 sintetiza os resultados obtidos pelo algoritmo LAHC, comparando-os com os de outros algoritmos da literatura. Nessa tabela, a coluna SS indica o método *Scatter Search* de Pantrigo et al. (2012); a coluna SA, o método *Simulated Annealing* de Cohoon and Sahni (1987); a coluna GRP, o método GRASP de Andrade and Resende (2007a); a coluna VFS, o método *Variable Formulation Search* de Pardo et al. (2013) e na última coluna, o LAHC. Na segunda linha são apresentadas as médias dos valores da função de avaliação de cada algoritmo, na terceira linha são mostrados os *gap*s médios de cada algoritmo e na última linha é indicado quantos ótimos cada método encontrou. Os resultados desses métodos da literatura são aqueles relatados em Pantrigo et al. (2012) e Pardo et al. (2013). Como estes últimos autores não forneceram as informações sobre os valores médios da função objetivo e o *gap* nas instâncias *Small*, colocou-se um sinal de traço (-) na célula.

Tabela 2: Comparação dos resultados médios do LAHC com aqueles de outros algoritmos da literatura

	SS	SA	GPR	VFS	LAHC
V. Médio	4,92	5,15	5,2	-	5,00
Gap(%)	0,00%	5,60%	6,56%	-	2,90%
Ótimos	84	64	60	84	76

Pela Tabela 2 verifica-se que o LAHC, com os parâmetros adotados e a implementação realizada, é competitivo com os melhores algoritmos da literatura. De fato, a média dos valores médios da função de avaliação do método LAHC é menor que nos métodos SA e GPR, e apenas um pouco inferior à do método SS. Já com relação à capacidade de encontrar os ótimos globais e ao *gap*, observa-se que o LAHC foi capaz de encontrar 76 ótimos globais dentre as 84 instâncias, tendo um *gap* médio de 2,90%. Nesses quesitos o LAHC só tem desempenho inferior ao dos métodos SS e VFS, os quais foram capazes de encontrar todos os ótimos globais.

5 Conclusões

Este trabalho apresentou uma aplicação da metaheurística *Late Acceptance Hill-Climbing* (LAHC) para a solução do *Cutwidth Problem*. O algoritmo LAHC desenvolvido utiliza o movimento de troca para explorar o espaço de soluções do problema.

Foi feita uma comparação do algoritmo LAHC com os quatro melhores métodos da literatura para este problema. Os resultados mostraram que o LAHC é competitivo e só tem desempenho inferior aos dos métodos baseados em *Scatter Search* e *Variable Formulation Search*. A vantagem do método proposto é que sua implementação é muito simples e requer apenas a calibração de dois parâmetros, sendo um o critério de parada.

Como trabalhos futuros é necessário testar o algoritmo proposto em instâncias de dimensões mais elevadas, bem como calibrar seus dois parâmetros usando um projeto de experimentos (DOE, *Design of Experiments*). Para essa tarefa sugere-se a utilização do pacote IRACE (López-Ibáñez et al., 2016), usado para calibração automática de parâmetros de algoritmos de otimização, disponível em `http://iridia.ulb.ac.be/irace/`. Além disso, propõe-se implementar outras estruturas de vizinhança, como por exemplo, o movimento de realocação e estudar melhor as propriedades do problema, de forma a evitar a análise de soluções não promissoras.

Agradecimentos

Os autores agradem à FAPEMIG, ao CNPq e à UFOP o apoio dado ao desenvolvimento da presente pesquisa.

Referências

Adolphson, D. and Hu, T. C. (1973). Optimal linear ordering, *SIAM Journal on Applied Mathematics* **25**(3): 403–423. Disponível em `http://www.jstor.org/stable/2100112`, acesso em 01/10/2017.

Andrade, D. V. and Resende, M. G. C. (2007a). Grasp with evolutionary path-relinking, *Proceedings of the Seventh Metaheuristics International Conference (MIC)*, Montreal, Canadá. Disponível em `http://mauricio.resende.info/doc/gevpr.pdf`, acesso em 01/10/2017.

Andrade, D. V. and Resende, M. G. C. (2007b). Grasp with path-relinking for network migration scheduling, *Proceedings of the In-*

ternational Network Optimization Conference (INOC 2007), Spa, Bélgica. Disponível em `http://mauricio.resende.info/doc/netmigr.pdf`, acesso em 01/10/2017.

Blin, G., Fertin, G., Hermelin, D. and Vialette, S. (2008). Fixed-Parameter Algorithms For Protein Similarity Search Under mRNA Structure Constraints, *Journal of Discrete Algorithms* **6**(4): 618–626.

Botafogo, R. A. (1993). Cluster analysis for hypertext systems, *Proceedings of the 16th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '93, ACM, New York, NY, USA, pp. 116–125. Disponível em `http://doi.acm.org/10.1145/160688.160704`, acesso em 01/10/2017.

Burke, E. K. and Bykov, Y. (2017). The late acceptance hill-climbing heuristic, *European Journal of Operational Research* **258**(1): 70–78.

Cohoon, J. P. and Sahni, S. (1987). Heuristics for backplane ordering, *Advances in VLSI and Computer Systems* **2**(1-2): 37–60.

Coudert, D. (2016). A note on Integer Linear Programming formulations for linear ordering problems on graphs, *Research report*, Inria. Disponível em `https://hal.inria.fr/hal-01271838`, acesso em 13/04/2018.

Díaz, J., Gibbons, A., Pantziou, G. E., Serna, M. J., Spirakis, P. G. and Toran, J. (1997). Parallel algorithms for the minimum cut and the minimum length tree layout problems, *Theoretical Computer Science* **181**(2): 267–287. Computing and Combinatorics.

López-Ibáñez, M., Dubois-Lacoste, J., Cáceres, L. P., Birattari, M. and Stützle, T. (2016). The irace package: Iterated racing for automatic algorithm configuration, *Operations Research Perspectives* **3**: 43–58.

Makedon, F. S., Papadimitriou, C. H. and Sudborough, I. H. (1985). Topological bandwidth, *SIAM Journal on Algebraic Discrete Methods* **6**(3): 418–444.

Martí, R., Campos, V. and Piñana, E. (2008). A branch and bound algorithm for the matrix bandwidth minimization, *European Journal of Operational Research* **186**(2): 513–528.

Palubeckis, G. and Rubliauskas, D. (2012). A branch-and-bound algorithm for the minimum cut linear arrangement problem, *Journal of Combinatorial Optimization* **24**(4): 540–563.

Pantrigo, J. J., Martí, R., Duarte, A. and Pardo, E. G. (2012). Scatter search for the cutwidth minimization problem, *Annals of Operations Research* **199**(1): 285–304.

Pardo, E. G., Mladenović, N., Pantrigo, J. J. and Duarte, A. (2013). Variable formulation search for the cutwidth minimization problem, *Applied Soft Computing* **13**(5): 2242–2252.

Resende, M. and Andrade, D. (2012). Method and system for network migration scheduling. United States Patent 8,139,502.

Toffolo, T. A., Christiaens, J., Van Malderen, S., Wauters, T. and Vanden Berghe, G. (2018). Stochastic local search with learning automaton for the swap-body vehicle routing problem, *Computers & Operations Research* **89**: 68–81.