12º Congresso de Inovação, Ciência e Tecnologia do IFSP - 2021

APLICAÇÃO DA META-HEURÍSTICA LARGE NEIGHBORHOOD SEARCH NA RESOLUÇÃO DO PROBLEMA DE GERAÇÃO E ORGANIZAÇÃO DE GRADE HORÁRIA DO IFSP – CAMPUS BIRIGUI

TIAGO JÚNIO TEGON NASCIMENTO[1], BRUNO RAFAEL GAMINO[2]

[1] Graduando em Engenharia da Computação, PIVICT, IFSP, Campus Birigui, tiago.tegon@aluno.ifsp.edu.br.
[2] Doutor em Engenharia Elétrica, Professor, IFSP, Campus Birigui, gamino.bruno@ifsp.edu.br.

Área de conhecimento (Tabela CNPq): 3.08.02.02-4 Programação Linear, Não-Linear, Mista e Dinâmica

RESUMO: O presente trabalho tem por objetivo desenvolver e implementar computacionalmente o algoritmo *Large Neighborhood Search* (LNS) para resolver o problema de geração e organização de grade horária do Instituto Federal de São Paulo (IFSP), Campus Birigui, na área da indústria e avaliar o desempenho dessa meta-heurística para o problema abordado. A elaboração da grade horária é realizada a cada semestre, por ser um problema que envolve diversas restrições, como a disponibilidade de professores e salas de aula, trata-se de um processo que consome muito tempo, quando realizado de forma manual. Nesse cenário, o algoritmo LNS desenvolvido consiste em, a partir de uma solução factível, obtida por uma busca local, utilizar um método de destruição adaptativo e reparação aleatória para encontrar uma solução ótima de forma automática. O algoritmo conseguiu igualar ao valor da função objetivo da grade horária obtida de forma manual e em alguns testes obteve um valor menor, cumprindo o objetivo proposto.

PALAVRAS-CHAVE: *Large Neighborhood Search*; meta-heurística; algoritmo; grade horária.

APPLICATION OF THE LARGE NEIGHBORHOOD SEARCH METAHEURISTIC IN SOLVING THE IFSP – CAMPUS BIRIGUI TIMETABLE GENERATION AND ORGANIZATION PROBLEM

ABSTRACT: This work aims to develop and implement computationally the algorithm Large Neighborhood Search (LNS) to solve the problem of generation and organization of the timetable of the Federal Institute of São Paulo (IFSP), Birigui Campus, in the area of industry and evaluate the performance of this meta-heuristic for the problem approached. The elaboration of the timetable is done every semester, and because it is a problem involving several constraints, such as the availability of teachers and classrooms, it is a time-consuming process when done manually. In this scenario, the LNS algorithm developed consists in, starting from a feasible solution obtained by a local search, use an adaptive destruction method and random repair to find an optimal solution automatically. The algorithm was able to match the value of the objective function of the timetable obtained manually and, in some tests, obtained a lower value, meeting the proposed objective.

KEYWORDS: Large Neighborhood Search; meta-heuristic; algorithm; timetable.

INTRODUÇÃO

As instituições de ensino devem elaborar sua grade horária escolar a cada semestre, de forma a evitar conflitos de horários entre professores e entre disciplinas e, quando possível, atendendo as preferências.

Segundo Hamawaki (2005), a solução do problema para o gerador de grade horária consiste em gerar uma tabela de horários, objetivando minimizar os conflitos, maximizar preferências e condensar horários de professores e alunos. Pelo fato desse problema admitir uma grande variedade de formulações, é necessário que cada instituição tenha uma definição precisa do problema específico. Em problemas como esse, métodos exatos provaram ser ineficientes, portanto, a aplicação de algoritmos heurísticos e meta-heurísticos tem sido adotada (ADEWUMI; SAWYERR; ALI, 2009).

A meta-heurística *Large Neighborhood Search* (LNS) foi proposta por Shaw (1998) e consiste em fazer movimentos como a busca local, porém esses movimentos podem ser mais poderosos, levando a busca mais longe a cada passo. Quando há possibilidade de movimentos de longo alcance, geralmente há também muito mais deles disponíveis do que se as mudanças fossem muito localizadas. De acordo com Pisinger e Ropke (2010), a vizinhança no LNS é definida através do método de destruir e reparar.

No caso do IFSP, Campus Birigui, a grade horária deve obedecer a algumas restrições: aulas que precisam de salas específicas (laboratórios de informática, eletrônica e outros), disponibilidade dos dias que cada professor tem para ministrar aulas e divisão de uma turma em duas para realização de aulas práticas. A geração e organização da grade horária é uma tarefa que consome muito tempo e esforço, quando realizada de forma manual. Com isso, torna-se necessário o desenvolvimento de um algoritmo utilizando a meta-heurística LNS para a realização dessa tarefa de forma eficiente e automatizada.

MATERIAL E MÉTODOS

Para realização do trabalho, foram coletados os dados referentes à disponibilidade dos professores da área da indústria no primeiro semestre de 2019, os componentes curriculares dos cursos Técnico em Automação Industrial e Tecnologia em Mecatrônica Industrial, além da grade horária utilizada no primeiro semestre de 2019.

Todos os dados foram analisados e tabelados com o Excel, onde cada tipo de dado foi convertido em um número. Para o curso Técnico em Automação Industrial, as disciplinas em que não era necessário dividir a turma foram numeradas de 1 a 27 e as disciplinas que era preciso dividir a turma, chamadas de duplas, de 500 a 509. Já para o curso Tecnologia em Mecatrônica Industrial, as disciplinas normais foram numeradas de 100 a 121 e as duplas de 600 a 603.

As salas de uso específico foram numeradas com seu respectivo número utilizado no campus, exceto a sala chamada de INF, que recebeu o número 150. As salas teóricas foram representadas pelo número 1. Para os professores foram considerados os da área da indústria em ordem alfabética, numerados de 1 a 13 e os que não eram dessa área na ordem que apareciam na grade horária coletada, de 14 a 22. Horários em que não tinham aulas receberam o valor -1.

Para a codificação foi utilizada a ferramenta Octave, onde foi desenvolvido um algoritmo para criação da grade horária aleatória servindo de entrada para o método, foi inserido também a grade horária coletada do primeiro semestre de 2019 e codificada a função objetivo do algoritmo.

A função objetivo é composta por dois tipos de restrições: restrições *hard*, são restrições que devem ser cumpridas para tornar a solução factível, e restrições *soft*, que estão relacionadas a preferências. As restrições *hard* são: (1) o mesmo professor não pode ministrar diferentes disciplinas no mesmo horário; (2) uma mesma sala não pode ser utilizada no mesmo horário por disciplinas diferentes. As restrições *soft* são: (1) disponibilidade do professor no dia em que foi determinado; (2) cada professor com, no mínimo, um dia sem aulas. Em seu cálculo, as restrições *hard* têm o peso 1000 e as *soft* possuem peso 25.

A implementação do método LNS também foi feita na ferramenta Octave, onde dividiu-se em três etapas. Na primeira etapa desenvolveu-se um algoritmo de busca local para encontrar uma solução inicial factível, na segunda o método de destruição e na terceira o método de reparação.

O algoritmo LNS utilizado funciona conforme o pseudocódigo apresentado a seguir:

1. **Inicialização:** Cria uma solução randômica;
2. Encontra uma solução inicial factível *x* através da busca local;
3. Considera a solução inicial como a melhor solução encontrada (*x_melhor = x*);

4. **Repita** os passos 5 a 7 até que o critério de parada seja satisfeito;
5. **Repita** o passo 6 até que a solução encontrada (*x_t*) seja melhor que a solução melhor (*x_melhor*);
6. Destrói e repara a solução: *x_t* = r(d(*x_melhor*));
7. Considera a solução temporária como a melhor solução encontrada (*x_melhor* = *x_t*);
8. **Retorna** a melhor solução encontrada (*x_melhor*);
9. **Fim.**

O critério de parada utilizado foi que a função objetivo da melhor solução encontrada deve ser melhor que a da grade horária usada no primeiro semestre de 2019 ou o limite de iterações seja atingido.

O algoritmo de busca local implementado é baseado no algoritmo utilizado por Neto e Vianna (2011), que procura melhorar o procedimento de busca local com a geração de novas soluções a partir de perturbações, juntamente com uma lista contendo todas as aulas que pioram a função objetivo para guiar a busca local. De modo geral, a busca local consiste em selecionar aleatoriamente uma das aulas da lista e uma outra aula aleatória da mesma turma, realizar a troca e verificar se a troca melhorou ou não a solução, se melhorou, então a troca se mantém, se não houver melhoria, a troca é desfeita. Com relação a perturbação, escolhe-se aleatoriamente duas aulas da mesma turma, realizando a troca e avaliando a solução, se a troca não piorou a solução, a troca é mantida e se piorou é desfeita. O pseudocódigo da etapa de busca local é apresentado a seguir:

1. Calcula a função objetivo da solução inicial;
2. Gera uma lista (*l*) contendo todos os elementos que pioram a função objetivo;
3. **Repita** os passos 4 a 14 até que a solução seja factível;
4. Seleciona randomicamente um elemento da lista (*l*);
5. Seleciona randomicamente um elemento da mesma turma que o anterior;
6. Realiza a troca;
7. **Se** a troca melhorou a função objetivo: mantém a troca e atualiza a lista (*l*);
8. **Se não:** desfaz a troca;
9. **Fim se;**
10. **Se** não houver melhoria em 10 iterações realiza uma perturbação:
11. **Se** a perturbação não piorou a função objetivo: mantém a perturbação e gera novamente a lista (*l*) contendo todos os elementos que pioram a função objetivo;
12. **Se não:** desfaz a perturbação;
13. **Fim se;**
14. **Fim se;**
15. **Fim.**

Para o método de destruição foram realizadas 4 etapas baseadas no trabalho de Kiefer, Hartl e Schnell (2017). Na primeira, o método destrói todas as aulas que pioram a função objetivo. Na segunda, o método destrói randomicamente uma aula de cada turma. A terceira etapa só é realizada se não houve melhoria nas duas etapas anteriores e consiste em escolher aleatoriamente um dia e uma sala presente nesse dia, na sequência, todas as aulas que utilizam essa sala no dia escolhido são destruídas. A quarta etapa também é executada apenas se não houve melhoria nas três etapas anteriores, nela é escolhido randomicamente um professor e todas as aulas desse professor são destruídas. As duas primeiras etapas sempre ocorrem, sendo que todas as aulas destruídas e suas respectivas posições são armazenadas nos vetores de elementos destruídos e posições destruídas. O pseudocódigo da etapa de destruição é apresentado a seguir:

1. Destrói todos os elementos que pioram a função objetivo;
2. Armazena todos os elementos destruídos e suas posições em vetores;
3. **Repita** os passos 4 a 7 até que uma aula seja destruída em cada turma (*x*);
4. Seleciona randomicamente uma aula da turma (*x*);
5. Destrói a solução;
6. Armazena a aula destruída e sua posição nos vetores de elementos destruídos e posições destruídas, respectivamente;
7. Incrementa *x* (vai para a próxima turma);
8. **Se** não houver melhoria em 200 iterações realiza adaptação:

9. Seleciona um dia (*d*) randomicamente;
10. Seleciona uma sala (*s)* randomicamente desse dia (*d*);
11. **Repita** os passos 12 e 13 até que todas as aulas que utilizam sala (*s*) no dia (*d*) sejam destruídas;
12. Destrói a solução;
13. Armazena a aula destruída e sua posição nos vetores de elementos destruídos e posições destruídas, respectivamente;
14. **Fim se;**
15. **Se** não houver melhoria em 200 iterações utilizando a adaptação anterior, realiza outra adaptação:
16. Seleciona um professor (*p*) aleatoriamente;
17. **Repita** os passos 18 e 19 até que todas as aulas que o professor (*p*) ministra sejam destruídas;
18. Destrói a solução;
19. Armazena a aula destruída e sua posição nos vetores de elementos destruídos e posições destruídas, respectivamente;
20. **Fim se;**
21. **Fim.**

O método de reparação baseia-se em preencher a solução escolhendo aleatoriamente uma aula e uma posição presente nos vetores de elementos destruídos e posições destruídas. O pseudocódigo da etapa de reparação é apresentado a seguir:

1. **Repita** os passos 2 a 5 até que a solução seja completamente reparada;
2. Seleciona randomicamente uma aula (*a*) presente no vetor de elementos destruídos;
3. Seleciona randomicamente uma posição (*p*) presente no vetor de posições destruídas;
4. **Se** a aula (*a*) pode ser alocada na posição (*p*): repara a solução;
5. **Fim se;**
6. **Fim.**

RESULTADOS E DISCUSSÃO

A matriz gerada pela grade horária elaborada de forma manual e utilizada no primeiro semestre de 2019, assim como seu valor da função objetivo são apresentados na Figura 1.

```
M_manual =

     3     6     1   501     8     1   502    14   157     4    16     1     1    15     1    -1    -1    -1
     6     8     1   503    13   156   500    15   156     2    11     1     5    13     1    -1    -1    -1
   505    13   156   506     6   159     7     9   158     9     7     1    10    17   142    -1    -1    -1
   505    13   156   507    12   166    11     7     1     7     9   158    10    17   142    -1    -1    -1
    17    12   159    16    18     1    14     4     1   508     4   165    15    11     1    -1    -1    -1
    13    12     1    12    10   157    18     6   158   509     6   166    19     5   158    -1    -1    -1
    26    19   158    23     1   157    27    11     1    22     2     1    28     5   159    -1    -1    -1
    21     6   159    20    11     1    25     2     1    24     2     1    -1    -1    -1    -1    -1    -1
   100    20     1   101    21     1   102     7     1   104    11     1   105     9     1   601    14   150
   106     4     1   103    22     1   103    22     1   107    14     1   104    11     1   600     9   156
   113     4     1   114    11     1   110     2     1   109     1     1   603     4   165   112    12     1
   108    15   158   110     2     1   115     8   159   111     7     1   602     1   156   112    12     1
   121     3   157   117    15   158   120     3     1   118     5   157   119     7   157   116    10   157
   121     3   157   117    15   158   120     3     1   118     5   157   119     7   157   116    10   157

resultadoMatrizManual = 100
```

FIGURA 1. Matriz manual e seu valor da função objetivo.

Considerando a Figura 1, observa-se que o LNS deve obter um valor para a função objetivo que seja menor ou igual à 100, já que este é o valor da função objetivo referente à grade horária gerada de forma manual. No total foram realizados 15 testes, os resultados são apresentados na Tabela 1.

De acordo com os resultados apresentados na Tabela 1, nota-se que o método implementado obteve valores iguais ao da grade horária gerada de forma manual nos testes 1, 3, 7, 8 e 14. Além disso, o método foi capaz de obter soluções de melhor qualidade em três testes, sendo dois deles (testes 11 e 13) com tempos baixos (3,0 e 1,2 minutos), evidenciando uma grande vantagem da utilização do método, tendo em vista que a elaboração da grade horária de forma manual pode levar até dias para ser concluída.

TABELA 1. Testes utilizando a meta-heurística LNS.

Número do Teste	Resultado da Função Objetivo	Tempo Necessário (min)
1	100	10
2	125	45
3	100	6,5
4	125	10
5	150	43
6	175	45
7	100	12
8	100	30
9	125	60
10	125	9,5
11	75	3,0
12	150	12,6
13	75	1,2
14	100	13
15	75	56

CONCLUSÕES

Com base nos resultados apresentados, conclui-se que o algoritmo desenvolvido utilizando a meta-heurística *Large Neighborhood Search* conseguiu resolver o problema de geração e organização da grade horária do IFSP (Campus Birigui), obtendo, de forma automática e rápida, soluções de igual ou melhor qualidade em comparação à grade horária gerada de forma manual e utilizada no primeiro semestre de 2019, cumprindo assim aos objetivos estabelecidos. Além disso, o algoritmo apresenta margem de melhoria, como a implementação de outros métodos de destruição e reparação para aumentar a diversidade das soluções.

AGRADECIMENTOS

Agradecemos ao Prof. Dr. Eduardo Shigueo Hoji, ex-coordenador da área da indústria do IFSP, Campus Birigui, por disponibilizar os dados utilizados no desenvolvimento do trabalho.

REFERÊNCIAS

ADEWUMI, A.O.; SAWYERR, A.B.; ALI, M.M. A heuristic solution to the university timetabling problem. Engineering Computations: International Journal for Computer Aided Engineering and Software, v.26, n.8, p.972-984, 2009.

HAMAWAKI, C.D.L. Geração Automática de Grade Horária Usando Algoritmos Genéticos: O Caso da Faculdade de Engenharia Elétrica da UFU. Engenharia Elétrica, 2005. Disponível em:<http://repositorio.ufu.br/bitstream/123456789/29960/1/GeracaoAutomáticaGrade.pdf>. Acesso em: 24 ago 2021.

KIEFER, A.; HARTL, R.F.; SCHNELL, A. Adaptive large neighborhood Search for the curriculim-based course timetabling problem. Annals of Operations Research, v.252, n.2, p.255-282, 2017.

NETO, E.P.; VIANNA, D.S. Heurísticas eficientes para o problema de geração de grade escolar automatizada. Revista Eletrônica Produção & Engenharia, v.4, n.1, p.322-329, 2011.

PISINGER, D.; ROPKE, S. Large neighborhood search. In: Handbook of metaheuristics. Springer, 2010. p.399-419.

SHAW, P. Using constraint programming and local search methods to solve vehicle routing problems. In: International conference on principles and practice of constraint programming. Springer, 1998. p.417-431.