
**Universidade Federal de Ouro Preto
Instituto de Ciências Exatas e Aplicadas
Colegiado de Engenharia de
Computação**

**LAHC Aplicado ao Problema de
Escalonamento de Múltiplos
Projetos com Múltiplos Modos e
Restrições de Recursos**

Davi Dalfior Baltar

**TRABALHO DE
CONCLUSÃO DE CURSO**

ORIENTAÇÃO:
Janniele Aparecida Soares Araujo

**Março, 2017
João Monlevade/MG**

Davi Dalfior Baltar

**LAHC Aplicado ao Problema de Escalonamento
de Múltiplos Projetos com Múltiplos Modos e
Restrições de Recursos**

Orientador: Prof^a. MSc. Janniele Aparecida Soares Araujo

Coorientador: Prof. MSc. Samuel Souza Brito

Monografia apresentada ao curso de Engenharia de
Computação do Departamento de Computação e Sis-
temas da Universidade Federal de Ouro Preto como
requisito parcial para obtenção do grau de Bacharel
em Engenharia de Computação

Universidade Federal de Ouro Preto

João Monlevade

Março de 2017

B1971

Baltar, Davi Dalfior.

LAHC aplicado ao problema de escalonamento de múltiplos projetos com múltiplos modos e restrições de recursos [manuscrito] / Davi Dalfior Baltar. - 2017.

55f.: il.: color; grafs; tabs.

Orientador: Prof. Me. Janniele Aparecida Soares Araujo.

Monografia (Graduação). Universidade Federal de Ouro Preto. Instituto de Ciências Exatas e Aplicadas. Departamento de Computação e Sistemas de Informação.

1. Programação . 2. Algoritmos. 3. Escalonamento de projetos. I. Araujo, Janniele Aparecida Soares. II. Universidade Federal de Ouro Preto. III. Título.

CDU: 004.42

Catálogo: ficha@sisbin.ufop.br



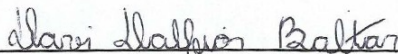
UNIVERSIDADE FEDERAL DE OURO PRETO
INSTITUTO DE CIÊNCIAS EXATAS E APLICADAS
COLEGIADO DE ENGENHARIA DE COMPUTAÇÃO

Curso Engenharia de Computação

TERMO DE RESPONSABILIDADE

Eu, Davi Dalfior Baltar, declaro que o texto do trabalho de conclusão de curso intitulado "*LAHC Aplicado ao Problema de Escalonamento de Múltiplos Projetos com Múltiplos Modos e Restrições de Recursos*" é de minha inteira responsabilidade e que não há utilização de texto, material fotográfico, código fonte de programa ou qualquer outro material pertencente a terceiros sem as devidas referências ou consentimento dos respectivos autores.

João Monlevade, 24 de março de 2017



Assinatura do aluno



ATA DE DEFESA

Aos 24 dias do mês de março de 2017, às 13 horas, na sala C204 do Instituto de Ciências Exatas e Aplicadas, foi realizada a defesa de Monografia pelo aluno **Davi Dalfior Baltar**, sendo a Comissão Examinadora constituída pelos professores: Prof^a. MSc. Janniele Aparecida Soares Araujo, Prof. MSc. Samuel Souza Brito e Prof. Dr. Euler Horta Marinho.

O candidato apresentou a monografia intitulada: "LAHC Aplicado ao Problema de Escalonamento de Múltiplos Projetos com Múltiplos Modos e Restrições de Recursos". A comissão examinadora deliberou, por unanimidade, pela aprovação do candidato, com nota 10 (dez), concedendo-lhe o prazo de 15 dias para incorporação das alterações sugeridas ao texto final.

Na forma regulamentar, foi lavrada a presente ata que é assinada pelos membros da Comissão Examinadora e pelo graduando.

João Monlevade, 24 de março de 2017.

Prof^a. MSc. Janniele Aparecida Soares Araujo
Professor Orientador/Presidente

Prof. MSc. Samuel Souza Brito
Professor Coorientador/Presidente

Prof. Dr. Euler Horta Marinho
Professor Convidado

Davi Dalfior Baltar
Graduando

RESUMO

Este trabalho aborda uma estratégia de aceitação tardia em subida da encosta para o MMRCMPSP (*Multi-Mode Resource Constrained Multi-Project Scheduling Problem*). O problema visto acima pode ser resolvido a partir de programação inteira (PI), programação por restrições, heurísticas, etc. Para este trabalho, retivemos o foco da nossa pesquisa no campo das heurísticas, porém uma parte utilizou-se da programação inteira. O estudo concentra-se na pesquisa de métodos eficientes de busca local para o problema. Neste sentido, foram avaliadas 14 vizinhanças. No trabalho proposto, as vizinhanças foram manipuladas pelo algoritmo LAHC (*Late Acceptance Hill-Climbing*) que é uma adaptação do algoritmo HC (*Hill-Climbing*), o qual possui uma aceitação tardia para a subida de encosta. Após vários testes, foram selecionados os melhores parâmetros para compor a entrada do algoritmo em questão. Para melhorar o desempenho do código, foi feita também uma paralelização com múltiplas *threads*, permitindo executar mais de uma vizinhança ao mesmo tempo, fazendo com que os resultados finais chegassem a um nível satisfatório. Com os métodos utilizados neste trabalho, foram obtidos alguns resultados melhores em relação aos encontrados na literatura.

Palavras-chaves: Late Acceptance Hill-Climbing, MMRCMPSP, escalonamento de projetos, busca local, otimização.

ABSTRACT

This work addresses late acceptance hill climbing for the MMRCMPSP (Multi-Mode Resource Constrained Multi-Project Scheduling Problem). The problem seen above can be solved using Integer Programming (IP), constraint programming, heuristics, etc. This assignment was largely focused on heuristics; however some Integer Programming was also utilized. This study was focused on researching efficient methods for problem solving. In this case, 14 neighborhoods were evaluated. For this purpose, the heuristics were manipulated by the algorithm Late Acceptance Hill-Climbing (LAHC) algorithm, which is an adaptation of the Hill-Climbing (HC) algorithm, in which it has a late acceptance hill climb. After several tests, we selected the best parameters to compose the input of the algorithm in question. To optimize improvements in code performance, a multi-threaded parallelization was also made, allowing more than one neighborhood to be executed at the same time, which brought the final results to a satisfactory level. With the methods used in this study, better results were obtained in relation to those found in other researches.

Keywords: Late Acceptance Hill-Climbing, MMRCMPSP, project scheduling, local search, optimization.

AGRADECIMENTOS

Primeiramente, agradeço a Deus por sempre me sustentar.

Gostaria de agradecer a minha família por estar comigo em todo o tempo e aos professores que me ajudaram a construir meu conhecimento.

Um agradecimento especial à professora MSc. Janniele A. S. Araujo e ao professor Dr. Haroldo G. Santos, que sem eles, o meu conhecimento jamais seria o mesmo.

“...tudo é vaidade.”
(Eclesiastes 1:2)

SUMÁRIO

1	Introdução	15
1.1	Objetivos	16
1.1.1	Objetivo Geral	16
1.1.2	Objetivos Específicos	16
2	Revisão Bibliográfica	17
2.1	Trabalhos Relacionados	17
2.2	Ambiente de Desenvolvimento	18
3	Problema	19
3.1	Recursos	21
3.1.1	Recursos Renováveis e Não Renováveis	21
3.1.2	Recursos Globais	21
3.1.3	Recursos Locais	22
3.2	Função Objetivo (FO)	23
3.3	Factibilidade e Infactibilidade	23
3.4	Instâncias	25
4	Desenvolvimento	28
4.1	Abordagem Proposta	28
4.2	Soluções Iniciais	28
4.3	Estrutura das Vizinhanças	30
4.3.1	Troca Um Modo (TUM)	30
4.3.2	Troca Dois Modos (TDM)	31
4.3.3	Troca Três Modos (TTM)	31
4.3.4	Troca Quatro Modos (TQM)	32
4.3.5	Inverte Subsequência (INS)	32
4.3.6	Desloca Atividades (DEA)	33
4.3.7	Troca Duas Atividades (TDA)	33
4.3.8	Desloca Projeto (DEP)	34
4.3.9	Compacta Projeto (COP)	34
4.3.10	Troca Dois Projetos (TDP)	35
4.3.11	Compacta Projeto Extremo (CPE)	35
4.3.12	Move Projetos (MOP)	36
4.3.13	Troca Atividades Janela (TAJ)	37
4.3.14	Inseri Atividades Janela (IAJ)	37

4.4	Implementação do LAHC	38
4.5	Repositório de Experimentos	41
4.5.1	Parâmetros e Estrutura dos Testes	41
4.5.2	Avaliação dos Experimentos	43
4.5.3	Repositório	43
5	Experimentos	45
5.1	Soluções Iniciais	46
5.2	Convergência	47
5.3	Parâmetros Utilizados	48
5.4	Resultados Finais	49
6	Considerações Finais	50
6.1	Trabalhos Futuros	50
	Referências	51
	Apêndice A – LAHC applied to The Multi-Mode Resource-Constrained Multi- Project Scheduling Problem	52

LISTA DE FIGURAS

Figura 1 – Exemplo: Recurso global	22
Figura 2 – Exemplo: Recurso local (renovável)	22
Figura 3 – Exemplo de soluções inactiváveis	24
Figura 4 – Instância genérica para o MMRCMPSP	25
Figura 5 – Instância A-1	25
Figura 6 – Projeto 1 (<i>j1011_7.mm</i>) da instância A-1	26
Figura 7 – Projeto 2 (<i>j1060_2.mm</i>) da instância A-1	26
Figura 8 – Solução factível para a instância A-1	27
Figura 9 – Exemplo da vizinhança $N_1(s)$ - Troca Um Modo (TUM)	31
Figura 10 – Exemplo da vizinhança $N_2(s)$ - Troca Dois Modos (TDM)	31
Figura 11 – Exemplo da vizinhança $N_3(s)$ - Troca Três Modos (TTM)	32
Figura 12 – Exemplo da vizinhança $N_4(s)$ - Troca Quatro Modos (TQM)	32
Figura 13 – Exemplo da vizinhança $N_5(s)$ - Inverte Subsequência (INS)	33
Figura 14 – Exemplo da vizinhança $N_6(s)$ - Desloca Atividades (DEA)	33
Figura 15 – Exemplo da vizinhança $N_7(s)$ - Troca Duas Atividades (TDA)	34
Figura 16 – Exemplo da vizinhança $N_8(s)$ - Desloca Projeto (DEP)	34
Figura 17 – Exemplo da vizinhança $N_9(s)$ - Compacta Projeto (COP)	35
Figura 18 – Exemplo da vizinhança $N_{10}(s)$ - Troca Dois Projetos (TDP)	35
Figura 19 – Exemplo da vizinhança $N_{11}(s)$ - Compacta Projeto Extremo (CPE)	36
Figura 20 – Exemplo da vizinhança $N_{12}(s)$ - Move Projetos (MOP)	37
Figura 21 – Exemplo da vizinhança $N_{13}(s)$ - Troca Atividades Janela (TAJ)	37
Figura 22 – Exemplo da vizinhança $N_{14}(s)$ - Inserir Atividades Janela (IAJ)	38
Figura 23 – Exemplos: LAHC e as variações no tamanho da lista L	39
Figura 24 – Página inicial do repositório	44
Figura 25 – Sumário dos testes para o LAHC	44
Figura 26 – Instância B-10 usando diferentes tamanhos de lista L	47
Figura 27 – Instância B-8 usando diferentes tamanhos de lista L	47

LISTA DE TABELAS

Tabela 1 – Intensidade: parâmetros utilizados nos testes	42
Tabela 2 – Lista L : parâmetros utilizados nos testes	42
Tabela 3 – Valores mínimos e máximos para k	42
Tabela 4 – Características das instâncias: A, B e X	45
Tabela 5 – Comparação: Soluções iniciais com as melhores da literatura	46
Tabela 6 – Intensidade: parâmetros escolhidos	48
Tabela 7 – Valores mínimos e máximos escolhidos para k	48
Tabela 8 – Melhores resultados para o LAHC, MISTA 2013 e um relatório técnico	49

LISTA DE ABREVIATURAS E SIGLAS

CP - *Constraint Programming*

CP-based HPSO - *Combinatorial-Priority-Based Hybrid PSO*

CPD - *Critical Path Duration*

FBI - *Forward-Backward Improvement*

GRC - *Greedy Randomized Constructive*

FO - *Função Objetivo*

HC - *Hill-Climbing*

LAHC - *Late Acceptance Hill-Climbing*

MCTS - *Monte-Carlo Tree Search*

MIP - *Mixed Integer Program*

MISTA - *Multidisciplinary International Scheduling Conference: Theory and Applications*

MMRCMPSP - *Multi-Mode Resource Constrained Multi-Project Scheduling Problem*

MRCPSP - *Multi-Mode Resource-Constrained Project Scheduling Problem*

PD - *Project Delay*

PI - *Programação Inteira*

PSO - *Particle Swarm Optimization*

PSP - *Project Scheduling Problem*

PSPLIB - *Project Scheduling Problem Library*

RCPSP - *Resource-Constrained Project Scheduling Problem*

TMS - *Total Makespan*

TPD - *Total Project Delay*

VNS - *Variable Neighborhood Search*

1 INTRODUÇÃO

Desde os primórdios da sociedade, o ser humano tenta encontrar formas mais organizadas de realizar suas tarefas com mais rapidez. Esse tipo de problema pode ser identificado desde a construção das pirâmides do Egito, em que as atividades e formas de trabalho eram rigorosamente pensadas e estudadas, bem como na construção de um túnel de metrô nos dias de hoje. Para estes e outros casos, a organização das atividades realizadas nesses projetos, são cruciais na influência sobre o tempo em que elas acabarão. Pensando nisso, surge então o Problema de Escalonamento de Projeto, ou PSP (*Project Scheduling Problem*), que consiste em escalonar atividades ao longo do tempo, de tal forma que as relações de precedência entre as atividades são satisfeitas e os limites de consumo de recursos são respeitados. Uma visão abrangente sobre o PSP pode ser visto na obra de [Demeulemeester e Herroelen \(2002\)](#). Neste trabalho, foi tratada uma variação do PSP, denominada: Problema de Escalonamento de Múltiplos Projetos com Múltiplos Modos e Restrições de Recursos, ou MMRCMPSP (*Multi-Mode Resource Constrained Multi-Project Scheduling Problem*) ([Wauters et al \(2013\)](#)). Para este problema, as atividades podem ser processadas em diferentes modos, com variação no tempo de execução e no consumo de diferentes quantidades de recursos. Estes modos podem ser vistos como formas diferentes de se executar uma mesma atividade.

O PSP pertence à classe dos problemas NP-Difícil ([Demeulemeester e Herroelen \(2002\)](#)). Isso faz com que as derivações do problema, como o MMRCMPSP, também sejam. Até a geração de uma solução inicial factível para o MMRCMPSP também é NP-Difícil, pois, uma seleção factível da maioria dos modos selecionados é correspondente a resolver o problema da mochila de forma multidimensional ([Wauters et al \(2013\)](#)). O MMRCMPSP pode ser usado para modelar muitos problemas em várias áreas, como por exemplo: gestão de projetos em companhias de tecnologia da informação, escalonamento de instruções para a arquitetura de processadores, construções na área da engenharia civil ([Xu e Feng \(2014\)](#)), entre outras.

Neste trabalho é proposto e avaliado computacionalmente um algoritmo baseado na meta-heurística LAHC (*Late Acceptance Hill-Climbing*) ([Burke e Bykov \(2008\)](#)) que é uma melhoria do algoritmo HC (*Hill-Climbing*). Este algoritmo baseado no LAHC usa uma memória de longo prazo para diversificar e operar com buscas locais estocásticas em uma série de vizinhanças.

A técnica implementada considera como entrada um conjunto de instâncias de múltiplos projetos, que estão disponíveis no *site* da MISTA 2013 (*Multidisciplinary Inter-*

national Scheduling Conference: Theory and Applications, 2013)¹, sendo estas instâncias compostas de alguns projetos específicos encontrados na PSPLIB² (*Project Scheduling Problem Library*). Isto permite que qualquer instância especificada neste formato possa ser trabalhada com as técnicas propostas neste trabalho.

1.1 OBJETIVOS

As subseções seguintes mostrarão o objetivo geral e os objetivos específicos do trabalho em questão.

1.1.1 OBJETIVO GERAL

Este trabalho tem como objetivo geral a implementação do algoritmo LAHC com novas técnicas, a fim de melhorar as soluções conhecidas na literatura para um dado conjunto de instâncias do problema em questão. Essas instâncias e soluções estão disponíveis no *site* oficial da MISTA 2013.

1.1.2 OBJETIVOS ESPECÍFICOS

- Estudar o funcionamento do algoritmo LAHC.
- Analisar as instâncias.
- Identificar técnicas para aprimorar o código.
- Realizar experimentos.
- Enviar os resultados para a MISTA.
- Implementar *scripts* de forma a facilitar a automatização dos testes, fazendo com que estes sejam executados na forma de bateria de testes.
- Criar uma página *web* a fim de sumarizar os testes realizados com o algoritmo LAHC, bem como todos os parâmetros utilizados em cada teste e armazenar na página as melhores soluções encontradas em relação aos testes anteriores e as melhores conhecidas pela literatura.
- Implementar um *software* de análise dos resultados obtidos pela bateria de testes de forma a gerar as páginas *web* automaticamente.
- Publicar um artigo com os resultados obtidos.

¹ <https://gent.cs.kuleuven.be/mista2013challenge/>

² <http://www.om-db.wi.tum.de/psplib/>

2 REVISÃO BIBLIOGRÁFICA

Neste capítulo serão apresentados alguns trabalhos que abordam o problema de escalonamento de projetos com múltiplos modos e restrições de recursos, bem como o ambiente de desenvolvimento utilizado.

2.1 TRABALHOS RELACIONADOS

O artigo de [Xu e Feng \(2014\)](#) trabalha com o MMRCMPSP no projeto da construção de uma grande hidrelétrica na região sudeste da China. A finalidade é executar todos os projetos paralelos da hidrelétrica no menor tempo possível. Os autores levam em consideração todos os parâmetros existentes, como: recursos, modos, atividades e projetos paralelos encontrados na elaboração do projeto geral. Os autores utilizaram um método baseado na otimização de enxame de partículas, ou PSO (*Particle Swarm Optimization*) com o uso de *Fuzzy* em um ambiente aleatório, no qual foi desenvolvido um algoritmo baseado na prioridade combinacional híbrida, ou CP-based HPSO (*Combinatorial-Priority-Based Hybrid PSO*) para resolver o problema e utilizando-o para atribuir os modos e escalonar as atividades. O algoritmo foi executado no *software* MATLAB 7.0 e com os métodos utilizados, os autores conseguiram, para um projeto X, diminuir o tempo das obras de 200.64×10^2 horas para 170.8878×10^2 horas, ou seja, aproximadamente 15% no tempo total inicialmente estipulado.

[Asta et al \(2013a\)](#) propuseram uma heurística para o MMRCMPSP utilizando o método MCTS (*Monte-Carlo Tree Search*) e uma heurística construtiva para gerar soluções iniciais para o problema. Os autores implementaram treze vizinhanças, cada uma com um propósito diferente, como por exemplo: a troca de dois projetos, troca de duas atividades, troca aleatória do modo de uma das atividades, entre outras. Para gerenciar a escolha das vizinhanças, foi utilizada uma combinação de meta-heurísticas e uma hiper-heurística, tudo no contexto de uma abordagem baseada na população de vários segmentos que usa ideias de algoritmos meméticos. Com os métodos utilizados, os autores conseguiram igualar ou superar todas as soluções propostas na primeira fase da competição MISTA 2013 e alcançaram o primeiro lugar.

[Geiger \(2013\)](#) abordou em seu trabalho o problema MMRCMPSP propondo uma heurística com quatro vizinhanças, que tem como base o VNS (*Variable Neighborhood Search*) e uma busca local iterativa. As vizinhanças são responsáveis por realizar atividades distintas no problema, como por exemplo: troca de duas atividades, troca de dois modos de cada atividade, entre outras. O autor também utiliza o conceito de movimento de

perturbação nos modos de uma atividade com o intuito de encontrar alternativas para as soluções. Com as ideias propostas pelo autor, o mesmo conseguiu igualar ou superar as soluções propostas na primeira fase da competição MISTA 2013 e alcançou o segundo lugar.

No trabalho proposto por Toffolo et al (2013) o MMRCMPSP foi abordado a partir do uso de programação inteira e uma heurística híbrida. Uma heurística construtiva foi usada na geração de soluções iniciais. Os autores fizeram o uso de uma busca local inteira que trabalha com janela de tempo combinada com o método heurístico FBI (*Forward-Backward Improvement*) em que, a partir de modelos implementados no solver Gurobi 5.5, foram realizadas as trocas de modos, atividades e, conseqüentemente, diminuiu-se o tempo do projeto. Com os resultados, os autores conseguiram a terceira colocação na competição MISTA 2013.

O artigo de Artigues e Hebrard (2013) propôs uma solução para o problema através de uma relaxação do MMRCMPSP com o uso de MIP (*Mixed Integer Program*) e fizeram o uso de PC (*Constraint Programming*) através do solver IBM CPOptimizer. Com essas técnicas, foi possível encontrar uma boa alocação inicial para os modos. A partir disso, uma larga busca com vizinhanças foi utilizada para otimizar o escalonamento. Com os resultados, os autores conseguiram a quarta colocação na competição MISTA 2013.

Borba et al (2013) propuseram em seu artigo um simples método de busca local estocástico para o MMRCMPSP. Eles mantiveram uma permutação viável na seleção dos modos da atividade, melhorando com isso a troca da ordem de duas atividades, ou mudando o modo de uma atividade. Foi proposta uma técnica para reduzir o tamanho das vizinhanças analisadas. Com os métodos, eles foram qualificados em quinto lugar na MISTA 2013.

2.2 AMBIENTE DE DESENVOLVIMENTO

O ambiente de desenvolvimento deste trabalho contou com o auxílio de vários *softwares*. Todo o desenvolvimento foi feito em ambiente Linux openSUSE 13.2¹. Foi utilizada a linguagem C com o compilador GCC 4.7.1² para construção do código; o solver IBM ILOG CPLEX Optimizer 12.6³ na execução de um modelo para ajudar na criação das soluções iniciais para o problema; Python 2.7.12⁴ para a execução de *scripts* a fim de realizar uma bateria de testes e o conjunto: *javascript*, CSS e HTML na elaboração de uma página web, na qual os resultados serão sumarizados.

¹ <https://pt.opensuse.org/>

² <https://gcc.gnu.org/>

³ <https://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/>

⁴ <https://www.python.org/>

3 PROBLEMA

O problema estudado neste trabalho é derivado de um problema já conhecido na computação, que é o Problema de Escalonamento de Projeto, ou PSP (*Project Scheduling Problem*). Kolisch e Sprecher (1996) abordaram e propuseram um conjunto de instâncias para o PSP com diferentes características e níveis de dificuldade. O campo de estudo sobre escalonamento de projeto ganhou contribuições significativas de Demeulemeester et al (1998) introduzidas na publicação do *Handbook of Recent Advances in Project Scheduling* e posteriormente em um estudo detalhado no livro *Project Scheduling: A Research Handbook*, onde os autores Demeulemeester e Herroelen (2002) aprofundam o assunto demonstrando suas características, detalhes, complexidade, etc. Os problemas do tipo PSP aparecem quando se tem várias atividades em um projeto, sendo que cada uma delas possui um tempo de início e fim, em que devem ser realizadas as alocações com o intuito de minimizar o tempo gasto para realizar todas as atividades, respeitando a ordem de precedência entre elas. Entretanto, saber qual o momento ideal para que as atividades de um projeto se iniciem em um determinado tempo, não é uma tarefa fácil, uma vez que a complexidade para a resolução desse tipo de problema pertence à classe dos problemas NP-Difícil, como foi mostrado na obra de Demeulemeester e Herroelen (2002).

Uma derivação do PSP incluiu uma restrição de recursos às atividades de um projeto. Essa derivação recebe o nome de RCPSP (*Resource-Constrained Project Scheduling Problem*), que também pertence à classe NP-Difícil (Blazewicz et al (1983)), onde cada atividade de um projeto consome uma determinada quantidade de recursos e o objetivo desse problema é alocar as atividades de forma a não extrapolar a quantidade de recursos estipuladas. Além dos recursos, o RCPSP herda igualmente as relações de precedência e tempo de início e fim das atividades do PSP. Uma extensão do RCPSP, também conhecida como MRCPSP (*Multi-Mode Resource-Constrained Project Scheduling Problem*), propõe que cada atividade possua dois ou mais modos de execução em que, cada modo possua um determinado tempo de duração, bem como a quantidade de recursos gastos por este. Problemas da classe MRCPSP são denominados como *single-project* e *multi-mode*, ou seja, possuem um único projeto com mais de um modo de execução para cada atividade. Um levantamento abrangente sobre o RCPSP e o MRCPSP pode ser visto no trabalho de Demeulemeester e Herroelen (2002)

A partir desses conceitos apresentados, surge então um novo tipo de problema, que pode ser considerado como uma extensão do MRCPSP, que é o Problema de Escalonamento de Múltiplos Projetos com Múltiplos Modos e Restrições de Recursos, ou MMRCMPSP (*Multi-Mode Resource Constrained Multi-Project Scheduling Problem*), proposto pela MISTA 2013. Por ser derivado do PSP, a análise de sua complexidade o faz pertencente

também à classe dos problemas NP-Difícil (Demeulemeester e Herroelen (2002)). Para este tipo de problema, além das características já herdadas do PSP, RCPS e MRCPSP, o MMRCMPSP possui múltiplos projetos do tipo MRCPSP, bem como um novo tipo de recurso, conhecido como recurso global. O recurso global é um tipo de recurso compartilhado entre todos os projetos. A subseção 3.1.2 discute o assunto sobre recursos globais com mais detalhes. O modelo do MMRCMPSP (Fonte: adaptado de Toffolo et al (2015)) considera os seguintes dados de entrada:

P conjunto de projetos;

J conjunto de atividades;

J_p conjunto de atividades de um projeto p , tal que $J_p \subseteq J \forall p \in P$;

M_j conjunto de modos para a atividade j ;

K conjunto de recursos não renováveis;

R conjunto de recursos renováveis;

$Pred$ conjunto de predecessores entre duas atividades;

$Pred_j$ conjunto de predecessores imediatos da atividade j ;

d_{jm} duração da atividade j no modo m ;

u_{kjm} unidades necessárias de recurso não renovável k para a atividade j ser processada no modo m ;

v_{rjm} unidades necessárias de recurso renovável r para a atividade j ser processada no modo m ;

o_k unidades disponíveis do recurso não renovável k ;

q_r unidades disponíveis do recurso renovável r ;

O MMRCMPSP compreende um conjunto de restrições relacionadas com a precedência entre as atividades J e o consumo de recursos. Cada atividade $j \in J$ pode ser executada em um determinado modo $m \in M$, ele determina o tempo necessário para a execução d_{jm} e o consumo de recursos renováveis R e não renováveis K para as quantidades necessárias u_{kjm} e v_{rjm} , respectivamente. Este consumo não pode exceder o montante disponível de recursos renováveis e não renováveis. Todas as relações de precedência $Pred$ e $Pred_j$ devem ser garantidas. Uma abordagem mais detalhada sobre os recursos é apresentada na seção 3.1.

As soluções geradas para MMRCMPSP podem ser tanto factíveis como infactíveis. As soluções factíveis são aquelas que se restringem aos limites de recursos estipulados pela instância e que a alocação das atividades respeite as relações de precedências. As

soluções infactíveis para este problema são aquelas que tiveram alguma extrapolação do limite de recursos estipulados pela instância e/ou uma quebra na relação de precedência da alocação de uma ou várias atividades, porém neste trabalho não são consideradas as soluções infactíveis. A seção 3.3 trata com mais detalhe o assunto sobre factibilidade e infactibilidade. As seções seguintes mostrarão com mais detalhes as propriedades do MMRCMPSP definidas pela MISTA 2013.

3.1 RECURSOS

Os recursos de um problema são os responsáveis por dar as diretrizes e limitar a quantidade de atividades alocadas em um mesmo instante de tempo, fazendo com que a dificuldade na resolução do problema aumente ou diminua, dependendo da quantidade de recurso disponível. Para o MMRCMPSP, os recursos podem ser renováveis R , local ou global (compartilhados com outros projetos) e não renováveis K , somente local. Nesta seção, serão apresentados os tipos de recursos que compõem o MMRCMPSP.

3.1.1 RECURSOS RENOVÁVEIS E NÃO RENOVÁVEIS

Os recursos renováveis para o problema proposto são aqueles cuja quantidade é renovada em cada unidade de tempo. Exemplos práticos de recursos renováveis podem ser vistos como: dias, meses, mão de obra, etc.

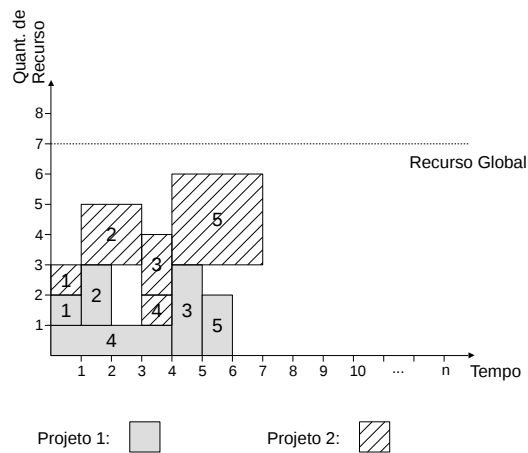
Os recursos não renováveis são aqueles cuja quantidade se esgota ao longo do tempo. Exemplos práticos de recursos não renováveis podem ser vistos como: dinheiro, matéria-prima, etc.

3.1.2 RECURSOS GLOBAIS

Os recursos globais fazem parte dos recursos renováveis e são compartilhados entre todos os projetos. De acordo com as definições do problema, a soma dos recursos consumidos em cada unidade de tempo de cada projeto, juntos, não podem ultrapassar o limite máximo do recurso global estipulado pela instância. A Figura 1 mostra um exemplo de alocação de dois projetos, cada um contendo cinco atividades e um limite máximo de recurso global igual a sete.

Para o exemplo da Figura 1, as únicas regras de precedência, são: para o projeto 1, a atividade 1 deve começar antes da atividade 2 e a atividade 4 deve começar antes da atividade 5; para o projeto 2, a atividade 1 e 2 deverão começar antes da atividade 3 e a atividade 3 antes da atividade 5. A partir do exemplo, é fácil perceber que existem infinitas formas de se alocar as atividades dos projetos 1 e 2 quando $n \rightarrow \infty$. Os limites de recursos globais podem variar dependendo da instância.

Figura 1: Exemplo: Recurso global

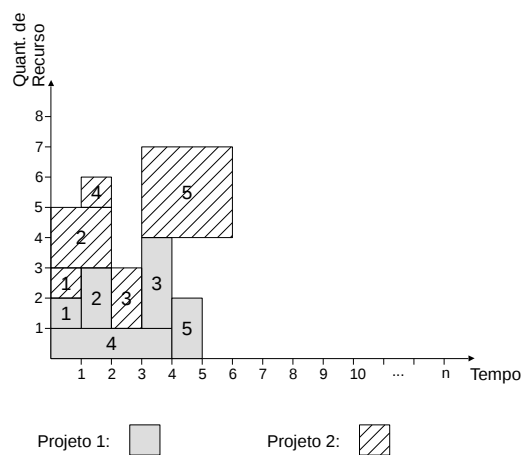


Fonte: adaptado de Demeulemeester e Herroelen (2002)

3.1.3 RECURSOS LOCAIS

Os recursos locais podem ser renováveis ou não renováveis e são compartilhados entre um único projeto. A utilização destes recursos, quando renováveis, não pode ultrapassar o limite estipulado pela instância para cada unidade de tempo. Os recursos não renováveis se esgotam ao longo do tempo. Para o exemplo mostrado na Figura 2, o limite máximo de recursos locais renováveis para o projeto 1 é de 4 unidades e para o projeto 2 é de 3 unidades. Na Figura 2 é possível ver que na primeira unidade de tempo, o projeto 1 alocou duas unidades de seu recurso local, enquanto o projeto 2 alocou três unidades de seu recurso local, respeitando assim o limite máximo de unidades de recurso por unidade de tempo. Os limites de recursos locais variam dependendo de cada projeto.

Figura 2: Exemplo: Recurso local (renovável)



Fonte: adaptado de Demeulemeester e Herroelen (2002)

3.2 FUNÇÃO OBJETIVO (FO)

O objetivo do problema proposto por [Wauters et al \(2013\)](#) é encontrar soluções factíveis que minimizem o TPD (*Total Project Delay*) e o TMS (*Total Makespan*), sendo o TPD o objetivo principal.

O TPD é o atraso total do projeto, definido como a diferença entre a duração do caminho crítico, ou CPD (*Critical Path Duration*) que é a menor duração que um projeto pode ter considerando somente a ordem de precedência das atividades. Formalmente, dado um projeto P , o atraso do projeto, ou PD (*Project Delay*), PD_i para um projeto $i \in P$, é definido como:

$$PD_i = MS_i - CPD_i \quad (3.1)$$

onde MS_i é o *makespan* (duração) de um projeto i e o CPD_i é a duração do caminho crítico de um projeto i . Com isso, o TPD pode ser definido como:

$$TPD = \sum_{i=1}^n PD_i \quad (3.2)$$

onde n é a quantidade de projetos.

O TMS é a duração de todo o escalonamento dos múltiplos projetos, o qual pode ser visto também como o tempo de término do último projeto alocado na linha do tempo.

[Wauters et al \(2013\)](#) adotou um padrão para facilitar os cálculos envolvendo a FO formada pelo TPD e o TMS. Esse padrão implica em concatenar o TMS logo após o TPD, de forma que o TMS mantenha sempre cinco números inteiros. Caso o TMS tenha uma quantidade de casas menor que cinco, os zeros completam a parte não preenchida à esquerda, caso contrário, o TPD e TMS são concatenados diretamente, por exemplo: Dado um $TPD = 1$ e um $TMS = 23$, a FO neste padrão seria 100023.

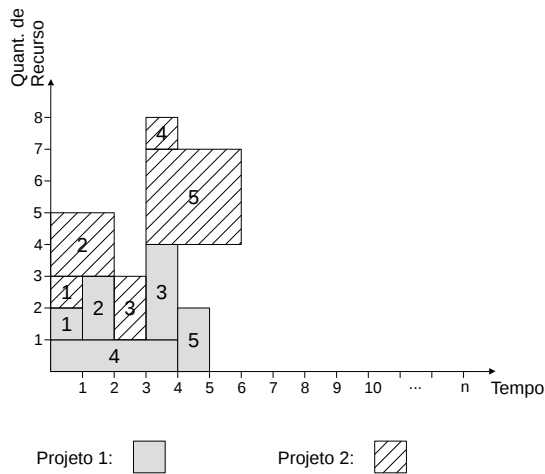
3.3 FACTIBILIDADE E INFACIBILIDADE

Uma solução é considerada factível quando todas as restrições da instância são satisfeitas. Para este problema a factibilidade ocorre se, e somente se, a distribuição dos recursos locais, dos recursos globais e o movimento das atividades satisfazem todas as regras estipuladas pela instância. Alguns exemplos de soluções factíveis podem ser vistas na Figura 8 da seção 3.4 e na Figura 2 da subseção 3.1.3.

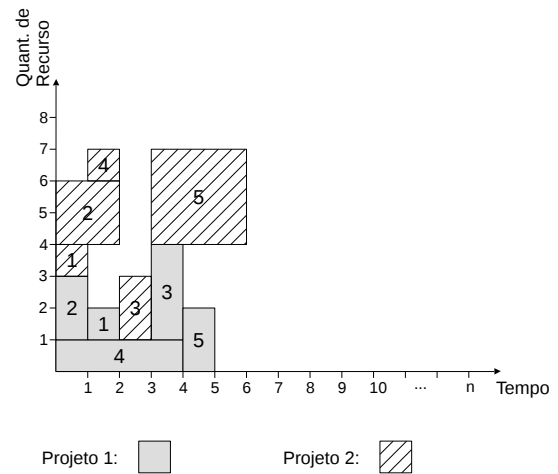
Uma solução é infactível quando uma ou mais restrições da instância são infringidas, ou seja, quando a distribuição dos recursos locais, ou dos recursos globais ou o movimento de alguma das atividades, não satisfazem uma ou mais regras estipuladas pela instância.

Com base no exemplo da Figura 2 (subseção 3.1.3), uma solução infactível que extrapola o recurso local é vista na Figura 3(a) em que, quando a atividade 4 do projeto 2 foi alocada no tempo 4, o projeto 2 ultrapassou o limite máximo do recurso local que era de no máximo 3 unidades para o projeto 2. Outro exemplo de solução infactível, porém com base na ordem de precedência das atividades é mostrado na Figura 3(b). Como pode ser visto, a solução rompeu uma das regras de precedência entre as atividades do projeto 1, onde a atividade 1 começou depois de sua sucessora, a atividade 2. Outra forma infactível para uma solução é quando as regras referentes aos recursos e às ordens de precedência são quebradas simultaneamente, como mostra a Figura 3(c), a qual concentra a quebra das regras citadas nos exemplos das Figuras 3(a) e 3(b). Por fim, a Figura 3(d) mostra o caso em que ocorre a infactibilidade por causa da extrapolação do recurso global.

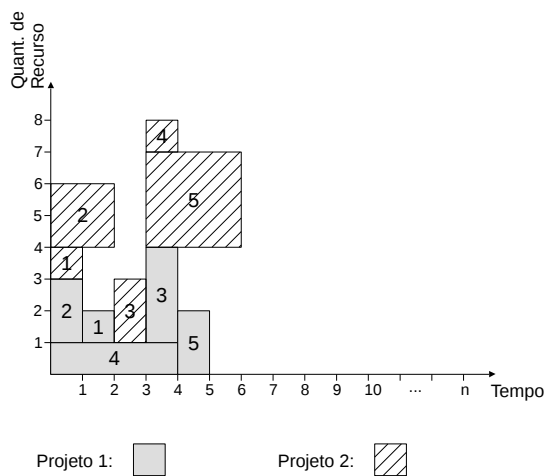
Figura 3: Exemplo de soluções infactíveis



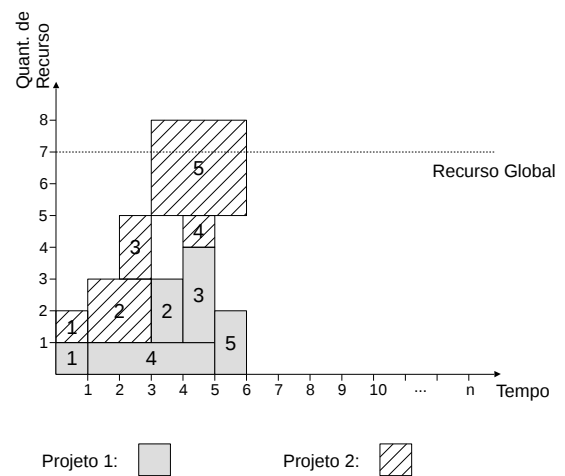
(a) Extrapolação do recurso local renovável



(b) Movimento inválido da atividade



(c) Extrapolação do recurso local renovável e movimento inválido da atividade



(d) Extrapolação do recurso global

3.4 INSTÂNCIAS

De forma genérica, as instâncias do MMRCMPSP possuem $n + 1$ projetos, ou seja, incorporam duas ou mais instâncias do MRCPSP. Isso se deve ao fato desse problema envolver múltiplos projetos. Na MISTA 2013 foram propostas trinta instâncias, em que cada uma contém uma quantidade que varia entre dez e vinte projetos, cada projeto contendo entre dez e trinta atividades cada, e todas as atividades possuindo três modos de execução. A estrutura genérica para uma instância com múltiplos projetos para o tipo MMRCMPSP pode ser vista na Figura 4. A Figura 5 mostra o exemplo de uma das instâncias utilizadas no trabalho (neste caso, a instância A-1).

Figura 4: Instância genérica para o MMRCMPSP

linha	
1	Número de projetos
2	Instante de início do projeto 0
3	Duração do caminho crítico do projeto 0
4	Caminho para o arquivo de projeto do projeto 0
5	Instante de início do projeto 1
6	Duração do caminho crítico do projeto 1
7	Caminho para o arquivo de projeto do projeto 1
8	...
9	Instante de início do projeto n-1
10	Duração do caminho crítico do projeto n-1
11	Caminho para o arquivo de projeto do projeto n-1
12	Quantidade de tipos de recursos
13	Capacidades de recursos globais (-1 é não global)

Fonte: adaptado de [Wauters et al \(2013\)](#)

Figura 5: Instância A-1

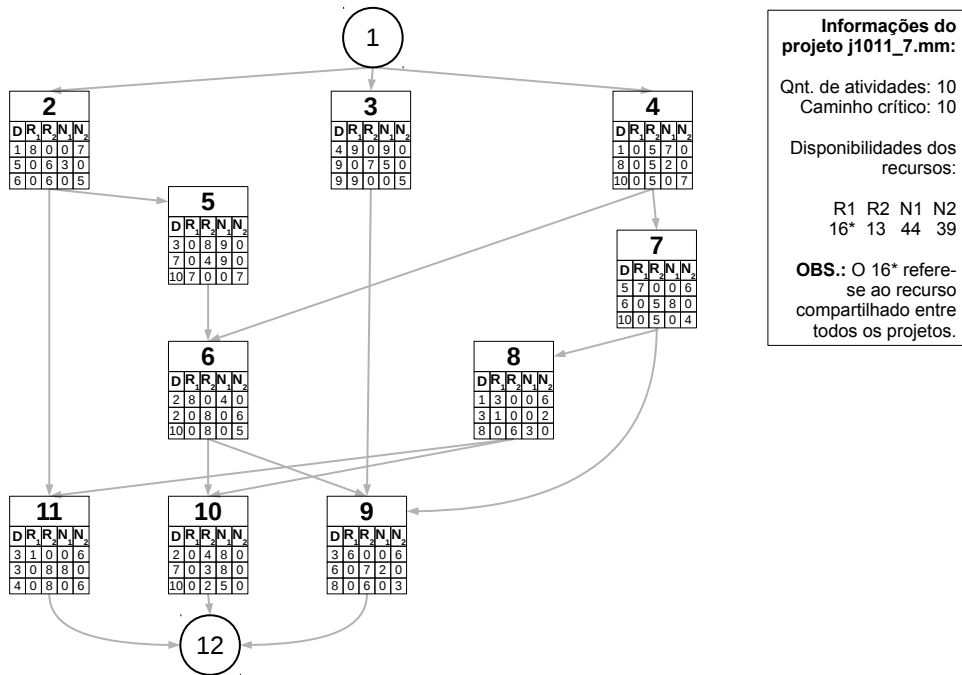
linha	
1	2
2	0
3	10
4	j10.mm/j1011_7.mm
5	4
6	19
7	j10.mm/j1060_2.mm
8	4
9	16 -1 -1 -1

Fonte: elaborado pelo autor

A instância A-1 possui dois projetos, o projeto 1 (*j1011_7.mm*) e o projeto 2 (*j1060_2.mm*), em que, cada um dos projetos possui dez atividades com quatro recursos cada, sendo dois recursos do tipo renovável e dois recursos do tipo não renovável e cada atividade contendo três modos possíveis de execução, como pode ser visto nas Figuras 6 e 7. A instância A-1 possui um recurso global igual a 16 para o recurso 1 (como pode ser visto na Figura 5). Os demais recursos da instância são utilizados de forma local de acordo com cada projeto. Um diagrama foi construído para cada um dos projetos: projeto 1 (*j1011_7.mm*) e projeto 2 (*j1060_2.mm*), com o intuito de melhorar a visualização das ordens de precedência, da duração de cada modo e da quantidade de recursos consumidos por cada um dos projetos. Os diagramas podem ser vistos nas Figuras 6 e 7. Nos diagramas,

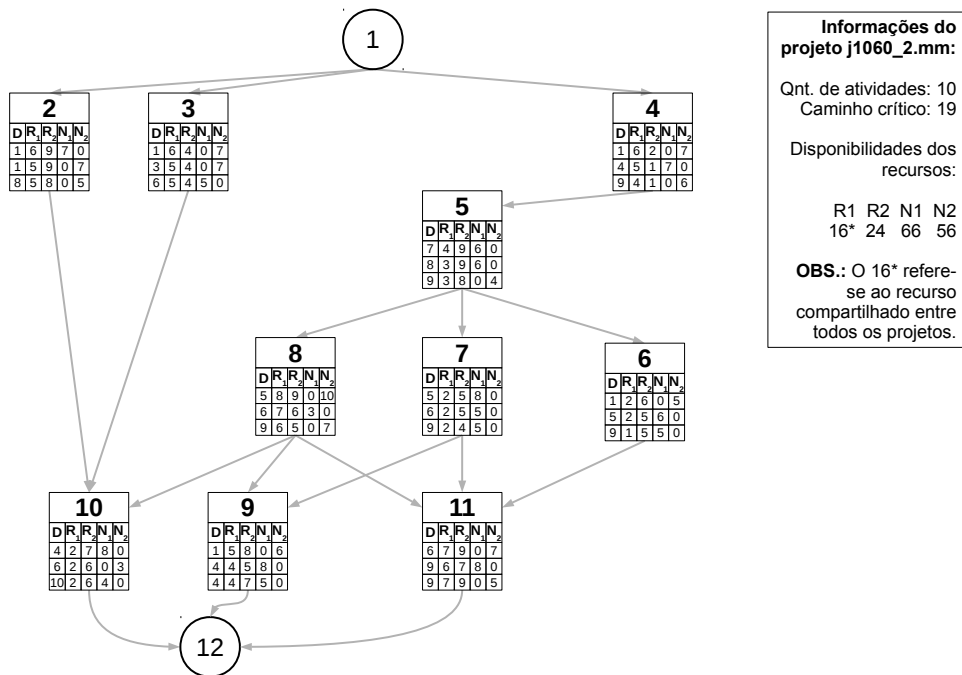
cada nó representa uma atividade do projeto. Dentro dos nós, o D representa a duração, o R_1 o primeiro recurso renovável, o R_2 o segundo recurso renovável, o N_1 o primeiro recurso não renovável e o N_2 o segundo recurso não renovável. Os nós de número 1 e 12 são nós fictícios, pois possuem duração e quantidade de recurso igual a zero.

Figura 6: Projeto 1 (*j1011_7.mm*) da instância A-1



Fonte: elaborado pelo autor

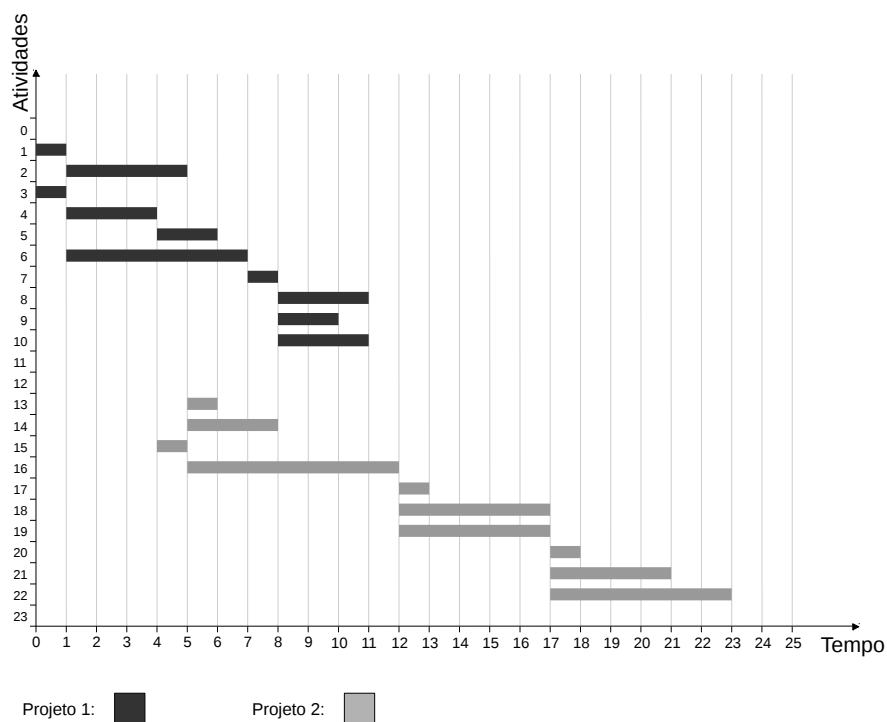
Figura 7: Projeto 2 (*j1060_2.mm*) da instância A-1



Fonte: elaborado pelo autor

Uma possível solução válida para a instância A-1 pode ser vista a partir do gráfico de *Gantt* na Figura 8. Esta solução tem como FO um TPD igual a 1 e um TMS igual a 23.

Figura 8: Solução factível para a instância A-1



Fonte: adaptado de Soares (2013)

4 DESENVOLVIMENTO

Neste capítulo será esboçado todo o desenvolvimento do trabalho, mostrando as técnicas e os recursos utilizados, bem como a estrutura implementada para o armazenamento e sumarização dos experimentos.

4.1 ABORDAGEM PROPOSTA

Como todo trabalho na área de otimização, o objetivo principal é sempre bem claro: maximizar ou minimizar a Função Objetivo (FO) das soluções. Neste trabalho, o objetivo principal é minimizar FO das soluções já conhecidas na literatura. Encontrar ou melhorar soluções pode não ser uma tarefa fácil. Com isso, diferentes técnicas são propostas para tentar resolver o problema em questão, dentre elas estão a: Programação Inteira (PI), programação por restrições, heurísticas, etc. Não existe uma técnica melhor que outra, e sim, uma que se adapta melhor que outras dependendo do problema a ser resolvido. Por se tratar de um amplo campo de busca provido pelas instâncias, baseado em [Toffolo et al \(2013\)](#), este trabalho se utilizou de duas técnicas: programação inteira e heurísticas. Em um primeiro momento, foi utilizada a programação inteira como parte da técnica para gerar uma solução inicial para o problema. Em um segundo momento foram utilizadas as vizinhanças e o LAHC para fazer o refinamento (melhoria) na solução inicial. Neste trabalho, as soluções têm como base uma sequência (vetor) contendo as atividades de todos os projetos, em que esta sequência é organizada com base na ordenação topológica das atividades. Outra base para a solução é a sequência contendo os modos de execução de cada uma das atividades. As modificações nas sequências de atividades e modos que vão determinar os atrasos no tempo de término de cada projeto. Ao final, uma sumarização dos resultados é feita, onde as soluções são comparadas com as melhores soluções encontradas na literatura. As seções seguintes abordarão mais detalhadamente cada uma das ferramentas, adaptações e sumarizações.

4.2 SOLUÇÕES INICIAIS

Para melhorar a resolução do problema em questão, dividiu-se o trabalho em duas etapas, sendo a primeira etapa referente à geração de uma solução inicial e a segunda etapa como sendo a etapa de refinamento da solução inicial gerada. Para construir parte da solução inicial, foi utilizado o solver IBM ILOG CPLEX Optimizer (ou simplesmente, CPLEX), o qual utiliza Programação Inteira. O CPLEX foi utilizado para construir um conjunto de diferentes conjuntos de modos (*mode sets*) que satisfazem o consumo de

recursos não renováveis. Uma vez que a escolha dos modos também determinará a duração das atividades, uma estratégia gulosa para priorizar a escolha dos modos mais rápidos foi implementada. Deve-se observar, no entanto, que isso não garante um TPD menor, porque as restrições dos recursos renováveis podem atrasar as atividades.

Baseado no modelo proposto por Santos et al (2014), o programa implementado para criar estes conjuntos iniciais de modos considera: J atividades com seus respectivos tempos de processamento p_{jm} e os K recursos não renováveis. Cada atividade tem um conjunto M_j de possíveis modos e o consumo do recurso não renovável k da atividade j no modo m , que é denotada como r_{jmk} . Assim, o programa é capaz de escolher para qual modo m a atividade j será alocada, considerando suas respectivas variáveis de decisão x_{jm} (onde 0 e 1 correspondem a falso e verdadeiro, respectivamente) e a disponibilidade de recursos q_k para cada recurso não renovável.

min. :

$$\sum_{j \in J} p_{jm} \cdot x_{jm} \quad (4.1)$$

s.t. :

$$\sum_{j \in J} \sum_{m \in M_j} r_{jmn} x_{jm} \leq q_k \quad \forall k \in K \quad (4.2)$$

$$x_{jm} \in \{0, 1\} \quad \forall j \in J, m \in M_j \quad (4.3)$$

Para produzir iterativamente diferentes conjuntos de modos válidos, o processo realizou-se da seguinte forma: A resolução do programa é repetida, impedindo que uma pequena porção de modos de processamento já selecionados nas soluções anteriores apareça na nova solução (isto é, fixando as respectivas variáveis de decisão como zero). Cada conjunto de modo válido é usado para construir uma solução completa utilizando um algoritmo construtivo aleatório guloso, ou GRC (*Greedy Randomized Constructive*) para decidir a ordem de alocação das atividades. O GRC iterativamente seleciona a próxima atividade a ser alocada por uma seleção aleatória dentre todas as atividades disponíveis. A probabilidade de selecionar atividades com tempos de início bem pequenos é exponencialmente maior, como mostrado por Bresina (1996). Uma vez que um conjunto de soluções é criado, a busca local do LAHC pode ser iniciada em múltiplas *threads*.

Em posse das soluções iniciais, a tarefa a ser realizada na segunda etapa é a de refinar a solução, ou seja, melhorar a solução inicial encontrada. Para cumprir esta

tarefa, foi implementada uma modificação da meta-heurística LAHC e um conjunto de 14 vizinhança, em que cada vizinhança possui uma função específica na busca local. Ambos serão discutidos com mais detalhes nas seções seguintes.

4.3 ESTRUTURA DAS VIZINHANÇAS

A estrutura $N(S)$ é composta por 14 tipos de movimentos (vizinhanças), onde 6 delas são baseadas no trabalho de Geiger (2013) e 4 no trabalho de Asta et al (2013b). O movimento a ser executado em $N(S)$ é selecionado aleatoriamente com base na intensidade de cada vizinhança. A intensidade é probabilidade de uma vizinhança ser sorteada em uma roleta viciada como vista no trabalho de LINDEN (2006). A escolha do conjunto de intensidades será discutida nas seções posteriores. Essas vizinhanças também foram inspiradas nos trabalhos de Santos et al (2014) onde, a principal diferença é que neste trabalho os movimentos são feitos de forma estocástica e não determinista como proposto originalmente.

Todas as vizinhanças propostas neste trabalho realizam mudanças na sequência de atividades e na sequência de modos. É importante enfatizar que, para otimizar a reconstrução da solução a partir da sequência de atividades, foi feita uma ordenação topológica usando uma estrutura de dados *heap*. Através do *heap* é possível satisfazer as dependências entre as atividades e ordená-las por prioridade. Neste caso, a seleção de cada próxima atividade a ser alocada é mais rápida quando se usa esta prioridade, e a satisfação das dependências é feita em $O(n \log n)$ em vez de $O(n^2)$.

As subseções seguintes apresentam as estruturas das vizinhanças implementadas neste trabalho, as quais promoverão alterações nas sequências de atividades e modos.

4.3.1 TROCA UM MODO (TUM)

Baseada no trabalho de Geiger (2013), esta vizinhança é composta por um simples movimento, que troca um modo de execução m de uma determinada atividade j de forma aleatória. Em seguida é verificado se a mudança de modo é válida ou não em relação aos recursos não renováveis. Para realizar este procedimento, é necessário receber como parâmetro um vetor s de modos de execução de todas as atividades, a posição da respectiva troca e um novo modo m . A Figura 9 mostra o movimento feito por esta vizinhança. Neste exemplo, o método recebe como parâmetro a posição 1 e um novo modo 0. Assim, o modo de execução atual 1 desta atividade é alterado para 0.

Figura 9: Exemplo da vizinhança $N_1(s)$ - Troca Um Modo (TUM)
$$s' = \text{TUM}(s, 1, 0)$$

s	0	1	2	1	1	0	1	2	1
s'	0	0	2	1	1	0	1	2	1

Fonte: elaborado pelo autor

4.3.2 TROCA DOIS MODOS (TDM)

Semelhante à Troca Um Modo (TUM) (seção 4.3.1) e baseada no trabalho de Geiger (2013), esta vizinhança tem a finalidade de alterar dois modos de execução de forma aleatória, que correspondem a duas atividades j_1 e j_2 escolhidas de forma aleatória. Ela verifica se as mudanças são válidas em relação aos recursos não renováveis. A vizinhança recebe como parâmetro um vetor s de modos de execução de todas as atividades, a posição das respectivas trocas e os novos modos m_1 e m_2 . A Figura 10 mostra o movimento realizado por esta vizinhança. Neste exemplo, o método recebe como parâmetro as posições 1 e 2, e os novos modos 0 e 1. Assim, os modos atuais são alterados para 0 e 1, respectivamente.

Figura 10: Exemplo da vizinhança $N_2(s)$ - Troca Dois Modos (TDM)
$$s' = \text{TDM}(s, 1, 2, 0, 1)$$

s	0	1	2	1	1	0	1	2	1
s'	0	0	1	1	1	0	1	2	1

Fonte: elaborado pelo autor

4.3.3 TROCA TRÊS MODOS (TTM)

Baseada no trabalho de Geiger (2013), a finalidade desta vizinhança também é mudar os modos de execução. Neste caso são alterados três modos de forma aleatória, correspondentes a três atividades: j_1 , j_2 e j_3 , escolhidas de forma aleatória. Mas devido ao alto número de combinações, serão escolhidos apenas três atividades que têm relação de precedência entre si. Ela verifica se as mudanças feitas são válidas em relação aos recursos não renováveis. A vizinhança recebe como parâmetro um vetor s de modos de execução de todas as atividades, a posição das respectivas trocas, e os novos modos m_1 , m_2 e m_3 . A Figura 11 mostra o movimento proposto por essa vizinhança. Neste exemplo, o método recebe como parâmetro as posições 1, 2 e 4, e os novos modos 0, 1 e 0. Assim, os modos de execução atuais 1, 2 e 1 dessas atividades são alterados para 0, 1 e 0, respectivamente.

Figura 11: Exemplo da vizinhança $N_3(s)$ - Troca Três Modos (TTM)

$$s' = \text{TTM}(s, 1, 2, 4, 0, 1, 0)$$

s	0	1	2	1	1	0	1	2	1
s'	0	0	1	1	0	0	1	2	1

Fonte: elaborado pelo autor

4.3.4 TROCA QUATRO MODOS (TQM)

Também baseada no trabalho de Geiger (2013), esta vizinhança tem a finalidade de mudar os modos de execução. Neste caso são alterados quatro modos de forma aleatória, correspondentes a quatro atividades escolhidas de forma aleatória. Da mesma forma, as quatro atividades: j_1, j_2, j_3 e j_4 devem ter uma relação de precedência entre si e ao final é verificada se as mudanças feitas são válidas ou não em relação aos recursos não renováveis. Para realizar este processo, é necessário receber como parâmetro um vetor s de modos de execução de todas as atividades, a posição das respectivas trocas e os novos modos m_1, m_2, m_3 e m_4 . A Figura 12 mostra o movimento feito por essa vizinhança. Neste exemplo, o método recebe como parâmetro as posições 1, 2, 4 e 5, e os novos modos 0, 1, 0 e 1. Assim, os modos de execução atuais 1, 2, 1 e 0, dessas atividades são alterados para 0, 1, 0 e 1, respectivamente.

Figura 12: Exemplo da vizinhança $N_4(s)$ - Troca Quatro Modos (TQM)

$$s' = \text{TQM}(s, 1, 2, 4, 5, 0, 1, 0, 1)$$

s	0	1	2	1	1	0	1	2	1
s'	0	0	1	1	0	1	1	2	1

Fonte: elaborado pelo autor

4.3.5 INVERTE SUBSEQUÊNCIA (INS)

Baseada no trabalho de Geiger (2013), o propósito desta vizinhança é inverter uma subsequência W de atividades. Para executar este processo, é necessário receber como parâmetro uma sequência de alocações s contendo todas as atividades e um parâmetro k indicando o tamanho da subsequência W . Uma posição z é selecionada aleatoriamente na sequência s até que a posição $z + k$ não exceda o tamanho de s . A cada posição $z + k$ inválida, o tamanho de k é decrementado até encontrar um tamanho possível para k .

Definida a subsequência W , todas as suas atividades são invertidas. Cada posição i da subsequência será trocada com a posição $k - i - 1$ da subsequência, considerando que o índice i é iniciado na posição zero. Devido ao fato de ser possível gerar sequências

inválidas por possuírem relações de precedência entre si, é necessário ao final da inversão aplicar uma ordenação topológica em s para garantir que as atividades estejam alocadas corretamente na nova sequência. A Figura 13 mostra o movimento realizado por essa vizinhança. Neste exemplo, o método recebe como parâmetro a posição 1 e o tamanho 4, ambos selecionados aleatoriamente. Assim, a subsequência é invertida.

Figura 13: Exemplo da vizinhança $N_5(s)$ - Inverte Subsequência (INS)

$$s' = \text{INS}(s, 1, 4)$$

s	2	3	5	8	14	10	7	9	12
s'	2	14	8	5	3	10	7	9	12

Fonte: elaborado pelo autor

4.3.6 DESLOCA ATIVIDADES (DEA)

A finalidade desta vizinhança é deslocar uma atividade j em uma sequência de atividades s . Neste movimento, o método recebe como parâmetro uma sequência de atividades s e um parâmetro k aleatório que determina a distância máxima que a atividade j pode ser deslocada. A atividade j é escolhida aleatoriamente. Se a atividade j for a última da sequência, o deslocamento é para a esquerda. Se j for a primeira da sequência, o deslocamento é para a direita. Se nenhum dos dois casos acontecer, a direção do deslocamento ocorrerá aleatoriamente com probabilidade de 50% de chance de ir para a direita ou para a esquerda. Escolhida a direção, será avaliado se k mais a posição da atividade j não excedem os limites da sequência. Se os limites forem extrapolados, o valor de k é decrementado até que os limites sejam respeitados. Uma vez deslocada a atividade, será feita uma ordenação topológica na sequência para garantir que a mesma seja válida. A Figura 14 mostra o movimento proposto por esta vizinhança. Neste exemplo, o método recebe como parâmetro a posição 2 e um tamanho de deslocamento igual a 3.

Figura 14: Exemplo da vizinhança $N_6(s)$ - Desloca Atividades (DEA)

$$s' = \text{DEA}(s, 2, 3)$$

s	2	3	5	8	14	10	7	9	12
s'	2	3	8	14	10	5	7	9	12

Fonte: elaborado pelo autor

4.3.7 TROCA DUAS ATIVIDADES (TDA)

Também baseada no trabalho de Geiger (2013), esta vizinhança tem como objetivo a troca de duas atividades em uma dada sequência de atividades s . As atividades j_1 e j_2 a

serem trocadas são selecionadas aleatoriamente. Este método recebe como parâmetro uma sequência de atividades s e um parâmetro k aleatório que restringe a distância máxima entre as duas atividades a serem trocadas. Ao final, a ordenação topológica é realizada para garantir que a nova sequência seja válida. A Figura 15 mostra o movimento feito por esta vizinhança. Neste exemplo, o método recebe como parâmetro as posições 2 e 4, aleatórias, que representam as atividades 5 e 14, respectivamente.

Figura 15: Exemplo da vizinhança $N_7(s)$ - Troca Duas Atividades (TDA)

$$s' = \text{TDA}(s, 2, 4)$$

s	2	3	5	8	14	10	7	9	12
s'	2	3	14	8	5	10	7	9	12

Fonte: elaborado pelo autor

4.3.8 DESLOCA PROJETO (DEP)

Esta vizinhança desloca para frente ou para trás todas as atividades de um projeto p . A vizinhança recebe como parâmetros uma sequência de atividades s , uma distância máxima de deslocamento k aleatório e um projeto p aleatório. O deslocamento ocorrerá com probabilidade de 50% de ir para a direita ou para a esquerda. Se a direção selecionada for para a esquerda, o deslocamento começará a partir da primeira atividade j pertencente ao projeto p ; caso contrário, o movimento começará a partir da última atividade j pertencente ao projeto p . Se os limites forem excedidos, o valor de k é decrementado até que os limites sejam respeitados. Por fim, a ordenação topológica é realizada para garantir que a nova sequência seja válida. A Figura 16 mostra o deslocamento realizado por esta vizinhança. Neste exemplo, o método recebe como parâmetro o projeto 4 e uma distância igual a 1. Foi sorteado o sentido do deslocamento para a direita. Todas as atividades pertencentes ao projeto p_4 estão coloridas em cinza.

Figura 16: Exemplo da vizinhança $N_8(s)$ - Desloca Projeto (DEP)

$$s' = \text{DEP}(s, 4, 1)$$

s	2	3	5	8	14	10	7	9	12
s'	3	2	8	5	7	14	10	12	9

Fonte: elaborado pelo autor

4.3.9 COMPACTA PROJETO (COP)

Esta vizinhança agrupa (compacta) as atividades de um projeto p selecionado aleatoriamente. Este movimento tenta acelerar o tempo de conclusão de um projeto. A

vizinhança recebe como parâmetro uma sequência de atividades s , um projeto p aleatório e uma porcentagem $perc \in (0, 1]$, que determina a porcentagem de atividades que serão compactadas. O projeto selecionado é agrupado da direita para a esquerda na sequência s . Uma verificação nos limites dos recursos é feita para garantir que a nova sequência seja válida. A Figura 17 mostra este movimento. Neste exemplo, o método recebe como parâmetro o projeto 2 e o percentual de compactação 0.4. Como o projeto 2 (coloridos em cinza) tem cinco atividades, a porcentagem de 40% indica uma compactação de 2 atividades. A compactação ocorrerá nas atividades 9 e 10.

Figura 17: Exemplo da vizinhança $N_9(s)$ - Compacta Projeto (COP)

$$s' = \text{COP}(s, 2, 0.4)$$

s	2	3	5	8	14	10	7	9	12
s'	2	3	5	8	14	10	9	7	12

Fonte: elaborado pelo autor

4.3.10 TROCA DOIS PROJETOS (TDP)

A finalidade desta vizinhança é agrupar e realiza a troca entre dois projetos p_1 e p_2 em uma sequência s . A vizinhança recebe como parâmetro uma sequência de atividades s , um parâmetro k aleatório que restringe a distância máxima entre os dois projetos a serem trocados, e dois projetos: p_1 e p_2 , selecionados aleatoriamente. Uma verificação nos limites dos recursos é feita para garantir que a nova sequência seja válida. A Figura 18 mostra este movimento. Neste exemplo, o método recebe como parâmetro os projetos 2 e 3. Todas as atividades dos projetos foram compactadas e suas posições são trocadas. Todas as atividades pertencentes ao projeto p_2 estão coloridas em cinza claro e as atividades pertencentes ao projeto p_3 estão coloridas em cinza escuro.

Figura 18: Exemplo da vizinhança $N_{10}(s)$ - Troca Dois Projetos (TDP)

$$s' = \text{TDP}(s, 2, 3)$$

s	5	9	3	6	13	2	1	7	25
s'	5	9	6	7	3	13	1	2	25

Fonte: elaborado pelo autor

4.3.11 COMPACTA PROJETO EXTREMO (CPE)

Proposta por [Asta et al \(2013b\)](#), esta vizinhança recebe como parâmetro de entrada uma sequência de atividades s e sua principal característica é agrupar (compactar) um

determinado projeto p no extremo. Para definir o projeto p a ser compactado, uma atividade j é selecionada aleatoriamente, e p é igual ao projeto em que esta atividade j pertence. Após isto, todas atividades da sequência pertencentes ao projeto p são compactadas e colocadas de forma adjacente à atividade j . Uma verificação em relação aos limites dos recursos é feita para garantir que a nova sequência seja válida. A Figura 19 mostra o movimento realizado por esta vizinhança. Neste exemplo, o método recebe como parâmetro a posição 4, que é a posição representada pela atividade 14 na sequência s . Todas as atividades pertencentes ao mesmo projeto da atividade 14, serão compactadas de forma adjacente a ela. As atividades pertencentes ao projeto da atividade 4 estão coloridas em cinza.

Figura 19: Exemplo da vizinhança $N_{11}(s)$ - Compacta Projeto Extremo (CPE)

$$s' = \text{CPE}(s,4)$$

s	2	3	5	8	14	10	7	9	12
s'	3	8	2	5	14	10	9	7	12

Fonte: elaborado pelo autor

4.3.12 MOVE PROJETOS (MOP)

Esta vizinhança foi proposta por [Asta et al \(2013b\)](#). Ela recebe como parâmetro uma sequência de atividades s , uma direção de movimento d aleatória, um projeto p aleatório e uma quantidade aleatória n de projetos subsequentes à p . Os projetos são compactados de forma semelhante aos da vizinhança Compacta Projeto (COP) (subseção 4.3.9). Depois de compactadas, as atividades pertencentes a cada projeto n são movidas para o início ou para o fim da sequência s de acordo com a direção d (0 para a esquerda e 1 para a direita). A escolha se o deslocamento será para o início ou fim da sequência s , é feita através de um sorteio em que as chances de se mover para qualquer direção são de 50%. Uma verificação em relação aos limites dos recursos é feita para garantir que a nova sequência seja válida. A Figura 20 mostra o movimento proposto por esta vizinhança. Neste exemplo, o método recebe como parâmetro a sequência s de atividades, a direção 0 aleatória, o projeto 1 e a quantidade de projetos subsequentes igual a 1, de modo que esses projetos são compactados e colocados em sequência no início de s . As atividades do projeto 1 estão coloridas de branco e as atividades do primeiro projeto subsequente estão coloridas na cor cinza claro. Os demais projetos estão coloridos em diferentes tonalidades de cinza mais escuro.

Figura 20: Exemplo da vizinhança $N_{12}(s)$ - Move Projetos (MOP)

$$s' = \text{MOP}(s, 0, 1, 1)$$

s	8	3	6	25	13	12	1	7	26
s'	8	12	3	13	1	6	25	7	26

Fonte: elaborado pelo autor

4.3.13 TROCA ATIVIDADES JANELA (TAJ)

Proposta por [Asta et al \(2013b\)](#), esta vizinhança seleciona uma atividade j aleatória em uma sequência s recebida como um parâmetro. Uma janela W de comprimento aleatório l é definida em uma posição aleatória entre as posições do predecessor mais próximo de j , o $posI$, e o sucessor mais próximo de j , o $posF$, de modo que a posição inicial de W seja maior que $posI$ e a posição final de W seja menor que $posF$. Em seguida, j é trocado sequencialmente com cada atividade de W , até que em uma melhoria na função objetivo aconteça. A ordenação topológica é realizada após cada alteração na janela W . Caso não ocorra uma melhoria, o método retorna a sequência recebida inicialmente. A Figura 21 mostra o movimento feito por esta vizinhança. Neste exemplo, o método recebe como parâmetro a posição 3, que é representada pela atividade 5. O predecessor mais próximo da atividade 5 é a atividade 2, e o sucessor mais próximo é a atividade 14. Com isso, são feitas as trocas sucessivas dentro da janela W . Para este exemplo, a melhora ocorreu quando a atividade 5 foi trocada com a atividade 10. As atividades de cada projeto estão em cores diferentes e o predecessor e o sucessor mais próximo possuem uma linha tracejada.

Figura 21: Exemplo da vizinhança $N_{13}(s)$ - Troca Atividades Janela (TAJ)

$$s' = \text{TAJ}(s, 3)$$

s	7	2	3	5	8	10	14	1	9
s'	7	2	3	10	8	5	14	1	9

Fonte: elaborado pelo autor

4.3.14 INSERI ATIVIDADES JANELA (IAJ)

Esta vizinhança também foi proposta por [Asta et al \(2013b\)](#) e é semelhante à Troca Atividades Janela (TAJ) (subseção 4.3.13). Ela recebe como parâmetro uma sequência de atividades s e um índice aleatório indicando a posição que terá a atividade j manipulada. Esse método trabalha com uma janela formada pelo predecessor e o sucessor mais próximo de j , porém, em vez da troca de atividades, j é inserido em cada posição da janela, deslocando as outras atividades até que uma solução melhor seja obtida. Se não houver

melhora na função objetivo quando a atividade j for inserida na primeira posição, j será inserido na próxima posição até alcançar o fim da janela W . Caso as inserções não encontrem uma solução melhor, o método retorna a sequência inicialmente recebida. A Figura 22 mostra o movimento realizado por esta vizinhança. Neste exemplo, o método recebe como parâmetro uma sequência s de atividades, a posição 2 referente a atividade 5 e faz sucessivas inserções da atividade em cada posição em W . Neste caso, a melhoria ocorreu quando a atividade 5 foi inserida na posição da atividade 8. As atividades de cada projeto estão em cores diferentes e o predecessor e o sucessor mais próximo possuem uma linha tracejada.

Figura 22: Exemplo da vizinhança $N_{14}(s)$ - Inserir Atividades Janela (IAJ)

$$s' = \text{IAJ}(s,3)$$

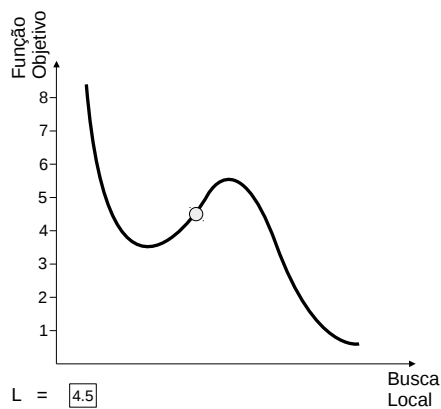
s	7	2	3	5	8	10	14	1	9
s'	7	2	3	8	5	10	14	1	9

Fonte: elaborado pelo autor

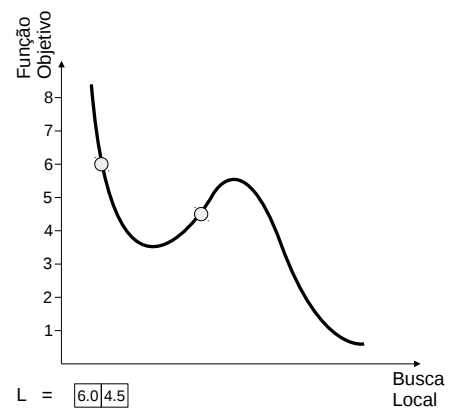
4.4 IMPLEMENTAÇÃO DO LAHC

O LAHC é conhecido como uma busca iterativa de um ponto que usa uma estratégia de aceitação tardia. De acordo com [Burke e Bykov \(2008\)](#), este método é simples de implementar porque tem apenas um parâmetro e é computacionalmente barato e eficaz para grandes problemas. O LAHC não usa parâmetros de resfriamento artificial e usa informações obtidas durante as iterações anteriores armazenadas em uma lista. Para aceitar ou não um novo candidato, seu custo é comparado com algum custo anterior pertencente à lista. A diferença entre o LAHC e o seu predecessor HC é o fato de o LAHC aceitar um conjunto de candidatos à solução, podendo estas soluções serem piores ou não em relação a melhor solução já encontrada até o momento na busca local. Essas soluções candidatas são armazenadas em uma lista L . Graficamente, alguns exemplos desse conjunto de soluções candidatas podem ser vistos na Figura 23, onde cada ponto representa uma solução que o LAHC possui em sua lista de soluções candidatas. Com esta lista, o LAHC consegue de forma satisfatória, para muitos casos, sair de ótimos locais.

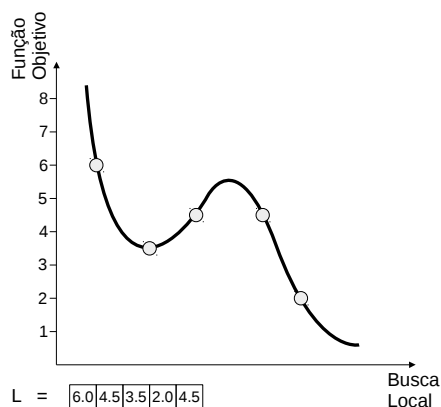
Figura 23: Exemplos: LAHC e as variações no tamanho da lista L



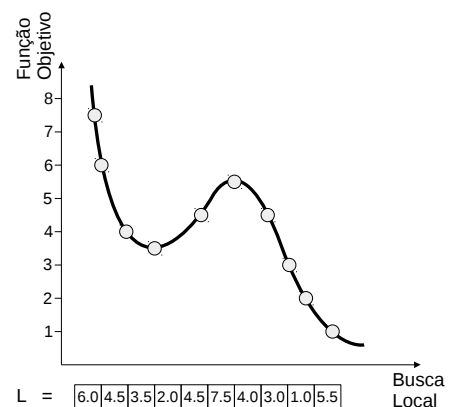
(a) Busca local com $L = 1$



(b) Busca local com $L = 2$



(c) Busca local com $L = 5$



(d) Busca local com $L = 10$

Fonte: elaborado pelo autor

O tamanho da lista L influencia diretamente na busca local. A Figura 23(a) mostra o caso de uma lista L de tamanho um. É possível notar pela Figura 23(a) que para o tamanho de L igual a um, o LAHC desempenha a mesma função do seu predecessor, o HC. A Figura 23(b) mostra o caso de uma lista L de tamanho dois, fazendo com que o LAHC ainda tenha dificuldades de sair do ótimo local no exemplo. A Figura 23(c) mostra o caso de uma lista L de tamanho cinco, proporcionando para o algoritmo uma certa facilidade para sair do ótimo local do exemplo e continuar na busca local com uma boa velocidade. A Figura 23(d) mostra o caso de uma lista L de tamanho dez, proporcionando para o algoritmo uma boa facilidade para sair do ótimo local mostrado no exemplo, porém, a busca torna-se mais lenta, pois, quanto maior o tamanho de L na busca, maior será

a demora em encontrar melhores soluções, pois, para cada solução pertencente à L , o LAHC é executado uma vez na iteração; este movimento se repete de forma cíclica sempre que o iterador do algoritmo chega na última posição de L . Por isso, determinar qual o melhor tamanho (parâmetro) para o tamanho de L não é uma tarefa fácil, uma vez que as instâncias possuem características diferentes. Achar o valor de tamanho médio para L no qual satisfaça a maioria das instâncias é uma tarefa que exige excessivos testes. O assunto sobre parâmetros e testes serão discutidos em seções posteriores.

Neste trabalho, durante a implementação do código foi necessário fazer algumas adaptações no algoritmo original do LAHC. Essas adaptações tratam-se de uma diversificação e da execução do código em paralelo. O algoritmo 1 (versão adaptada de [Burke e Bykov \(2008\)](#)) mostra o funcionamento original do LAHC.

Algoritmo 1: Proposta geral para o LAHC

```

1: Produzir uma solução inicial  $s$ 
2: Calcular o custo inicial  $C(s)$ 
3: Para todo  $k \in \{0..l - 1\}$  faça  $L_k \leftarrow C(s)$ 
4: Atribuir o número inicial de iterações  $I \leftarrow 0$ 
5: Enquanto uma condição de parada escolhida:
6:     Construir uma solução candidata  $s^*$ 
7:     Calcular o custo  $C(s^*)$ 
8:      $v \leftarrow I \bmod l$ 
9:     Se  $C(s^*) \leq L_v$  então
10:        Aceitar o candidato  $s \leftarrow s^*$ 
11:    Fim se
12:    Inserir o valor do custo dentro da lista  $L_v \leftarrow C(s)$ 
13:    Incrementar o numero de iteração  $I \leftarrow I + 1$ 
14: Fim enquanto

```

O algoritmo 2 mostra a versão implementada do LAHC neste trabalho. Como pode ser visto, para cada FO da solução pertencente à lista L o algoritmo realiza uma busca por outra solução de FO melhor do que a da posição atual de L . A lista L recebe inicialmente o valor da solução inicial em todas as suas posições. O LAHC armazena somente a FO da solução na lista L . A execução do algoritmo é realizada até que um critério de parada seja satisfeito, neste caso, o tempo de 5 minutos (300 segundos), o mesmo critério utilizado na MISTA 2013. O tempo recebido como parâmetro já leva em consideração o tempo gasto na geração da solução inicial. Em cada iteração i , é gerada uma solução candidata s' e seu custo é comparado a uma posição virtual em L . A solução s' é aceita se seu custo for menor do que o custo na posição virtual de L , ou se for menor que o custo da solução atual s . Se aceita, a solução será atualizada, se ainda possuir um custo menor que o da melhor solução encontrada até momento, s^* também será atualizada. Subsequentemente o conteúdo da posição virtual de L será atualizado com o novo custo de s .

O programa usa uma estratégia de diversificação de memória de longo prazo, que é uma variação na busca local. Essa diversificação pode ser vista em outros trabalhos com o nome de perturbação. A proposta da diversificação foi a criação de um artifício que impede a estagnação da solução após um número de iterações d sem melhorias. Para

isso, é calculado o número de vezes que cada atividade foi alterada em uma vizinhança aleatoriamente selecionada em cada iteração.

Algoritmo 2: LAHC adaptado

Entrada: solução inicial s , tamanho de lista l , tempo t , iterações sem melhorias a , iterações de diversificação d

Saída: melhor solução s^* encontrada

```

1:  $s^* \leftarrow s$ 
2:  $L_v \leftarrow C(s) \ ; \ \forall v \in \{0, \dots, |l - 1|\}$ 
3:  $J \leftarrow$  quantidade de atividades de todos os projetos;
4:  $\Delta^j = 0 \ ; \ \forall j \in \{0, \dots, |J - 1|\}$ 
5:  $i \leftarrow 0, j \leftarrow 0, m \leftarrow 0$ 
6:  $aceitar \leftarrow$  Falso
7: Enquanto tempoGasto <  $t$ 
8:   escolhe uma vizinhança aleatória  $s' \in N(s)$ 
9:    $\Gamma \leftarrow$  atividades que tenham suas posições ou modos alterados de  $s$  para  $s'$ 
10:   $v \leftarrow i \bmod l$ 
11:  Se  $i - m \leq a$  então
12:    /* iteração normal */
13:    Se  $C(s') \leq L_v$  or  $C(s') \leq f(s)$  então
14:       $aceitar \leftarrow$  Verdadeiro
15:    Fim se
16:  Senão
17:    /* iteração de diversificação */
18:     $\bar{\Delta} = \max\{\Delta^j \ \forall j \in |J - 1|\}$ 
19:     $j \leftarrow j + 1$ 
20:    Se  $C(s') - (\bar{\Delta} - \Delta^j) \leq L_v$  então
21:       $aceitar \leftarrow$  Verdadeiro
22:    Fim se
23:    Se  $j \geq d$  então
24:       $j \leftarrow 0, m \leftarrow i$ 
25:    Fim se
26:  Fim se
27:  Se  $aceitar$  então
28:     $s \leftarrow s'$ 
29:     $\Delta^j \leftarrow \Delta^j + 1 \ \forall \{j \in \Gamma\}$ 
30:    Se  $C(s) < C(s^*)$  então
31:       $s^* \leftarrow s, m \leftarrow i$ 
32:       $aceitar \leftarrow$  Falso
33:    Fim se
34:  Fim se
35:   $L_v \leftarrow C(s)$ 
36:   $i \leftarrow i + 1$ 
37: Fim enquanto
38: Retorna  $s^*$ 

```

4.5 REPOSITÓRIO DE EXPERIMENTOS

Para explorar a capacidade do algoritmo, muitos testes com diferentes parâmetros serão feitos. Para otimizar a análise dos resultados, um repositório *offline* em formato de *website* foi criado com o intuito de sumarizar os testes realizados. O *site* foi criado de forma genérica, onde é possível, caso necessite, acrescentar novos testes envolvendo novas heurísticas, atributos ou novos algoritmos.

4.5.1 PARÂMETROS E ESTRUTURA DOS TESTES

Os testes serão realizados em lotes por um *script* feito em Python. Neste *script*, todos os conjuntos de parâmetros serão recebidos e 10 execuções serão realizadas para um

tempo de 300 segundos cada.

A Tabela 1 mostra os possíveis valores de intensidade utilizados nos testes.

Tabela 1: Intensidade: parâmetros utilizados nos testes

Intensidades avaliadas													
TUM	TDM	TTM	TQM	INS	DEA	TDA	DEP	COP	TDP	CPE	MOP	TAJ	IAJ
1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0
0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0
0.5	0.6	0.7	0.8	0.9	1.0	1.0	1.0	1.0	0.9	0.8	0.7	0.6	0.5
1.0	1.0	0.8	1.0	0.6	1.0	0.4	1.0	0.2	1.0	1.0	1.0	0.1	0.1

Fonte: elaborado pelo autor

Outro parâmetro a ser configurado é o tamanho da lista L . Alguns valores empiricamente escolhidos para o tamanho desta lista podem ser vistos na Tabela 2.

Tabela 2: Lista L : parâmetros utilizados nos testes

Tamanhos para a lista L		
10	500	1000

Fonte: elaborado pelo autor

E por último, os valores aleatórios dos limites mínimos e máximos para o k presente nas vizinhanças: Inverte Subsequência (INS) (subseção 4.3.5), Desloca Atividades (DEA) (subseção 4.3.6), Troca Duas Atividades (TDA) (subseção 4.3.7) e Desloca Projeto (DEP) (subseção 4.3.8). Estes valores podem ser vistos na Tabela 3.

Tabela 3: Valores mínimos e máximos para k

Valores mínimos e máximos para k							
INS		DEA		TDA		DEP	
mín.	máx.	mín.	máx.	mín.	máx.	mín.	máx.
3	10	10	30	5	15	10	15
3	15	1	30	5	10	5	15
3	10	1	10	1	10	1	10
10	30	1	30	1	30	5	10
5	30	1	30	10	15	1	10
5	15	5	30	1	10	5	10
3	30	10	15	1	15	1	15
3	15	5	30	1	10	5	10
5	10	10	30	5	15	1	30
10	30	1	15	1	30	1	30

Fonte: elaborado pelo autor

4.5.2 AVALIAÇÃO DOS EXPERIMENTOS

Foi estipulada uma qualidade para indicar o quão bom ou ruim foram os resultados dos testes. Os valores da qualidade são referentes a um conjunto de parâmetros de entrada, ou seja, para cada conjunto de parâmetros diferentes passados para o algoritmo, uma qualidade é calculada. Isso visa a informar para qual conjunto de parâmetros o algoritmo teve melhores resultados. Caso o algoritmo tenha executado um mesmo conjunto de parâmetros mais de uma vez, o cálculo da qualidade é baseado na média das execuções. Essa qualidade é composta por dois valores: Q_{media} e Q_{desvio} . Seu cálculo tem como base as médias das FO's executadas para N instâncias e um determinado conjunto de parâmetros de entrada executados repetitivamente por n vezes. De forma analítica, a qualidade pode ser vista como:

$$Q_{inst} = \frac{1}{n} \sum_{j=1}^n \frac{B_i}{C_{i,j}} \quad (4.4)$$

$$Q_{media} = \frac{1}{N} \sum_{i=1}^N Q_{inst} \quad (4.5)$$

$$Q_{desvio} = \sqrt{\frac{1}{N-1} \sum_{i=1}^N (Q_{inst} - Q_{media})^2} \quad (4.6)$$

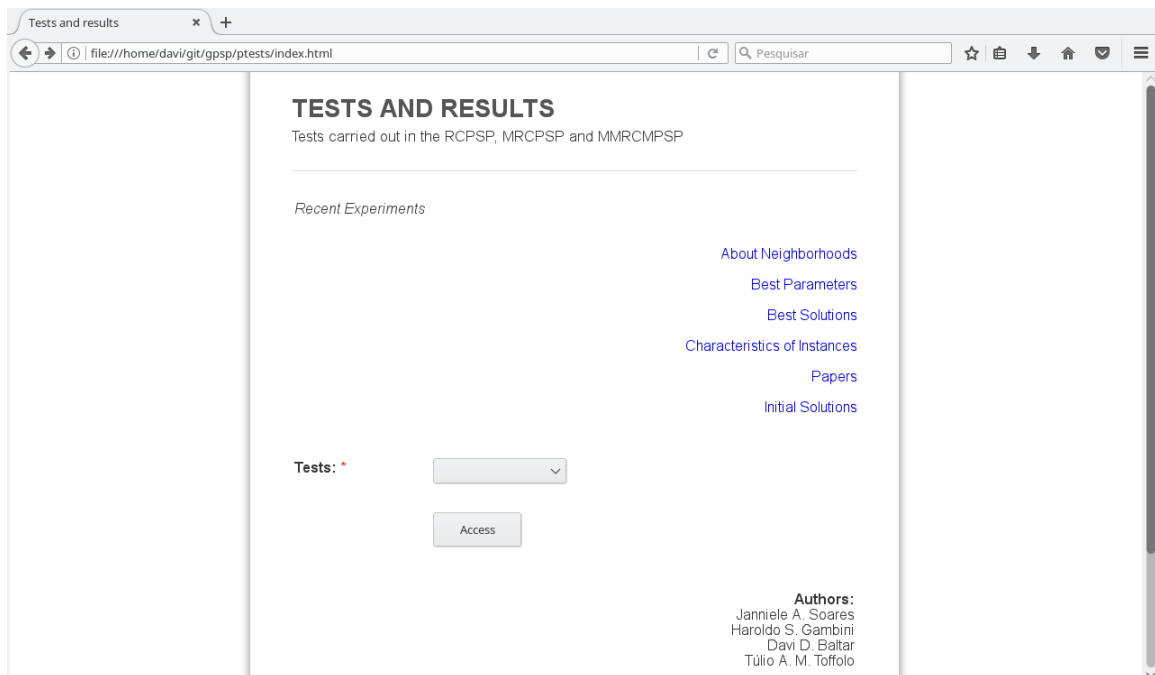
Onde, B_i é a melhor solução encontrada na literatura para a instância N_i . O termo $C_{i,j}$ é a solução encontrada em cada execução n_j para cada instância N_i .

Exemplificando, dado um conjunto hipotético contendo somente duas instâncias, C-1 e C-2, se executarmos o algoritmo duas vezes para cada uma delas utilizando o parâmetro de entrada $L = 500$ e obtivéssemos as seguintes soluções para a instância C-1: 600022 e 500045, e para a instância C-2: 800061 e 900018. Se a melhor solução conhecida para a instância C-1 fosse 400021 e para a instância C-2 fosse 700042, a média e o desvio padrão da qualidade deste teste seria 0.775 e 0.068, respectivamente.

4.5.3 REPOSITÓRIO

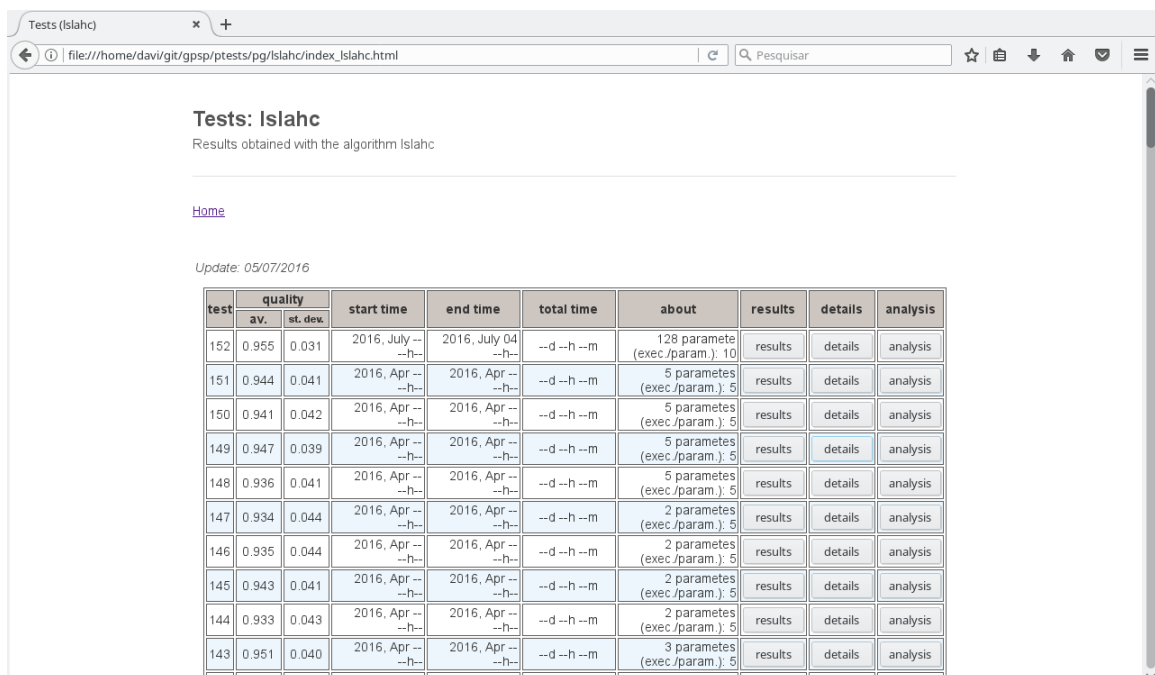
O repositório feito em página *web* foi desenvolvido com o conjunto de ferramentas: HTML, *javascript* e CSS. Ele é mantido *offline*, por isso, seu acesso não é aberto ao público geral. Para auxiliar na sumarização, um *software* foi desenvolvido totalmente em linguagem C para construir de forma automática as páginas, bastando entrar com o arquivo contendo os resultados dos testes em um determinado padrão, que o *software* realiza os cálculos, as comparações com as melhores soluções conhecidas na literatura e a construção das páginas. Em seguida, as Figuras 24 e 25 mostram a aparência do repositório.

Figura 24: Página inicial do repositório



Fonte: elaborado pelo autor

Figura 25: Sumário dos testes para o LAHC



Fonte: elaborado pelo autor

5 EXPERIMENTOS

Todos os algoritmos foram codificados em C e os modelos de programação binária foram resolvidos pelo CPLEX 12.6. O código foi compilado com o GCC 4.7.1 usando a *flag* -O3. Todos os testes foram executados em um computador com um processador Intel(R) Core(TM) i7-4960X @ 3.60GHz com 32 GB de RAM, executando o openSUSE Linux 13.2.

Detalhes sobre as instâncias podem ser vistos na Tabela 4. As colunas dessa tabela incluem informações como a quantidade de projetos $|P|$ e a quantidade de atividades $|J|$ em cada projeto. $|R|$ e $|K|$ representam respectivamente a quantidade de recursos renováveis e não renováveis existentes na instância. Também é apresentada as informações sobre o mínimo, média e máximo para as durações e o número de sucessores.

Tabela 4: Características das instâncias: A, B e X

Inst.	P	J	Recurso		Duração			Sucessor		
			R	K	mín.	méd.	máx.	mín.	méd.	máx.
A-1	2	24	3	4	1	5.533	10	1	1.500	3
A-2	2	44	3	4	1	5.058	10	1	1.850	3
A-3	2	64	3	4	1	5.600	10	1	1.833	3
A-4	5	60	6	10	1	5.340	10	1	1.500	3
A-5	5	110	6	10	1	5.803	10	1	1.850	3
A-6	5	160	6	10	1	5.180	10	1	1.833	3
A-7	10	120	2	20	1	5.683	10	1	1.500	3
A-8	10	220	2	20	1	5.440	10	1	1.850	3
A-9	10	320	11	20	1	5.435	10	1	1.833	3
A-10	10	320	11	20	1	5.602	10	1	1.833	3
B-1	10	120	11	20	1	5.580	10	1	1.500	3
B-2	10	220	2	20	1	5.645	10	1	1.850	3
B-3	10	320	11	20	1	5.495	10	1	1.833	3
B-4	15	180	16	30	1	5.702	10	1	1.500	3
B-5	15	330	16	30	1	5.511	10	1	1.850	3
B-6	15	480	16	30	1	5.511	10	1	1.833	3
B-7	20	240	21	40	1	5.498	10	1	1.500	3
B-8	20	440	2	40	1	5.445	10	1	1.850	3
B-9	20	640	21	40	1	5.580	10	1	1.833	3
B-10	20	460	2	40	1	5.457	10	1	1.800	3
X-1	10	120	2	20	1	5.403	10	1	1.500	3
X-2	10	220	11	20	1	5.500	10	1	1.850	3
X-3	10	320	11	20	1	5.500	10	1	1.833	3
X-4	15	180	2	30	1	5.586	10	1	1.500	3
X-5	15	330	16	30	1	5.360	10	1	1.850	3
X-6	15	480	16	30	1	5.462	10	1	1.833	3
X-7	20	240	21	40	1	5.265	10	1	1.500	3
X-8	20	440	21	40	1	5.520	10	1	1.850	3
X-9	20	640	21	40	1	5.359	10	1	1.833	3
X-10	20	450	21	40	1	5.474	10	1	1.800	3

Fonte: adaptado de Soares (2013)

5.1 SOLUÇÕES INICIAIS

Foram construídas soluções iniciais factíveis para as trinta instâncias da MISTA 2013 utilizando o CPLEX e o GRC. Para verificar o quão boa ou ruim foram as soluções iniciais geradas, uma tabela foi criada com o intuito de comparar as soluções iniciais geradas pelo programa com algumas outras soluções encontradas na literatura. Neste trabalho, as soluções iniciais foram comparadas com as da MISTA 2013 e do trabalho de [Asta et al \(2013b\)](#). Essa comparação pode ser vista na Tabela 5.

Tabela 5: Comparação: Soluções iniciais com as melhores da literatura

Inst.	CPLEX + GRC		MISTA		Asta et al.		B/C
	Solução Inicial		Wauters et al (2013)		Asta et al (2013b)		
	TPD	TMS	Melhor Solução		Melhor Solução		
	TPD	TMS	TPD	TMS	TPD	TMS	
A-1	26	33	1	23	1	23	0.04
A-2	13	52	2	41	2	41	0.15
A-3	12	50	0	50	0	50	0.00
A-4	150	70	65	42	65	42	0.43
A-5	289	160	153	105	150	103	0.52
A-6	353	173	147	96	133	99	0.04
A-7	1060	278	596	196	590	190	0.56
A-8	608	203	302	155	272	148	0.45
A-9	566	200	223	119	197	122	0.35
A-10	1745	443	969	314	836	303	0.48
B-1	673	185	349	127	294	118	0.44
B-2	750	207	434	160	431	158	0.57
B-3	813	246	545	210	526	200	0.65
B-4	2918	463	1274	289	1252	275	0.43
B-5	1847	333	820	254	807	245	0.44
B-6	1552	299	912	227	905	225	0.58
B-7	2441	334	792	228	782	225	0.32
B-8	5506	694	3176	533	3048	523	0.55
B-9	11722	1262	4192	746	4062	738	0.35
B-10	5127	542	3249	456	3140	436	0.61
X-1	702	207	392	142	386	137	0.55
X-2	906	257	349	163	345	158	0.38
X-3	491	229	324	192	310	187	0.63
X-4	1777	297	955	213	907	201	0.51
X-5	3620	555	1768	374	1727	362	0.48
X-6	2272	399	719	232	690	226	0.30
X-7	2505	357	861	237	831	220	0.33
X-8	3729	480	1233	283	1201	279	0.32
X-9	9254	1071	3268	643	3155	632	0.34
X-10	6955	733	1600	381	1573	383	0.23

Fonte: elaborado pelo autor

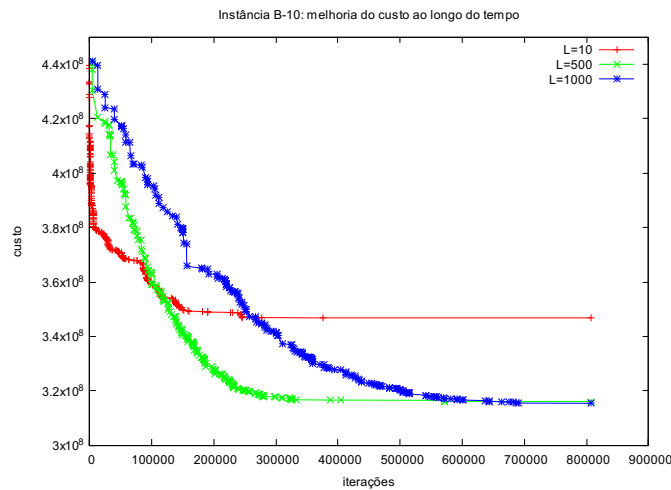
Além de esboçar as soluções iniciais e as melhores soluções encontradas na literatura para o problema, a Tabela 5 também possui uma coluna B/C , que é a divisão da FO da melhor solução entre as comparadas ([Wauters et al \(2013\)](#) e [Asta et al \(2013b\)](#)) e a FO da solução inicial. A fração B/C também pode ser vista como 1-*gap*. Lembrando que o *gap* na otimização computacional é a medida que nos diz o quão longe estamos de uma solução que desejamos alcançar. A partir da tabela, é possível ver que as soluções iniciais geradas atingem em média 40.09% das melhores soluções encontradas na literatura. Com isso, o

restante que falta para alcançar um B/C igual ou superior a 1, fica sob a responsabilidade das vizinhanças e da meta-heurística LAHC.

5.2 CONVERGÊNCIA

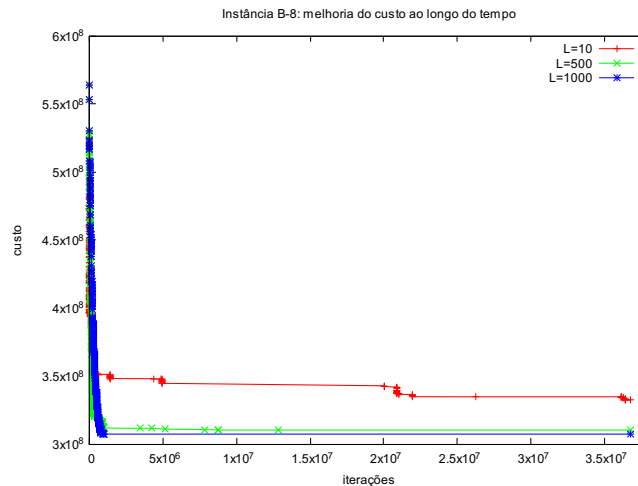
O tamanho da lista L é o único parâmetro a ser definido no LAHC. Como visto na seção 4.4, quanto maior o tamanho da lista, mais lenta é a convergência. Considerando estas informações, experimentos foram realizados a fim de avaliar a convergência do LAHC após a sua implementação. As Figuras 26 e 27, apresentam a convergência do custo das instâncias B-10 e B-8 usando apenas os tamanhos de L iguais a 10, 500 e 1000, executados por 10 horas seguidas.

Figura 26: Instância B-10 usando diferentes tamanhos de lista L .



Fonte: elaborado pelo autor

Figura 27: Instância B-8 usando diferentes tamanhos de lista L .



Fonte: elaborado pelo autor

A partir das Figuras 26 e 27, é possível observar que para $L = 500$, foram obtidos bons resultados, tanto na instância B-10 quanto na instância B-8 em tempos computacionalmente razoáveis. Considerando o limite de tempo de 300 segundos, por exemplo, usando um tamanho de lista $L = 1000$, isso resultaria em um método lento. Este mesmo comportamento também foi observado nas demais instâncias.

5.3 PARÂMETROS UTILIZADOS

O método desenvolvido funcionou em paralelo usando 4 *threads*. A quantidade 4 de *threads* foi definida com base na quantidade de núcleos reais do processador utilizado nos experimentos. Com base na seção 5.2, o tamanho 500 foi escolhido para a lista L . Os demais valores para os parâmetros foram obtidos após uma avaliação empírica e são apresentados nas Tabelas 6 e 7. Os parâmetros da Tabela Tabelas 6 correspondem, respectivamente, aos limites usados em todas as vizinhanças da seção 4.3: Troca Um Modo (TUM), Troca Dois Modos (TDM), Troca Três Modos (TTM), Troca Quatro Modos (TQM), Inverte Subsequência (INS), Desloca Atividades (DEA), Troca Duas Atividades (TDA), Desloca Projeto (DEP), Compacta Projeto (COP), Troca Dois Projetos (TDP), Compacta Projeto Extremo (CPE), Move Projetos (MOP), Troca Atividades Janela (TAJ), Inserir Atividades Janela (IAJ).

Tabela 6: Intensidade: parâmetros escolhidos

Intensidade na busca local													
TUM	TDM	TTM	TQM	INS	DEA	TDA	DEP	COP	TDP	CPE	MOP	TAJ	IAJ
1	1	0.8	1	0.6	1	0.4	1	0.2	1	1	1	0.1	0.1

Fonte: elaborado pelo autor

Após algumas avaliações prévias, a vizinhança Compacta Projeto (COP) (subseção 4.3.9) teve seu parâmetro *perc* fixado como 1.0 para todas as execuções.

Dentre as avaliações feitas, os melhores valores para o parâmetro k podem ser vistos na Tabela 7.

Tabela 7: Valores mínimos e máximos escolhidos para k

Valores mínimos e máximos escolhidos para k							
INS		DEA		TDA		DEP	
mín.	máx.	mín.	máx.	mín.	máx.	mín.	máx.
3	10	10	30	5	15	10	15

Fonte: elaborado pelo autor

5.4 RESULTADOS FINAIS

A Tabela 8 mostra os melhores resultados encontrados pela abordagem proposta, bem como a média e o desvio padrão, após 10 execuções de 5 minutos (300 segundos) cada, com 4 *threads* e tamanho da lista L igual a 500. Alguns resultados obtidos foram melhores ou iguais aos melhores resultados conhecidos na literatura. É importante ressaltar que os resultados apresentados pelo relatório técnico [Asta et al \(2013b\)](#) foram obtidos em 2500 execuções diferentes para cada instância, enquanto os resultados apresentados pela MISTA 2013 foram obtidos com apenas 10 execuções diferentes. A partir da tabela é possível ver que as execuções obtiveram uma média 0.96% para as trinta instâncias avaliadas. Isso mostra que o método se mostra eficiente para instâncias com características diferentes.

Tabela 8: Melhores resultados para o LAHC, MISTA 2013 e um relatório técnico

Inst.	LAHC Melhores Soluções		LAHC Média		LAHC Desvio Padrão		MISTA Wauters et al (2013)		Asta et al. Asta et al (2013b)		B/C
	TPD	TMS	TPD	TMS	TPD	TMS	TPD	TMS	TPD	TMS	
A-1	1	23	1.00	23.00	0	0	1	23	1	23	1.00
A-2	2	41	2.27	41.13	0	0	2	41	2	41	1.00
A-3	0	50	0	50	0	0	0	50	0	50	1.00
A-4	65	42	67.50	43.10	0.81	1.58	65	42	65	42	1.00
A-5	157	105	169.00	109.40	6.03	2.06	153	105	150	103	0.96
A-6	145	96	159.00	99.60	9.26	3.38	147	96	133	99	0.92
A-7	608	195	630.90	206.80	11.63	5.74	596	196	590	190	0.97
A-8	286	152	308.90	155.20	10.99	1.99	302	155	272	148	0.95
A-9	207	126	220.50	129.50	6.25	2.91	223	119	197	122	0.95
A-10	896	312	943.10	320.70	18.45	3.66	969	314	836	303	0.93
B-1	352	125	366.70	130.80	6.96	2.68	349	127	294	118	0.84
B-2	444	167	460.90	168.70	8.09	2.41	434	160	431	158	0.97
B-3	557	213	579.30	213.10	10.87	0.83	545	210	526	200	0.94
B-4	1276	281	1355.70	291.10	31.37	4.64	1274	289	1252	275	0.98
B-5	845	250	871.90	257.40	16.06	4.20	820	254	807	245	0.96
B-6	911	226	938.20	229.60	12.22	3.17	912	227	905	225	0.99
B-7	807	233	863.00	240.50	23.49	4.15	792	228	782	225	0.97
B-8	2974	541	3081.20	545.60	54.66	4.36	3176	533	3048	523	1.02
B-9	4630	851	4730.30	852.30	52.19	7.24	4192	746	4062	738	0.88
B-10	3050	448	3106.90	450.20	34.59	5.33	3249	456	3140	436	1.03
X-1	393	143	426.50	148.50	13.38	4.10	392	142	386	137	0.98
X-2	367	166	394.20	171.40	12.16	2.94	349	163	345	158	0.94
X-3	325	191	342.30	193.50	10.59	2.50	324	192	310	187	0.95
X-4	915	208	958.10	211.90	21.16	3.39	955	213	907	201	0.99
X-5	1786	377	1862.80	382.50	36.02	5.20	1768	374	1727	362	0.97
X-6	700	234	745.00	239.50	23.86	3.56	719	232	690	226	0.99
X-7	861	236	902.20	236.80	15.65	5.90	861	237	831	220	0.97
X-8	1218	286	1277.50	292.40	25.46	2.97	1233	283	1201	279	0.99
X-9	3475	706	3520.90	697.60	27.89	7.79	3268	643	3155	632	0.91
X-10	1652	399	1695.90	396.90	18.29	4.81	1600	381	1573	383	0.95

Fonte: elaborado pelo autor

6 CONSIDERAÇÕES FINAIS

Neste trabalho foi apresentada uma aplicação da meta-heurística LAHC para MMRCMPSP. Uma combinação de programação inteira com busca gulosa proposta por outros autores foi utilizada para ajudar a construir a solução inicial que, junto ao LAHC, algumas variações da busca local foram aplicadas. A diversificação (perturbação) no algoritmo é executada de forma controlada, de modo que o método de busca sempre salte de uma solução viável para outra. Foram feitas algumas melhorias no código, principalmente na aplicação das vizinhanças. Este trabalho conseguiu melhorar os custos de dois casos relatados na literatura por MISTA 2013. Em outros casos, o algoritmo igualou ou chegou perto o suficiente de soluções existentes. Com os resultados obtidos, um artigo (Apêndice A) foi escrito e publicado na conferência internacional MISTA 2015 (*Multidisciplinary International Scheduling Conference: Theory and Applications, 2015*).

6.1 TRABALHOS FUTUROS

Outros testes podem ser realizados para configurar mais precisamente os parâmetros de intensidade das vizinhanças e de mínimos e máximos para o k com o intuito de obter melhores resultados. Melhorias adicionais para construir o conjunto de modos para a geração da solução inicial podem ser implementadas com a criação de conjuntos independentes por projetos. Além disso, técnicas mais sofisticadas de paralelização do código podem ajudar a melhorar os resultados.

REFERÊNCIAS

- Artigues, C. e E. Hebrard (2013) Mip relaxation and large neighborhood search for a multi-mode resource-constrained multi-project scheduling problem. 6th Multidisciplinary International Conference on Scheduling : Theory and Applications (MISTA 2013) 18
- Asta, S., D. Karapetyan, A. Kheiri, E. Ozcan e A. J. Parkes (2013a) Combining Monte-Carlo and Hyper-heuristic methods for the Multi-mode Resource-constrained Multi-project Scheduling Problem. 6th Multidisciplinary International Conference on Scheduling : Theory and Applications (MISTA 2013) 17
- Asta, S., D. Karapetyan, A. Kheiri, E. Ozcan e A. J. Parkes (2013b) Combining Monte-Carlo and Hyper-heuristic methods for the Multi-mode Resource-constrained Multi-project Scheduling Problem, technical report. Tech. rep., University of Nottingham, School of Computer Science 30, 35, 36, 37, 46, 49
- Blazewicz, J., J. Lenstra e A. Kan (1983) Scheduling subject to resource constraints: classification and complexity. *Discrete Applied Mathematics* 5(1):11 – 24, DOI [http://dx.doi.org/10.1016/0166-218X\(83\)90012-4](http://dx.doi.org/10.1016/0166-218X(83)90012-4) 19
- Borba, L. M., A. J. Benavides, T. Zubarán, G. M. Carniel e M. Ritt (2013) A simple stochastic local search for multi-mode resource-constrained multi-project scheduling. 6th Multidisciplinary International Conference on Scheduling : Theory and Applications (MISTA 2013) 18
- Bresina, J. (1996) Heuristic-Biased Stochastic Sampling. pp 271–278 29
- Burke, E. e Y. Bykov (2008) A late acceptance strategy in hill-climbing for exam timetabling problems. *PATAT08 Proceedings of the 7th International Conference on the Practice and Theory of Automated Timetabling* 15, 38, 40
- Demeulemeester, E. L. e W. S. Herroelen (2002) *Project Scheduling: A Research Handbook*. Kluwer Academic Publishers, Leuven Belgium 15, 19, 20, 22, 24
- Demeulemeester, E. L., W. S. Herroelen e B. De Reyck (1998) *Handbook of Recent Advances in Project Scheduling*. Kluwer Academic Publishers 19
- Geiger, M. J. (2013) Iterated variable neighborhood search for the resource constrained multi-mode multi-project scheduling problem. 6th Multidisciplinary International Conference on Scheduling : Theory and Applications (MISTA 2013) 17, 30, 31, 32, 33
- Kolisch, R. e A. Sprecher (1996) PSPLIB - a project scheduling problem library. *European Journal of Operational Research* 96:205–216 19

- LINDEN, R. (2006) Algoritmos Genéticos. Uma importante ferramenta da Inteligência Computacional. Rio de Janeiro, RJ, BR: Brasport, 2006. ISBN 8-574-52265-1 30
- Santos, H., S. J.A. e T. Toffolo (2014) Hybrid local search for the multi-mode resource-constrained multi-project scheduling problem. PATAT '14 Proceedings of the 10th International Conference on the Practice and Theory of Automated Timetabling 29, 30
- Soares, J. A. (2013) Heurísticas baseadas em programação inteira para o problema de escalonamento de múltiplos projetos com múltiplos modos e restrições de recursos. Master's thesis, Univerdade Federal de Ouro Preto 27, 45
- Soares, J. A., H. G. Santos, D. D. Baltar e T. A. M. Toffolo (2015) Lhc applied to the multi-mode resource-constrained multi-project scheduling problem. 7th Multidisciplinary International Conference on Scheduling : Theory and Applications (MISTA 2015)
- Toffolo, T. A. M., H. G. Santos, M. A. M. Carvalho e J. A. Soares (2013) An integer programming approach for the multi-mode resource-constrained multi-project scheduling problem. 6th Multidisciplinary International Conference on Scheduling : Theory and Applications (MISTA 2013) 18, 28
- Toffolo, T. A. M., H. G. Santos, M. A. M. Carvalho e J. A. Soares (2015) An integer programming approach to the multimode resource-constrained multiproject scheduling problem. Journal of Scheduling 20
- Wauters, T., J. Kinable, P. Smet, W. Vancroonenburg, G. Berghe e J. Verschickel (2013) Mista 2013 Challenge. URL <https://gent.cs.kuleuven.be/mista2013challenge/>, access date: 8 jun. 2016 15, 23, 25, 46, 49
- Xu, J. e C. Feng (2014) Multimode resource-constrained multiple project scheduling problem under fuzzy random environment and its application to a large scale hydropower construction project. Hindawi Publishing Corporation, Vol 2014 15, 17

APÊNDICE A – LAHC applied to The Multi-Mode Resource-Constrained Multi-Project Scheduling Problem

7th Multidisciplinary International Conference on Scheduling : Theory and Applications (MISTA 2015)
25-28 August 2015, Prague, Czech Republic

MISTA 2015

LAHC applied to The Multi-Mode Resource-Constrained Multi-Project Scheduling Problem

Janniele A. Soares · Haroldo G. Santos ·
Davi D. Baltar · Túlio A. M. Toffolo

1 Introduction

The Project Scheduling Problem (PSP) consists in to schedule jobs over time in such a way that precedence relations are satisfied and resource consumption limits are respected [5]. In this paper a comprehensive variation of the PSP is considered: the Multi-Mode Resource Constrained Multi-Project Scheduling Problem (MMRCMPSP). In this problem jobs can be processed in different modes, with varying execution speeds and consuming different amounts of resources. The objective function also considers project delays.

As a generalization of the PSP, the MMRCMPSP is NP-Hard and can be used to model many problems in several areas, such as project management in information technology companies, scheduling instructions to processor architecture, civil engineering, ingot production scheduling, among others. Even the generation of an initial feasible solution for the MMRCMPSP is also NP-Hard, a feasible selection of modes must be selected, which corresponds to solving the multi-dimensional knapsack problem.

The model of MMRCMPSP [6] has a set P projects, where each $p \in P$ consists of a set $J_p = 1, \dots, |J_p|$ jobs. Each project p has a start time, where jobs can be initiated. The beginning and the end of a project are delimited by fictitious jobs.

Furthermore, the MMRCMPSP comprises a set of constraints, related with the precedence between jobs J and the consumption of resources. These resources, can be renewables R , in way local or global (shared with another projects), and non-renewable K , may to deplete throughout each project $p \in P$. Each job $j \in J$ can be performed in a certain mode $m \in M$, it determines the time taken for the execution d_{jm} and the amount of renewable and non-renewable resources consumed, respectively v_{rjm} and u_{kjm} . This consumption can not exceed the amount available of renewable q_r and non-renewable o_k resources. All relations of precedence $Pred$ and $Pred_j$ must be guaranteed.

The objective function adopted in this paper, refers to the default in MISTA 2013 Challenge [1], having two components: the TPD, main goal, which is the sum of the

J. A. Soares¹ · H. G. Santos² · D. D. Baltar³ · T. A. M. Toffolo⁴ · ^{1,3} Computing and Information Systems Department, Federal University of Ouro Preto, Brazil · ^{2,4} Computing Department, Federal University of Ouro Preto, Brazil · E-mail: janniele@decsi.ufop.br, haroldo@iceb.ufop.br, davibaltar@gmail.com, tulio@toffolo.com.br

delays of the projects in relation to the duration of the critical path and the TMS, secondary objective, which is the difference between the maximum completion time of a project and the beginning minimum time of a project.

In this work, we propose and evaluate computationally a solver based in the Late Acceptance Hill-Climbing (LAHC) [4] metaheuristic. The techniques implemented consider as input specific instances of multi-projects available on the MISTA2013 Challenge site, these instances are composed of specific projects from PSPLib.

2 Solution Approach

Our approach, differently from most approaches used in the MISTA 2013 competition, always navigates in the search space of feasible solutions. The generation of different feasible mode sets is accomplished by the solution of a series of multi-dimensional knapsack problems and the use of indirect solution representations. After building an initial solution pool, several LAHC threads start local search around those solutions.

To increase diversity, we first build a pool of different mode sets which satisfy the consumption of non-renewable resources. Since the selection of processing modes also determines the duration of jobs, a greedy strategy to prioritize the selection of fast processing modes is employed. One must observe, however, that it does not guarantee a smaller TPD, because renewable resources constraints can delay the starting time of jobs.

The binary program to build this initial set of modes considers: J jobs with respective processing times p_{jm} and N non renewable resources. Each job has a set M_j of possible modes and the non-renewable resource n consumption of job j in mode m is denoted as r_{jmn} . Thus, the binary program is solved to select which mode m job j will be allocated, considering its respective decision variables x_{jm} and resource availability q_n for each non renewable resource. Which corresponds to the NP-Hard problem of the 0-1 multidimensional knapsack problem.

The local search procedures which will be described later operate over an indirect solution representation: a solution is stored in a (I, II) ordered pair of vectors where $I_j \in M_j$ indicates the selected mode for job j and $II_j \in \{1, \dots, |J|\}$ indicates the desired position for job j in the sequence of allocations. I always respects non-renewable resources consumption. As most of the processing time of this algorithm is spent in the local search phase, the ability to quickly decode a (I, II) pair is a fundamental aspect in the performance of the algorithm.

We implemented all optimizations proposed in [2], such as prefix detection and early exploration of resource insufficiency. Differently from [2] we do not guarantee a valid topological sort in II . This speeds up the generation of valid movements, since some validations may be disabled but the cost saved is moved to the decoding phase. To transform the sorting in II into a valid topological sorting, at each new allocation one has to check the available job with highest priority (smallest desired position), which yields an $O(n^2)$ algorithm to decode II . Fortunately we devised a simple heap to speed up this to $O(n \log n)$. Initially, all jobs are inserted into the heap with priority $|S_j^-| \times |J| + II_j$.

The complete neighborhood structure $\mathcal{N}(s)$ is composed by fourteen types of movements: Change One Mode (COMS), Change Two Modes (CTMS), Change Three Modes (CTRMS), Change Four Modes (CFMS), Invert Subsequence (INVS), Shift Jobs (SJS), Swap Jobs (SWJS), Compact Project (CP), Shift Project (SPS), Swap

Table 1 Best and average results after 10 runs of the algorithm sided with [1] and [2]

Inst.	Best		Avg.		Std.Dev.		Mista		Report [2]		B/C
	TPD	TMS	TPD	TMS	TPD	TMS	TPD	TMS	TPD	TMS	
A-1	1	23	1.00	23.00	0	0	1	23	1	23	1.00
A-2	2	41	2.27	41.13	0	0	2	41	2	41	1.00
A-3	0	50	0	50	0	0	0	50	0	50	1.00
A-4	65	42	67.50	43.10	0.81	1.58	65	42	65	42	1.00
A-5	157	105	169.00	109.40	6.03	2.06	153	105	150	103	0.96
A-6	145	96	159.00	99.60	9.26	3.38	147	96	133	99	0.92
A-7	608	195	630.90	206.80	11.63	5.74	596	196	590	190	0.97
A-8	286	152	308.90	155.20	10.99	1.99	302	155	272	148	0.95
A-9	207	126	220.50	129.50	6.25	2.91	223	119	197	122	0.95
A-10	896	312	943.10	320.70	18.45	3.66	969	314	836	303	0.93
B-1	352	125	366.70	130.80	6.96	2.68	349	127	294	118	0.84
B-2	444	167	460.90	168.70	8.09	2.41	434	160	431	158	0.97
B-3	557	213	579.30	213.10	10.87	0.83	545	210	526	200	0.94
B-4	1276	281	1355.70	291.10	31.37	4.64	1274	289	1252	275	0.98
B-5	845	250	871.90	257.40	16.06	4.20	820	254	807	245	0.96
B-6	911	226	938.20	229.60	12.22	3.17	912	227	905	225	0.99
B-7	807	233	863.00	240.50	23.49	4.15	792	228	782	225	0.97
B-8	2974	541	3081.20	545.60	54.66	4.36	3176	533	3048	523	1.02
B-9	4630	851	4730.30	852.30	52.19	7.24	4192	746	4062	738	0.88
B-10	3050	448	3106.90	450.20	34.59	5.33	3249	456	3140	436	1.03
X-1	393	143	426.50	148.50	13.38	4.10	392	142	386	137	0.98
X-2	367	166	394.20	171.40	12.16	2.94	349	163	345	158	0.94
X-3	325	191	342.30	193.50	10.59	2.50	324	192	310	187	0.95
X-4	915	208	958.10	211.90	21.16	3.39	955	213	907	201	0.99
X-5	1786	377	1862.80	382.50	36.02	5.20	1768	374	1727	362	0.97
X-6	700	234	745.00	239.50	23.86	3.56	719	232	690	226	0.99
X-7	861	236	902.20	236.80	15.65	5.90	861	237	831	220	0.97
X-8	1218	286	1277.50	292.40	25.46	2.97	1233	283	1201	279	0.99
X-9	3475	706	3520.90	697.60	27.89	7.79	3268	643	3155	632	0.91
X-10	1652	399	1695.90	396.90	18.29	4.81	1600	381	1573	383	0.95

Acknowledgements The authors thank CNPq and FAPEMIG for supporting this research.

References

1. Wauters, T., Kinable, J., Smet, P., Vancroonenburg, W., Berghe, G.V. and Verstichel, J. : Mista 2013 challenge (2013). URL <http://allserv.kahosl.be/mista2013challenge/>. Access date: 2 jan. 2015
2. Asta, S., Karapetyan, D., Kheiri, A., Ozcan, E., Parkes, A.J.: Combining Monte-Carlo and Hyper-heuristic methods for the Multi-mode Resource-constrained Multi-project Scheduling Problem, technical report. Tech. rep., University of Nottingham, School of Computer Science (2013)
3. Burke, E., Bykov, Y.: A late acceptance strategy in hill-climbing for exam timetabling problems. PATAT 08 Proceedings of the 7th International Conference on the Practice and Theory of Automated Timetabling (2008)
4. Burke, E., Bykov, Y.: The Late Acceptance Hill-Climbing Heuristic. Tech. rep., University of Nottingham, School of Computer Science (2012)
5. Kolisch, R., Hartmann, S.: Experimental investigation of heuristics for resource-constrained project scheduling: An update. European Journal of Operational Research **174**(1), 23 – 37 (2006)
6. Wauters, T., Kinable, J., Smet, P., Vancroonenburg, W., Berghe, G.V., Verstichel, J.: The multi-mode resource-constrained multi-project scheduling problem. Journal of Scheduling (2014)