Novos algoritmos heurísticos e híbridos para o Problema de Escalonamento de Projetos com Restrição de Recursos Dinâmicos

André Renato Villela da Silva

Tese de Doutorado submetida ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do título de Doutor em Computação. Área de concentração: Otimização Combinatória e Inteligência Artificial.

Orientador: Luiz Satoru Ochi

Niterói, Janeiro de 2010.

Novos algoritmos heurísticos e híbridos para o Problema de Escalonamento de Projetos com Restrição de Recursos Dinâmicos

André Renato Villela da Silva

Tese de Doutorado submetida ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do título de Doutor em Computação. Área de concentração: Otimização Combinatória e Inteligência Artificial.

Aprovada por:

Prof. Luiz Satoru Ochi/ IC-UFF (Presidente)

Profa. Simone de Lima Martins / IC-UFF

Prof. Eduardo Uchoa / PEP-UFF

Prof. Nelson Maculan / COPPE-UFRJ

Profa. Claudia Boeres / DI-UFES

Niterói, Janeiro de 2010.

Sors immanis
et inanis,
rota tu volubilis,
status malus,
vana salus
semper dissolubilis,
obumbrata
et velata
michi quoque niteris;
nunc per ludum
dorsum nudum
fero tui sceleris.
Sorte monstruosa
e vazia,
tu - roda volúvel -
és má,
vã é a felicidade
sempre dissolúvel,
nebulosa
e velada
também a mim contagias;
agora por brincadeira
o dorso nu
entrego à tua perversidade.
(Trecho da obra Carmina Burana)

A minha querida família e meus amigos pelo apoio e pela paciência de me aturarem.... e aos governantes que procuram investir seriamente em educação neste país (são bem poucos, é claro).

Agradecimentos

A Deus, pelo dom da vida.

A minha família, pelo incentivo e pelo carinho.

A minha namorada, Gheise, pela abnegação e pela compreensão durante a escrita da tese.

Aos meus amigos, pela parceria e pelo apoio.

A todos os professores que se dedicam verdadeiramente a este sublime ofício do magistério.

Ao CNPq - Conselho Nacional de Pesquisa - pela bolsa de doutorado que me foi concedida para a realização deste curso, sob o número 141074/2007-8.

A todas as demais pessoas que passaram pela minha vida, meu muito obrigado por tudo o que deixaram de si e pelos inesquecíveis momentos que vivemos.... estaremos todos juntos um dia....

Resumo da Tese apresentada à UFF como requisito parcial para a obtenção do grau de Doutor em Computação (D.Sc.)

Novos algoritmos heurísticos e híbridos para o Problema de Escalonamento de Projetos com Restrição de Recursos Dinâmicos

André Renato Villela da Silva
Janeiro/2010

Orientador: Luiz Satoru Ochi
Programa de Pós-Graduação em Computação

Esta tese apresenta novos métodos para resolver o Problema de Escalonamento de Projetos com Restrições de Recursos Dinâmicos (PEPRRD). Este tipo de recurso é diferente dos demais porque é consumido quando uma tarefa do projeto é ativada, mas também é produzido ao final desta ativação. Sua quantidade máxima não é limitada como nos recursos renováveis, muito comuns em problemas de escalonamento de projetos. O objetivo do PEPRRD é maximizar a quantidade de recursos ao final de um horizonte de planejamento, por meio da ativação de tarefas consideradas lucrativas. O PEPRRD pode ser usado para modelar projetos de expansão de empresas, onde o objetivo principal é obter a maior quantidade possível de recursos. É proposta nesta tese uma nova modelagem matemática para o problema, bem como algoritmos meta-heurísticos e métodos híbridos. Alguns testes mostraram que os Algoritmos Evolutivos que utilizam uma forma específica de representação das soluções são bastante eficientes comparando com outras meta-heurísticas. Métodos híbridos que utilizam este evolutivo com o otimizador CPLEX apresentaram desempenho muito bom em várias instâncias.

Abstract of Thesis presented to UFF as a partial fulfillment of the requirements for the degree of Doctor in Science (D.Sc.)

New heuristic and hybrid algorithms for the Dynamic Resource-Constrained Project Scheduling Problem

André Renato Villela da Silva

January/2010

Advisor: Luiz Satoru Ochi
Department: Computer Science

This thesis presents new methods for solving the Dynamic Resource-Constrained Project Scheduling Problem (DRCPSP). This kind of resource is different from others because it is consumed when a project task is activated, but is also produced at the end of this activation. Its maximum amount is not bounded like the renewable resources, which are very common in project scheduling problems. The objective of DRCPSP is to maximize the amount of resources at the end of a planning horizon, through the activation of tasks considered profitable. The DRCPSP may be used to model expansion projects of companies, where the main objective is to obtain the greatest possible amount of resources. It is proposed in this thesis a new mathematical model for the problem, as well as meta-heuristic algorithms and hybrid methods. Some tests showed that the evolutionary algorithms that use a specific form of representation of the solutions are quite efficient compared with other meta-heuristics. Hybrid methods that use these evolutionary algorithms with the CPLEX optimizer had very good performance in several instances.

Palavras-chave

1. Problema de Escalonamento de Projetos
2. Algoritmos Evolutivos
3. Meta-Heurísticas
4. Métodos Híbridos
5. Formulações Matemáticas

Sumário

Resumo **v**

Abstract **vi**

Lista de Figuras **xii**

Lista de Tabelas **xiii**

1 Introdução **1**

1.1 O Problema de Escalonamento de Projetos - PEP 2

1.1.1 Recursos utilizados em Problemas de Escalonamento de Projetos . . 3

1.1.2 Produção de Recursos nos Problemas de Escalonamento de Projeto 4

1.1.3 Múltiplos Tipos de Recursos 5

1.1.4 Objetivo dos Problemas de Escalonamento de Projetos 6

1.2 O PEP tratado neste trabalho 7

1.3 Aplicações do PEPRRD 7

1.4 Objetivo e organização do trabalho 8

2 O Problema de Escalonamento de Projetos com Restrição de Recursos Dinâmicos - PEPRRD **9**

2.1 Revisão da Literatura dos PEPs 12

2.2 Literatura referente ao PEPRRD 13

3 Formulações Matemáticas para o PEPRRD 20

3.1 Menor tempo de Ativação 22

3.2 Número máximo de tarefas 24

3.3 Pares de Tarefas 25

3.4 Soma das variáveis 25

3.5 Nova formulação 26

3.6 Restrições adicionais para a segunda formulação 28

3.7 Relação entre as formulações 29

3.8 Prova da complexidade do PEPRRD 32

4 Métodos heurísticos para o PEPRRD 35

4.1 Algoritmos Evolutivos 36

4.1.1 Representação das soluções 36

4.1.2 Algoritmo Evolutivo com prioridades 37

4.1.2.1 População inicial 37

4.1.2.2 Recombinação 38

4.1.2.3 Seleção Natural 39

4.1.2.4 Mutação 39

4.1.2.5 Critério de Parada 40

4.1.3 Algoritmo Evolutivo com reconstruções 40

4.1.3.1 População inicial 40

4.1.3.2 Refinamento de Soluções 41

4.1.3.3 Busca Local 42

4.1.3.4 Critério de Parada 43

4.1.3.5 Reconstrução da População 44

4.1.4 Versão Paralela do EA_ages 45

4.2 GRASP - *Greedy Randomized Adaptive Search Procedure* 46

4.2.1 GRASP com recombinação de soluções 48

4.3 ILS - *Iterated Local Search* 48

4.3.1 Perturbações 50

4.3.2 ILS com recombinação de soluções 51

5 Algoritmos Híbridos para o PEPRRD 53

5.1 EA_ages + CPLEX 53

5.1.1 Prioridade para a fixação das variáveis 56

5.2 CPLEX como busca local 57

5.3 Escalonamentos Parciais 58

6 Resultados Computacionais 63

6.1 Instâncias 63

6.2 Formulações Matemáticas 64

6.3 Algoritmos Heurísticos 70

6.4 Métodos Híbridos 82

6.4.1 CPLEX como busca local 87

6.4.2 Intervalos de tamanho fixo 92

6.4.3 Intervalos de tamanho variável 93

6.4.4 Múltiplos intervalos 95

7 Conclusões 104

7.1 Trabalhos futuros 108

Apêndice A Generalizações para o PEPRRD 110

A.1 Modelagem com tempo de construção 110

A.2 Formulação com produção limitada 112

A.3 Aquisição de recursos extras 114

Referências **117**

Lista de Figuras

2.1 Etapas do exemplo de escalonamento 11

2.2 Exemplo de arco redundante 15

4.1 Exemplo de solução com representação direta 36

4.2 Exemplo de solução com representação indireta 37

4.3 Esquema estrutural do algoritmo EA_ages 45

5.1 Exemplo de situação onde a ativação irrestrita é prejudicial 54

5.2 Exemplo de divisão do horizonte de planejamento em 4 partições 60

5.3 Exemplo de divisão do horizonte de planejamento em 6 partições 60

5.4 Esquema de múltiplos intervalos 62

6.1 Evolução da melhor solução e da média da população 74

6.2 Análise probabilística da melhor versão de cada Meta-heurística estudada . 81

6.3 Análise probabilística das versões EA_ages +CPLEX e EA_ages +LS_cplex 90

Lista de Tabelas

2.1 Custos e lucros utilizados no exemplo 11

3.1 Configuração das variáveis para o exemplo de escalonamento do início do Capítulo 2 - formulação F1 21

3.2 Valores fracionários que justificam as restrições sobre somas das variáveis . 26

3.3 Configuração das variáveis para o exemplo de escalonamento do Capítulo 2 - formulação F2 28

4.1 Principais parâmetros dos Algoritmos Evolutivos estudados e seus valores . 45

6.1 Impacto das restrições adicionais em cada formulação 65

6.2 Limites primais obtidos pelas formulações 67

6.3 Limites duais obtidos pelas formulações 67

6.4 Tempo gasto (seg) pelas formulações 68

6.5 Resultados da utilização de *user-cuts* 70

6.6 Média dos resultados dos Algoritmos Evolutivos 71

6.7 Comparação dos resultados dos operadores de recombinação 72

6.8 Tempo médio gasto em cada recombinação (milisegundos) 72

6.9 Evolução das melhor solução ao longo das Eras 75

6.10 *Speedup* das versões *multi-thread* do evolutivo EA_ages 76

6.11 Evolução da melhor solução nas três versões do EA_ages 76

6.12 Comparação dos resultados das versões GRASP 77

6.13 Comparação dos resultados das versões ILS 78

6.14 Comparação da contribuição percentual (de 0.00 a 1.00) das perturbações e da recombinação para a geração dos soluções 79

6.15 Comparação da melhor versão de cada Meta-heuristica estudada 80

6.16 Tempo gasto e melhor solução do híbrido EA_ages +CPLEX versus CPLEX 85

6.17 Limite dual e tempo da melhor solução do híbrido EA_ages +CPLEX versus CPLEX 86

6.18 Resultados do algoritmo que usa a LS_cplex 88

6.19 Resultados dos algoritmos baseado no EA_ages em comparação com o EA3 91

6.20 Resultados para divisão do horizonte de planejamento em intervalos de tamanho fixo 97

6.21 Tempos computacionais para divisão do horizonte de planejamento em intervalos de tamanho fixo 98

6.22 Resultados para divisão do horizonte de planejamento em intervalos de tamanho variável 99

6.23 Tempos computacionais para divisão do horizonte de planejamento em intervalos de tamanho variável 100

6.24 Resultados para múltiplos intervalos de tamanho variável 101

6.25 Tempos computacionais para múltiplos intervalos de tamanho variável . . . 102

6.26 Comparação do híbrido EA_ages +CPLEX e do particionamento com intervalos de tamanho variável 103

A.1 Exemplo de valores binários para a formulação com tempo de construção . 112

A.2 Exemplo de valores binários para a formulação com produção limitada . . . 114

A.3 Exemplo de valores binários para a formulação com aquisição de recursos extras 116

Capítulo 1

Introdução

A computação vem, há décadas, ajudando o homem a resolver os mais variados tipos de problemas tanto teóricos como práticos. Partindo de um caráter predominantemente vital, onde a capacidade de decifrar códigos era responsável pela salvamento ou perda de centenas de milhares de vidas, durante a 2ª Guerra Mundial, a computação evoluiu suprindo e seguindo as necessidades acadêmicas e comerciais das décadas posteriores. Uma quase infinidade de problemas teóricos e aplicações práticas surgiu, demandando uma capacidade computacional cada vez maior. Embora *software* e *hardware* tenham se tornado muito mais eficientes, ainda há inúmeros casos onde não é possível obter uma resposta em tempo razoável e há até mesmo problemas onde sequer é possível afirmar existir uma resposta.

Os problemas computacionais são costumeiramente classificados em três grupos: (i) os problemas de decisão, onde é necessário responder sim (verdadeiro) ou não (falso) a um certo questionamento; (ii) os problemas de localização, onde é necessário encontrar uma certa estrutura dentro de um conjunto de dados de entrada; (iii) os problema de otimização, onde é necessário encontrar a estrutura mais adequada também proveniente de um conjunto de dados de entrada, isto é, a estrutura que maximize ou minimize o valor de uma função objetivo através da apresentação de uma solução ótima.

Este último grupo de problemas é de crucial importância não só na área de computação mas principalmente em ambientes industriais e/ou comerciais, em instituições de ensino, em planejamento/execução de projetos diversos e em qualquer outro segmento onde a tomada de decisão seja parte fundamental do negócio. Dentre os problemas deste grupo, existem alguns particularmente difíceis de serem resolvidos em sua otimalidade.

Estes problemas são chamados de NP-árduos (ou NP-difíceis) e abordam temas como otimização em grafos (coloração e cobertura de grafos), roteamento (ciclo hamiltoniano mínimo, problema de entrega de mercadorias com janelas de tempo), problemas de alocação (*timetabling, bin-packing*) e problemas de escalonamento (escalonamento de tarefas e projetos), para citar alguns exemplos.

A execução mais rápida de um programa paralelo, o sequenciamento da produção de mercadorias numa linha de montagem, o planejamento da expansão dos negócios de uma companhia, tudo isto tem em comum o fato de poder ser modelado com um problema de escalonamento. Por isso, grande atenção tem sido dada a este tipo de problema, principalmente na última década.

Além do clássico problema de escalonamento onde se deseja executar um conjunto de tarefas, respeitando-se relações de precedência, no menor tempo possível (também chamado de *makespam*), é muito comum encontrarmos na literatura trabalhos que levam em conta uma restrição a mais para execução das tarefas: o consumo obrigatório de certa quantidade de recursos. Neste caso temos o chamado Problema de Escalonamento de Projetos - PEP.

1.1 O Problema de Escalonamento de Projetos - PEP

Nos Problemas de Escalonamento de Projetos, dois elementos são de fundamental importância: as tarefas, que constituem cada uma das etapas do projeto a ser executado, e os recursos, que são os insumos necessários para que uma tarefa seja executada. As tarefas estão conectadas entre si através de relações de precedência que determinam a ordem em que as tarefas podem ou não ser executadas. Normalmente, estas relações são do tipo (*finish-to-start*), ou seja, é preciso que uma tarefa predecessora seja completamente executada antes de uma tarefa sucessora começar a ser. Também é muito comum que uma tarefa possa ter mais de uma predecessora. Neste caso, todas as predecessoras precisam ter sido executadas antes da tarefa em questão começar. O objetivo mais comum é fazer com que todas as tarefas do projeto sejam executadas o mais rapidamente possível, respeitando as restrições de precedência e de utilização dos recursos.

Os recursos que são necessários para a execução das tarefas são o outro elemento a

ser administrado no PEP. Cada modelo de PEP vai definir como os recursos podem ser utilizados, suas quantidades máximas, seus tipos e outras características que se fizerem necessárias. Por haver uma grande variedade recursos e de políticas de utilização, maiores explicações serão fornecidas nas subseções seguintes.

1.1.1 Recursos utilizados em Problemas de Escalonamento de Projetos

Recurso é um nome muito genérico que pode ser aplicado a praticamente todos os componentes de um PEP: as unidades de processamento utilizadas para executar as tarefas são um exemplo que muitas vezes nem sequer é notado, uma vez que, obviamente, são fundamentais para a execução das tarefas. Normalmente, os PEPs modelam aplicações práticas como sequenciamento de linhas de produção, confecção de produtos manufaturados, projetos de sistemas computacionais, construção de edificações, entre outros. Nestes casos, ao se executar uma etapa do projeto, estamos interessados no consumo ou na utilização de elementos como madeira, ferro, papel, equipamentos, maquinários, desenhistas, programadores ou qualquer outro tipo de profissional ou substrato, inclusive o dinheiro.

O consumo dos materiais citados parece ser um claro exemplo de recurso num PEP. A utilização de mão-de-obra também o é, uma vez que sem ela não seria possível executar as etapas de um projeto. Já o dinheiro é, talvez, o recurso ao qual é dada a maior atenção, pois é sempre de interesse das empresas resolver seus problemas de planejamento e/ou execução com o menor gasto possível. Os recursos, mesmo na sua grande diversidade, são frequentemente classificados pela literatura de acordo com a maneira com que participam do escalonamento em questão. Uma classificação bastante tradicional os divide em dois grupos: recursos renováveis e recursos não-renováveis.

Por recursos renováveis entendem-se aqueles que, após serem utilizados na execução de uma tarefa do projeto, ficam novamente disponíveis para serem utilizados em outra tarefa ainda não executada. Alguns exemplos de recursos desta classe são as máquinas (escavadeiras, tratores, computadores, caldeiras) e os profissionais (engenheiros, programadores, diretores, assistentes). Estes recursos podem ser reutilizados ao final de uma etapa de projeto. É bastante razoável supor que, ao terminar de escrever um módulo de um sistema computacional, tanto o programador quanto o computador utilizado por

ele estejam disponíveis para participar da criação de outro módulo. A frequência com que é possível reutilizar os recursos e, principalmente, a taxa de renovação dos recursos fazem parte do enunciado do problema. Muito comumente, a totalidade (quantidade inicialmente disponível) dos recursos fica novamente apta a ser empregada a cada intervalo mínimo de tempo considerado no problema.

Os recursos são classificados como não-renováveis se eles estiverem disponíveis uma única vez durante todo horizonte de tempo no qual deve ser tratado o problema. Uma vez que eles são utilizados (consumidos) não é mais possível contar com eles até o fim do problema. Exemplos mais comuns citados na literatura são combustíveis e dinheiro, entre outros.

Mais raramente, pode-se encontrar também um tipo de recurso que procura apresentar características tanto dos recursos renováveis quanto dos não-renováveis. Estes recursos são chamados de parcialmente renováveis [1, 2]. A principal característica deste tipo de recurso é que, durante alguns instantes t_i pré-determinados, a utilização dos recursos é modelada conforme os recursos renováveis, ou seja, os recursos utilizados nestes instantes poderão ser novamente utilizados posteriormente. Já no restante do horizonte de tempo, se os recursos forem utilizados, eles serão consumidos em definitivo e não poderão ser empregados novamente. Esta interessante característica permite que os recursos parcialmente renováveis possam modelar os outros dois tipos de recursos como casos particulares: se o subconjunto de instantes t_i for equivalente ao horizonte de planejamento, os recursos parcialmente renováveis poderiam ser tratados como renováveis - em todos os instantes não há consumo de recursos; se o subconjunto t_i for vazio, os recursos terão o mesmo comportamento dos recursos não-renováveis - em todos os instantes a utilização dos recursos supõe seu consumo.

1.1.2 Produção de Recursos nos Problemas de Escalonamento de Projeto

Tradicionalmente, os PEPs não supõem a geração e sim o consumo dos recursos que são dados de entrada com valores pré-definidos uma vez que ou estes têm caráter não-renovável ou têm sua taxa de renovação bem definida pelo problema. Estes cenários, no entanto, não são capazes de modelar certas situações onde, a partir do término da

execução de uma etapa do projeto, esta passa a gerar recursos adicionais. Como forma de ilustrar estas situações, suponha que o projeto em questão seja a expansão comercial de uma companhia. Após a construção de uma nova filial, é razoável supor que ela possa contribuir financeiramente com a matriz por meio do lucro que se espera dela. O recurso que se apresenta mais importante neste caso, sem dúvida, é o retorno financeiro. A quantia, uma vez investida na construção da filial, não fica novamente disponível ao final desta etapa ao contrário de uma máquina ou um trabalhador. O que ocorre é que após a finalização deste investimento, o mesmo passa a retornar novos recursos financeiros que poderão ser aplicados na execução de outras etapas: abertura de novas filiais, contratação de pessoal, compra de equipamentos, entre outros.

No exemplo citado, o dinheiro é um recurso renovável, mas não como nos modelos tradicionais, onde há uma quantidade máxima disponível a cada instante de tempo. Neste nosso modelo, um recurso é renovável por ser possível haver diminuição e aumento de sua quantidade disponível ao longo do tempo.

Um trabalho que aborda esta característica pode ser visto em [3]. Naquele artigo, o modelo de escalonamento permite que, após a conclusão de uma etapa i, uma quantidade de recursos r_i seja produzida e disponibilizada para o problema. Supondo que a_i seja a quantidade de recursos necessária para executar a etapa i, $a_i = r_i$ poderia modelar o recurso como renovável, $a_i > r_i$ implicaria em uma restituição incompleta e $a_i < r_i$ implicaria na produção de recursos extras.

Nem sempre, entretanto, a produção de recursos ocorre de maneira tão simples. Voltando ao exemplo da matriz que deseja abrir filiais, a produção de recursos (dinheiro, no caso) permanece desde o término da construção da filial até o fim do horizonte de planejamento em questão. Inclusive, pode-se supor que, com o passar do tempo, os recursos gerados (lucro) aumentem, diminuam ou oscilem livremente - a cada instante de tempo t, posterior à execução de uma tarefa i, são gerados $p_i(t)$ recursos.

1.1.3 Múltiplos Tipos de Recursos

Com o objetivo de generalizar ainda mais o PEP, é possível assumir a existência de mais de um tipo de recurso de forma que a execução de uma etapa só ocorra ao serem aplicados

os diversos recursos necessários, em quantidades mínimas previamente definidas. Esta generalização serve para modelar, por exemplo, etapas da produção de alimentos que precisam de farinha, água, sal, açúcar, leite, ovos, entre outras substâncias. Cada uma delas pode ser entendida como um recurso necessário à produção do alimento desejado.

Pensando de forma mais abstrata, os múltiplos recursos abrem precedentes para que se permitam formas distintas de se executar as etapas do projeto. Estas formas consumirão recursos distintos em quantidades também distintas, mas todas elas resultarão na finalização da mesma tarefa. Em outras palavras, a execução de uma tarefa i poderá ser feita de k_i formas distintas, com custos e, potencialmente, lucros também distintos. O executor do PEP em questão poderá escolher qual modo de execução será utilizado, desde que sejam respeitadas as restrições de precedência e custo, normalmente existentes em qualquer PEP. Este tipo de problema é normalmente classificado como Multi-modal [4, 5].

1.1.4 Objetivo dos Problemas de Escalonamento de Projetos

Como dito anteriormente, o objetivo mais comum de um PEP é a minimização do *makespam*, que é o tempo de encerramento do projeto como um todo. Deseja-se, portanto, completar a execução de todas as tarefas do projeto no menor tempo possível, respeitando-se, obviamente, as restrições do problema.

Em situações onde o número de tarefas é muito grande, pode haver casos onde este não seja o melhor objetivo a ser buscado. Se houver a existência de tarefas não-lucrativas, nas quais a quantidade máxima de recursos que podem ser produzidos é sempre inferior à quantidade de recursos consumidos, pode ser interessante não executar estas tarefas, de forma que a execução completa do projeto passa a ser impraticável.

Também existe a situação onde a quantidade de possibilidades (tarefas) é tão grande que não se pode esperar indefinidamente até que todas sejam executadas. Nestes casos, um objetivo interessante seria maximizar a quantidade de recursos disponíveis ao final de um instante de tempo H. Este objetivo é plausível nos problemas onde o horizonte de planejamento é dimensionado conforme um período de tempo de interesse real: o planejamento de uma semana, um mês, um ano, uma década, por exemplo.

1.2 O PEP tratado neste trabalho

Esta tese irá abordar o PEP onde as tarefas consomem recursos ao serem ativadas e, a partir de então, passam a gerar recursos até o final do horizonte de planejamento cujo tamanho é dado por um dado de entrada H. Há apenas um tipo de recurso e o modelo pressupõe ainda a existência de precedências entre as tarefas e uma quantidade inicial de recursos que poderão ser gastos nas primeiras ativações. O objetivo deste modelo é chegar ao final do horizonte de planejamento com a maior quantidade possível de recursos.

O recurso abordado neste caso não pode ter uma quantidade máxima definida, já que o objetivo citado perderia o sentido. Este recurso que é consumido e, posteriormente, produzido apresenta variações de quantidade ao longo do horizonte de planejamento que são difíceis de prever. Por isto este tipo de recurso é chamado de Recurso Dinâmico. Desta forma, o problema de escalonamento em questão será denominado Problema de Escalonamento de Projetos com Restrição de Recursos Dinâmicos- PEPRRD.

1.3 Aplicações do PEPRRD

O PEPRRD encontra aplicações potenciais em expansões comerciais ou industriais, como no citado exemplo da abertura de filiais que produzem lucro após sua abertura. Também é possível utilizar o PEPRRD em problemas de prestação de serviço ou provimento de infra-estrutura em regiões mais afastadas. Suponha, por exemplo, que se deseja expandir uma rede de fibra ótica por várias cidades do interior do estado ou do país. Não é possível, a princípio, fornecer um ponto de acesso em qualquer localidade por questões físicas, logísticas ou mesmo financeiras. A ampliação da rede deve ocorrer atendendo primeiramente as regiões onde o custo de implantação de um ponto de acesso é pequeno ou onde é possível conseguir maior lucratividade com a prestação de algum tipo de serviço (internet, telefonia, entre outros), desde que esta implantação seja tecnicamente viável, obviamente. Cria-se então uma situação de precedência entre as localidades que diferencia este problema de outros como os das p-medianas [6, 7, 8], por exemplo.

1.4 Objetivo e organização do trabalho

O objetivo principal deste trabalho é propor algoritmos e métodos, para o PEP estudado - PEPRRD, que ofereçam melhores limites duais e primais ou mesmo encontrem a solução ótima para algumas instâncias da literatura. Embora o foco não esteja na construção de métodos exatos, serão analisadas duas formulações matemáticas, na forma de programas inteiros mistos (MIP, em inglês). O ponto central do trabalho fica, portanto, na discussão sobre algoritmos (meta-)heurísticos e métodos híbridos, que incorporem características das abordagens exata e heurística.

O restante deste trabalho está organizado da seguinte forma: o Capítulo 2 traz a definição do problema e uma revisão da literatura existente. O Capítulo 3 mostrará as formulações matemáticas para o problema e como elas podem ser melhoradas. Algoritmos meta-heurísticos serão introduzidos no Capítulo 4. Métodos híbridos - que mesclam características dos algoritmos exatos e heurísticos - serão o assunto do Capítulo 5. No Capítulo 6 serão apresentados os resultados computacionais. Por fim, a conclusão e os trabalhos futuros serão apresentados nos Capítulos 7. O apêndice A introduz generalizações propostas para o Problema de Escalonamento de Projetos com Restrição de Recursos Dinâmicos.

Capítulo 2

O Problema de Escalonamento de Projetos com Restrição de Recursos Dinâmicos - PEPRRD

Antes de apresentar a definição do problema, é importante deixar claro alguns conceitos utilizados no funcionamento do modelo. Alguns deles são provenientes do clássico Problema de Escalonamento de Projetos, outros foram definidos especificamente para este modelo.

- *Tarefa*: é cada uma das etapas de um projeto que precisa ou deve ser executada. Seu dado mais importante é o instante de tempo em que ela deve ser executada.
- *Solução*: é um vetor de n números inteiros não-negativos, onde n é o número de tarefas do projeto. Cada elemento v_i do vetor indica o momento em que a tarefa i deve ser executada.
- *Recurso*: é qualquer insumo (material, equipamento, profissional ou qualquer outro item) necessário para executar cada uma das tarefas do projeto. Embora possível em vários outros modelos, o PEPRRD admite apenas um único tipo de recurso que pode acumular quantidade não-limitada de unidades ao longo da execução do projeto.
- *Recursos Disponíveis*: são os recursos que podem ser aplicados num determinado instante de tempo t para a execução de tarefas. Denota-se por Q_t.
- *Custo de uma tarefa*: é a quantidade total (positiva) de recursos necessários para que uma tarefa possa ser executada. O custo de uma tarefa i será denotado por c_i.

- *Lucro de uma tarefa*: é a quantidade não-negativa de recursos que serão produzidos pela tarefa a partir do instante de tempo seguinte a sua ativação. Será denotado por p_i para cada tarefa i do problema.

- *Lucro acumulado*: é a soma do lucro das tarefas ativadas até um instante de tempo t. A notação é feita por P_t.

- *Ativação de uma tarefa*: é a indicação de um instante de tempo t no qual a tarefa deve ser executada. Neste instante de tempo, deve haver recursos disponíveis em quantidade igual ou maior do que o custo da tarefa em questão. Também é necessário que todas as tarefas predecessoras da tarefa em questão já estejam ativadas até o instante de tempo anterior $(t-1)$.

- *Duração da tarefa*: é o tempo necessário para que uma tarefa possa ser inteiramente processada, ou seja, ter a sua execução completada. No PEPRRD, já no final do instante da sua ativação, a tarefa é considerada completada e passa, então, a produzir recursos até o final do horizonte de planejamento.

- *Horizonte de planejamento*: é o conjunto de instantes de tempo nos quais as tarefas podem ser executadas. No PEPRRD, o tempo é discretizado em unidades (instantes) de 1 até H, sendo este um dado de entrada do problema.

O PEPRRD é composto de um grafo acíclico direcionado $G = (V, A)$, onde V é um conjunto de vértices e A é o conjunto de arcos que unem os vértices. A cada tarefa $i \in V$, está associado um custo c_i e um lucro p_i, inteiros não-negativos. Inicialmente, existe uma quantidade de recursos disponíveis $Q_0 > 0$ e um lucro acumulado $P_0 = 0$. O escalonamento deverá ser realizado durante um horizonte de planejamento composto por H unidades de tempo. O objetivo do problema é maximizar a quantidade de recursos (recursos disponíveis e lucro acumulado) ao final do horizonte de planejamento.

A Figura 2.1 e a Tabela 2.1 apresentam um exemplo deste modelo de escalonamento sendo resolvido por um algoritmo arbitrário. No exemplo, o horizonte de planejamento é composto por quatro unidades ($H = 4$) e a quantidade inicial de recursos disponíveis $Q_0 = 4$. Por questões de simplificação, a quantidade de recursos disponíveis Q_t e o lucro acumulado P_t serão indicados por Q e P, respectivamente. As tarefas já ativadas estão

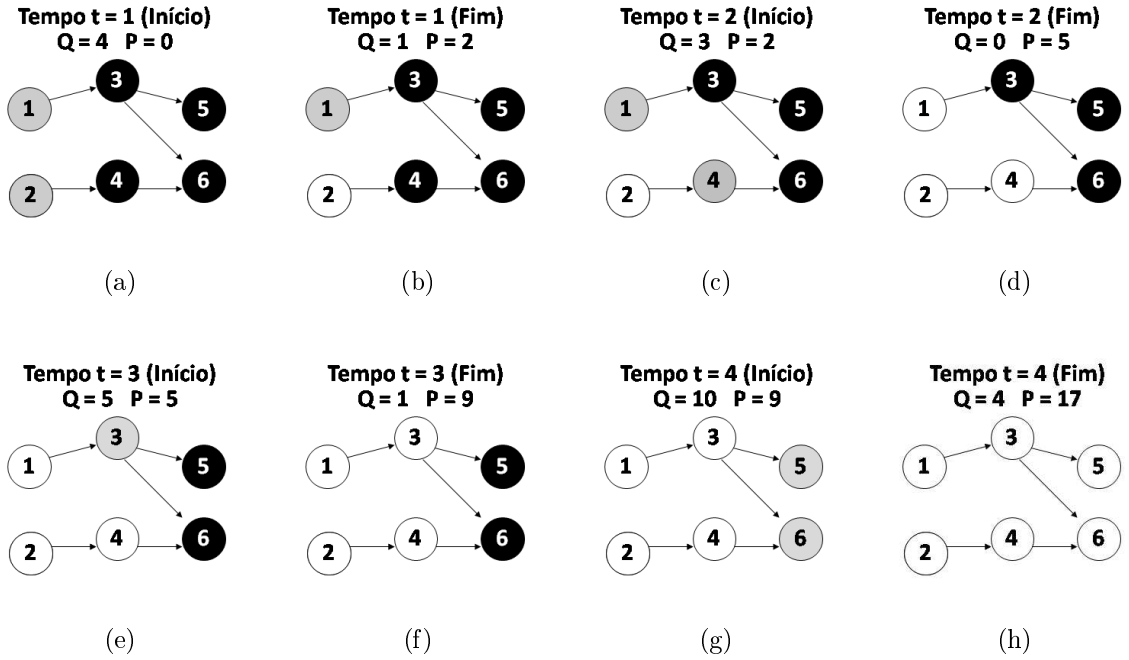


Figura 2.1: Etapas do exemplo de escalonamento

em branco, as que estão disponíveis para ativação aparecem em cinza e as que ainda não podem ser ativadas são mostradas em preto.

Tarefa	Custo	Lucro
1	2	1
2	3	2
3	4	4
4	1	2
5	2	3
6	4	5

Tabela 2.1: Custos e lucros utilizados no exemplo

No instante de tempo $t = 1$, Figura 2.1(a), as tarefas 1 e 2 estão disponíveis para ativação. Suponha que o algoritmo escolha a tarefa 2 para ser ativada. É necessário consumir 3 unidades de recursos, já que $c_2 = 3$. O lucro acumulado que inicialmente era zero passa a conter 2 unidades ($p_2 = 2$). Com o recurso disponível restante não é possível ativar a tarefa 1, logo, o tempo $t = 1$ deve ser encerrado como na Figura 2.1(b). Ao se passar para o instante seguinte ($t = 2$, Figura 2.1(c)) este lucro acumulado torna-se recurso disponível, uma vez que ele representa os recursos que estão sendo produzidos pela tarefa já ativada. Com a ativação da tarefa 2 realizada, a tarefa 4 fica apta a ser ativada.

A quantidade de recursos disponíveis, agora, permite que ambas as tarefas 1 e 4 sejam ativadas. Esta decisão faz com que sejam subtraídas 3 unidades de recursos ($c_1 + c_4 = 3$) e sejam incorporadas mais 3 unidades ($p_1 + p_4 = 3$) ao lucro acumulado, como na Figura 2.1(d). Como não há mais tarefas para serem ativadas neste momento, é necessário iniciar a transição para o instante de tempo seguinte: o lucro acumulado é somado aos recursos disponíveis e o estado das tarefas é atualizado.

No início do tempo $t = 3$ (Figura 2.1(e)), a tarefa 3 pode ser ativada já que há recursos suficientes. Atualizados os recursos disponíveis, o lucro acumulado e o estado das tarefas, chegamos ao último instante de tempo ($t = 4$, Figura 2.1(g)), quando as tarefas 5 e 6 serão ativadas. Ao final deste instante (Figura 2.1(h)), restaram quatro recursos disponíveis e o lucro acumulado é de dezessete unidades. O resultado deste escalonamento, portanto, é igual a vinte e uma unidades de recurso.

2.1 Revisão da Literatura dos PEPs

O Problema de Escalonamento de Projetos mais básico é chamado de Problema de Escalonamento de Projetos com Restrição de Recursos (PEPRR). Neste problema, o objetivo é reduzir o tempo de execução do projeto (*makespam*), respeitando as relações de precedência e os recursos disponíveis. Em [9], foi mostrado que o PEPRR é NP-árduo. Desta forma, o uso de métodos exatos ([10, 11, 12]) para resolvê-lo fica restrito a instâncias de pequeno porte. Estes métodos foram propostos como variações do algoritmo *branch-and-bound*.

Métodos heurísticos têm sido mais explorados para instâncias de maior porte. Heurísticas simples ([13, 14]) e Meta-Heurísticas ([15, 16, 17]) são muitos comuns até mesmo em generalizações para o PEPRR como a sua versão que permite a execução das tarefas em múltiplos modos ([4, 18, 5]). Esta versão permite que as tarefas sejam executadas de diversas formas, consumindo recursos e quantidades diferentes de acordo com o modo escolhido.

Recentemente, bons resultados têm sido obtidos por meio da utilização das chamadas *random-keys* ([19]), inclusive para outros problemas de otimização combinatória ([20, 21]). Nestes casos, a solução é representada por números reais que definem uma forma de

prioridade para que cada uma das tarefas seja ativada. Uma solução representada por prioridades pode ser amplamente manipulada sem causar inviabilidades na solução. Isto se deve ao fato de que qualquer conjunto de números reais pode ser igualmente tratado pelo algoritmo de escalonamento associado a esta forma de representação.

A literatura específica do PEPRRD não contemplava a utilização desta forma de representação por prioridades até este trabalho. No Capítulo 4, serão apresentados os algoritmos propostos para trabalhar com esta abordagem. No entanto, a literatura do PEPRRD será revisada na seção seguinte, para que se possa ter uma idéia dos métodos já propostos e dos seus pontos fortes e fracos.

2.2 Literatura referente ao PEPRRD

O Problema de Escalonamento de Projetos com Restrição de Recursos Dinâmicos é uma extensão do tradicional Problema de Escalonamento de Projetos com Restrição de Recursos (PEPRR). Embora muitos conceitos sejam comuns a ambos os problemas, a produção de recursos faz com quem os algoritmos propostos para o PEPRR não contemplem todas as possibilidades de escolha das tarefas que serão executadas, uma vez que é feita apenas a administração dos recursos já existentes. Esta ineficiência se torna ainda mais visível se levarmos em conta que os objetivos dos problemas são bastante diferentes.

Em [22] é apresentado o primeiro trabalho especificamente elaborado para o PEPRRD. Nele, uma heurística construtiva chamada de ADDR gera um escalonamento válido usando como critério principal a relação de lucro/custo de cada tarefa. A cada instante de tempo, uma lista é gerada contendo todas as tarefas disponíveis para ativação neste instante. A lista é ordenada de acordo com a relação custo/lucro, de forma que as tarefas mais lucrativas estejam no começo da lista. Um parâmetro α, semelhante ao utilizado em algoritmos GRASP (*Greedy Randomized Adaptive Seach Procedure*, [23]), restringe a lista apenas aos melhores elementos. Desta lista restrita, um elemento é escolhido aleatoriamente para ativação, se ainda houver recursos suficientes para tanto. A quantidade de recursos disponíveis e o lucro acumulado são atualizados; o algoritmo prossegue tentando ativar mais tarefas até que não haja mais recursos ou tarefas disponíveis. Quando isto ocorre, a única coisa a ser feita é passar para o próximo instante de tempo. Estes procedimentos são repe-

tidos até se analisar o último instante de tempo. O Algoritmo 1 mostra um pseudo-código para o método ADDR.

Algoritmo 1 ADDR

```
1: Q ← Recursos Iniciais;
2: P ← 0;
3: for t = 1 to H do
4:    Q ← Q + P; //calcula os recursos disponíveis para o próximo instante
5:    Gera-se a lista D de tarefas disponíveis;
6:    Ordena-se D de acordo com c_i/l_i; //crescentemente
7:    while (Q>0)e(|D| > 0) do
8:       Gera-se a lista L com as α melhores tarefas;
9:       k ← aleatoria(L); //escolhe e retira uma tarefa de L
10:      if c_k ≤ Q then
11:         Ativa-se k;
12:         Q ← Q − c_k; //decrementa o total de recursos
13:         P ← P + p_k; //incrementa o lucro acumulado
14:      end if
15:   end while
16: end for
17: return Q + P;
```

Vale a pena destacar que a heurística ADDR tem caráter duplamente guloso. Primeiro, por escalonar as tarefas por ordem de lucratividade; depois, por tentar utilizar todos os recursos disponíveis sem se preocupar em reservá-los ou economizá-los. Evidentemente, instâncias interessantes vão apresentar situações onde este caráter duplamente guloso resultará em um escalonamento muito ruim. Pensando em se precaver destas situações, alguns critérios adicionais foram incorporados no ponto onde se forma a lista ou se decide a tarefa a ser executada.

A primeira técnica visa evitar a ativação das tarefas que consomem mais recursos do que poderão gerar. Se no momento em que uma tarefa estiver para ser ativada, o lucro que será gerado por ela até o final do escalonamento for menor do que seu custo, esta tarefa não será ativada. Suponha que uma tarefa i tenha custo $c_i = 3$ e lucro $p_i = 2$. Se esta tarefa for ativada no último instante de tempo, ela consumirá 3 unidades e será capaz de gerar apenas 2. Obviamente esta ativação não deve ser permitida.

O critério adicional referente à primeira técnica traz uma consequência negativa: é possível existir uma tarefa não-lucrativa que seja predecessora de outra muito lucrativa. A proibição da ativação da primeira acarretará na não-ativação da segunda, o que pode

ser muito desvantajoso. Com o objetivo de suprir esta carência, estabeleceu-se um fator de folga para que o critério seja aplicado com mais inteligência: um valor real maior do que 1 (um) é multiplicado pelo lucro total que pode ser produzido pela tarefa e, então, o resultado é comparado com o custo. Assim fica permitida a ativação de tarefas que tragam um prejuízo pequeno, visando que este prejuízo seja coberto pela ativação de uma ou mais tarefas sucessoras que sejam efetivamente lucrativas.

A terceira técnica muda o critério de ordenação da lista D (Algoritmo 1). Ao invés de se utilizar o critério lucro/custo, é feito um cálculo de qual o maior lucro possível EP_i pode ser gerado por cada tarefa i, admitindo-se que elas serão ativadas o mais cedo possível - isto é feito analisando-se a topologia do grafo de entrada. Este valor EP_i é então dividido pelo custo da tarefa c_i para que se possa estabelecer a ordenação da lista D.

A última das técnicas propostas não traz impacto sobre o resultado imediato do escalonamento, mas sobre seu tempo de execução. Numa etapa de pré-processamento é feita uma varredura no grafo de entrada para se detectar arcos considerados redundantes, ou seja, arcos que impõem uma precedência já estabelecida por meio de outras tarefas, como na Figura 2.2, onde a precedência entre as tarefas 1 e 3 é dada diretamente e por meio da tarefa 2. O arco (1,3) pode ser removido pois não será possível ativar a tarefa 3 imediatamente após a tarefa 1, já que há a necessidade de se ativar também a tarefa 2 antes da tarefa 3.

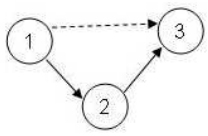


Figura 2.2: Exemplo de arco redundante

Ainda neste primeiro trabalho foram apresentadas duas buscas locais: LS1 e LS2. A primeira delas recebe um escalonamento já realizado e tenta localizar tarefas que foram ativadas em vão, ou seja, tarefas não-lucrativas que foram ativadas por causa de possíveis sucessoras mais lucrativas. Se estas sucessoras acabaram não sendo ativadas, as predecessoras foram ativadas desnecessariamente e devem ser removidas do escalonamento.

Já a busca local LS2 é uma extensão da primeira, pois analisa o conjunto de todas as sucessoras de uma tarefa i. Se este conjunto de sucessoras não estiver produzindo mais recursos do que a soma dos custos das tarefas envolvidas, todo este conjunto de tarefas deve ser removido. Os resultados computacionais mostrados naquele artigo indicam que se os parâmetros das técnicas aplicadas ao construtivo ADDR forem bem calibrados, as buscas locais conseguem melhorar muito pouco o escalonamento já realizado.

O algoritmo construtivo e as buscas locais serviram de base para a elaboração de dois algoritmos evolutivos chamados de EA1 e EA2. Ambos operam sobre uma população de tamanho fixo contendo 30 indivíduos e um critério de parada de 50 gerações. As populações são divididas em três classes compostas pelos 20% melhores individuos (classe A), pelos 20% piores indivíduos (classe C) e pelos indivíduos restantes (classe B). Um pai da classe A e outro da classe B são escolhidos. Cada tarefa do filho gerado será ativada de acordo com o menor tempo em que ela foi ativada em cada um dos pais. Por exemplo: se a tarefa 5 é ativada no primeiro pai no tempo 7 e no segundo pai no tempo 4, no filho esta tarefa será ativada no tempo 4. Apenas os 30 melhores indivíduos (entre pais e filhos) permanecem para a próxima geração.

A diferença entre os dois algoritmos evolutivos é que o primeiro não utiliza as técnicas adicionadas ao ADDR nem as buscas locais. Ele é constituído apenas do método construtivo e dos operadores evolutivos. O objetivo foi mostrar que as técnicas e as buscas locais ajudaram a gerar soluções melhores. As buscas, como já mencionado, não foram responsáveis por grandes aprimoramentos até porque as técnicas foram bem calibradas, o que permitiu a construção de soluções já bem melhoradas em relação àquelas obtidas pelo construtivo isoladamente.

A forma de representação da solução utilizada é a representação direta. Nela, uma solução é dada por um vetor de n inteiros não-negativos, onde n é o número de tarefas da instância. Cada valor indica o tempo de ativação de uma tarefa e o valor zero indica que a tarefa não foi ativada.

O segundo trabalho sobre o PEPRRD foi apresentado em [24]. O objetivo do trabalho foi propor uma solução para o principal ponto fraco dos algoritmos evolutivos anteriores: a convergência prematura. Após algumas gerações, os evolutivos tinham dificuldade em continuar melhorando a qualidade dos indivíduos por meio do operador de cruzamento.

A proposta do artigo foi utilizar outra meta-heurística onde fosse possível descartar soluções ruins ou que não pudessem ser melhoradas de maneira eficiente. A meta-heurística escolhida para substituir os algoritmos evolutivos foi o GRASP [23, 25, 26].

No GRASP, a cada iteração, uma nova solução é gerada por um método construtivo e depois aprimorada por alguma busca local. Além do construtivo ADDR, dos critérios de escolha das tarefas e das duas buscas locais, foi utilizada uma nova busca local proposta naquele artigo. A busca local chamada de LS3 aproveita parte de um escalonamento já construído e o completa utilizando um critério menos guloso do que aquele utilizado pelo construtivo ADDR.

Para que a busca local pudesse funcionar bem, era necessário que a parte do escalonamento a ser mantida não fosse pequena, pois pouca informação seria aproveitada da solução original, nem fosse muito grande, já que desta forma poucas alterações seriam possíveis em relação à solução original.

Como definir este exato ponto intermediário é algo muito custoso computacionalmente, uma forma mais simples foi empregada: escolhe-se aleatoriamente, a cada vez que a busca LS3 for utilizada, um instante de tempo TL dentro do intervalo $[\frac{H}{4}, \frac{H}{2}]$. Já o ajuste do critério guloso empregado foi feito por meio da modificação do parâmetro α fornecido para o método ADDR.

Os resultados computacionais mostrados naquele artigo indicam que a busca LS3 tem capacidade de melhorar uma solução bem maior que as buscas LS1 e LS2. As duas versões de GRASP testadas no artigo foram melhores do que os evolutivos EA1 e EA2, propostos anteriormente.

A busca local LS3 foi empregada novamente em [27], como parte de um novo algoritmo evolutivo. Desta vez, além dos operadores evolutivos básicos, foram propostos outros mecanismos para tentar evitar a convergência prematura. Cada indivíduo terá uma probabilidade de passar pela busca local LS2, de acordo com a sua aptidão - indivíduos mais aptos terão chances maiores. Como a busca local LS2 pode ser bastante custosa computacionalmente, faz pouco sentido aplicá-la indiscriminadamente a toda a população. Isto, por si só, não evita a convergência prematura, mas faz com que a população entre em estado de estagnação com um ótimo local ligeiramente melhor.

Os mecanismos de intensificação e diversificação mais específicos começam a agir quando a população passa algumas gerações sem aprimorar a melhor solução encontrada. Num primeiro lugar, a melhor solução encontrada servirá de semente para a geração de indivíduos bastante semelhantes. Usando a busca LS3, este melhor indivíduo dá origem uma população inteira de novos outros, de forma semelhante ao GRASP proposto em [24]. Esta fase funciona como uma intensificação ao redor desta solução, o que pode fazer com que melhores soluções sejam mais facilmente encontradas.

Se isto não ocorrer, porém, e mais algumas gerações se sucederem sem aprimoramentos na melhor solução, toda a população será descartada e uma nova população será criada a partir do algoritmo construtivo ADDR, como foi feito na população inicial do algoritmo evolutivo. Esta fase tem o objetivo de diversificar o foco do algoritmo evolutivo para outras soluções diferentes daquelas nas quais ele está trabalhando no momento. Um pseudo-código deste algoritmo chamado EA3 está mostrado no Algoritmo 2; os operadores evolutivos de cruzamento e seleção natural são idênticos aos propostos em [22].

Algoritmo 2 EA3

```
Mecanismo ← Intensificação;
População ← IniciaPopulação(20 indivíduos);
Geração ← 1;
while Geração ≤ 40 do
  PopAuxiliar ← Cruzamento(População);
  Popauxiliar ← LS2(PopAuxiliar);
  População ← SeleçãoNatural(População,PopAuxiliar);
  if Não Melhorou Por 4 Gerações then
    if Mecanismo = Intensificação then
      População ← Intensifica(População);
      Mecanismo ← Diversificação;
    else
      População ← Diversifica(População);
      Mecanismo ← Intensificação;
    end if
  end if
  Geração ← Geração + 1;
end while
return Melhor Indivíduo;
```

Também neste artigo é proposto um método híbrido capaz de combinar um escalonamento parcial gerado por um método exato com o evolutivo EA3. Este escalonamento parcial é obtido pelo *software* CPLEX [28], de acordo com a formulação matemática pro-

posta em [22, 24] e limitando-se a ativação das tarefas até o instante de tempo $\frac{H}{2}$, onde H é o tamanho do horizonte de planejamento. O objetivo é gerar a população inicial e as populações produzidas pela diversificação a partir do escalonamento parcial e da aplicação da busca local LS3. Com a primeira metade do escalonamento a cargo do método exato, espera-se passar para a busca local uma solução parcial de qualidade superior em relação àquelas geradas de forma heurística.

O método híbrido chamado de CPLEX+EA3 se mostrou particularmente eficiente nas instâncias que tinham um horizonte de planejamento não muito longo (até 25 unidades de tempo aproximadamente), mesmo que houvesse grande número de tarefas. Com o horizonte de planejamento mais extenso, o tempo computacional gasto pela primeira parte do escalonamento não se refletia na qualidade da solução gerada, o que fez com que este método fosse considerado inviável para estas instâncias.

Outra dificuldade que não foi bem superada encontra-se na geração dos filhos. O algoritmo de cruzamento adotado pelo EA3 - o mesmo dos primeiros algoritmos evolutivos - gasta muito tempo analisando a viabilidade dos filhos que são gerados, uma vez que a escolha dos menores tempos de ativação para as tarefas, como é feita, não assegura que os filhos gerados sejam soluções válidas, sem a execução de algoritmos de correção.

Infelizmente, são poucos os artigos que abordam os recursos dinâmicos. Em [29], é apresentado um problema de fluxo de caixa que tem algumas semelhanças como o fato de recursos serem produzidos. No entanto, os recursos empregados na execução das tarefas daquele modelo não são do mesmo tipo que os recursos produzidos, que são o foco principal do problema. Em outras palavras, é como se tivéssemos dois tipos de recursos: um que é empregado na ativação das tarefas e outro que é contabilizado na solução do problema. No caso do PEPRRD, existe apenas um tipo de recurso que é utilizado na ativação das tarefas e que também é considerado na solução do problema.

Capítulo 3

Formulações Matemáticas para o PEPRRD

Em [22], foi proposta uma modelagem matemática (F1), na forma de um programa inteiro misto, para o PEPRRD. Esta modelagem foi revisada para se tornar mais clara e é apresentada a seguir.

$$(F1) \quad Max\ Q_H + P_H \tag{3.1}$$

Sujeito à

$$x_{it} \leq \sum_{t'=1}^{t-1} x_{jt'} \quad \forall i = 1, ..., n \quad \forall t = 2, ..., H \quad \forall j \in Pred(i) \tag{3.2}$$

$$x_{i0} = 0 \quad \forall i = 1, ..., n \tag{3.3}$$

$$Q_t = Q_{t-1} + P_{t-1} - \sum_{i=1}^{n} c_i x_{it} \quad \forall t = 1, ..., H \tag{3.4}$$

$$P_t = P_{t-1} + \sum_{i=1}^{n} p_i x_{it} \quad \forall t = 1, ..., H \tag{3.5}$$

$$\sum_{t=1}^{H} x_{it} \leq 1 \quad \forall i = 1, ..., n \tag{3.6}$$

$$x_{it} \in \{0, 1\} \quad \forall i = 1, ..., n \quad \forall t = 1, ..., H \tag{3.7}$$

$$Q_t, P_t \in \mathbb{N} \quad \forall t = 0, ..., H \tag{3.8}$$

As variáveis binárias x_{it} indicam se uma tarefa i foi ativada no tempo t ou não. As variáveis Q_t e P_t indicam, respectivamente, a quantidade de recursos disponíveis e o lucro acumulado ao final de cada instante de tempo (valores inteiros não-negativos). Em (3.1), está declarado o objetivo do problema: maximizar a quantidade de recursos

ao final do horizonte de planejamento, equivalente a H instantes de tempo. As restrições (3.2) garantem que uma tarefa i só poderá ser ativada no tempo t se cada uma das suas predecessoras $j \in Pred(i)$ ($Pred(i)$ é o conjunto de predecessoras da tarefa i) tiver sido ativada até um tempo t', anterior a t. Inicialmente, todas as tarefas estão desativadas (3.3).

As restrições (3.4) mostram como a quantidade de recursos flutua ao longo do tempo. Estas restrições também impedem que, num instante qualquer, sejam gastos mais recursos do que aqueles disponíveis, pois Q_t precisa ser um número inteiro não negativo. Desta forma, o somatório do custo das tarefas que foram ativadas em t não pode ser maior do que a soma de dos recursos disponíveis ao final de $t-1$ mais o lucro acumulado até $t-1$.

As restrições (3.5) mostram que o lucro acumulado até um instante t é o lucro acumulado até o instante anterior mais a soma do lucro das tarefas ativadas neste instante. As restrições (3.6) garantem que um tarefa i só poderá ser ativada no máximo uma única vez ao longo de todo o horizonte de planejamento. As restrições (3.7) e (3.8) definem o domínio das variáveis do problema. A Tabela 3.1 mostra uma possível solução para o exemplo utilizado no início do Capítulo 2 (Fig. 2.1, página 11).

Tempo	x_{1t}	x_{2t}	x_{3t}	x_{4t}	x_{5t}	x_{6t}	Q_t	P_t
0	0	0	0	0	0	0	4	0
1	0	1	0	0	0	0	1	2
2	1	0	0	1	0	0	0	5
3	0	0	1	0	0	0	1	9
4	0	0	0	0	1	1	4	17

Tabela 3.1: Configuração das variáveis para o exemplo de escalonamento do início do Capítulo 2 - formulação F1

A formulação proposta foi muito útil para que instâncias pequenas fossem resolvidas até a sua otimalidade e os resultados fossem comparados com os métodos heurísticos apresentados em [22, 24]. Instâncias de maior porte ainda demoravam muitas horas, mesmo utilizando versões recentes de otimizadores considerados de bom desempenho [28].

A dificuldade encontrava-se na fraca relação entre as variáveis binárias relativas a uma mesma tarefa: os somatórios das variáveis deve ser menor ou igual a 1 (um). Se olharmos, portanto, uma determinada variável dessas, não é possível afirmar mais nada além do que ocorre naquele instante de tempo a que a variável se refere. Se a variável apresentar valor

igual a 1 (um), podemos afirmar que a tarefa foi ativada neste instante de tempo, mas se o valor da variável for zero, não será possível afirmar se a variável ainda não foi ativada ou se já foi ativada em algum instante de tempo anterior. Isto ocorre porque, nos instantes de tempo posteriores à ativação, o valor da variável volta a ser igual a zero da mesma forma que antes da ativação. A Tabela 3.1 ajuda a ilustrar esta situação.

Uma tentativa de melhorar o desempenho da formulação é a incorporação de restrições adicionais. Estas restrições, uma vez adicionadas, fazem com que o valor obtido pela Relaxação Linear seja mais próximo do valor ótimo, o que tende a acelerar a otimização realizada pelo algoritmo de *branch-and-cut* utilizado pelo otimizador CPLEX.

As restrições adicionais não devem impedir que uma solução ótima seja encontrada, mas tentar fazer com que algumas variáveis não tenham um valor indesejado após a Relaxação Linear. Uma maneira interessante de fazer isso é analisar a configuração das variáveis binárias e inteiras do problema logo após a Relaxação. O valor destas variáveis pode inspirar a geração das restrições adicionais que poderão ser bem aproveitadas durante a resolução do problema. Assim, várias restrições foram propostas para esta primeira formulação (F1); elas serão apresentadas a seguir bem como os conceitos que as fundamentam.

3.1 Menor tempo de Ativação

O menor tempo no qual uma tarefa pode ser ativada (*earliest time*) depende, inicialmente, de sua topologia no grafo de entrada. Uma tarefa sem predecessoras, por exemplo, pode ser ativada já na primeira unidade de tempo. Uma tarefa que possua somente predecessoras diretamente ligadas a ela (existe um arco entre elas no grafo de entrada) pode ser ativada a partir da segunda unidade de tempo, desde que todas as predecessoras tenham sido ativadas antes, e assim por diante. No entanto, no PEPRRD existe um outro fator que pode fazer esta ativação não ocorrer no tempo definido pela topologia: a quantidade de recursos disponíveis. Suponha que uma tarefa i tenham um custo K, mas não tenha predecessoras. Se a quantidade inicial de recursos Q_0 for menor do que K, esta tarefa não poderá ser ativada no primeiro instante de tempo, mas poderá ser ativada daí por diante desde que haja recursos para tanto. O menor tempo de ativação de uma tarefa é,

portanto, o primeiro instante de tempo no qual esta tarefa está apta a ser ativada.

O Algoritmo 3 apresenta um pseudo-código para o cálculo do menor tempo. Inicialmente, o menor tempo atribuído a cada tarefa é seu nível topológico (linhas 1-3). A partir de então, estipula-se uma previsão de melhor caso sobre quantos recursos estarão disponíveis a cada instante de tempo. Para o primeiro instante, a quantidade de recursos é igual a Q_0 (linha 4). A cada instante de tempo (linhas 5-24), cria-se uma lista com todas as tarefas que podem, a princípio, ser ativadas neste instante (linha 6). Cada uma das tarefas de D passa então por um teste (linha 8): se seu custo for maior do que a previsão de recursos disponíveis, seu menor tempo deve ser postergado uma unidade, bem como o menor tempo de todas as suas sucessoras (linhas 10-11); a tarefa também é removida de D (linha 9). Lá só permanecem as tarefas que puderem ser ativadas em t. Essas tarefas são copiadas para duas outras listas C e L, ordenadas respectivamente em ordem crescente e decrescente (linhas 14-15). Neste ponto, o objetivo do algoritmo é realizar a previsão sobre os recursos do próximo instante de tempo.

Essa estimativa é baseada nas tarefas que puderam ser ativadas no instante de tempo em questão. O melhor caso compreende o menor consumo e a maior produção de recursos possível. As listas C e L ordenadas como estão facilitam este cálculo, pois serão consumidos os recursos da menor para a maior quantidade; já o inverso ocorrerá com os recursos produzidos. O ponto fraco desta estimativa é que a ordem de escolha das tarefas menos custosas não reflete a mesma ordem das tarefas mais lucrativas. Encontrar, no entanto, um subconjunto de tarefas que maximize a quantidade de recursos para o tempo seguinte é semelhante ao problema da mochila [30, 31, 32], onde temos uma capacidade máxima que deverá ser preenchida por itens visando aumentar a soma do valor de cada item escolhido. Em princípio, quanto mais itens e maior valor tiverem, melhor; porém, as restrições de capacidade devem ser respeitadas, tornando o problema NP-árduo.

Ao final do Algoritmo 3 temos, portanto, uma estimativa de melhor caso sobre quando as tarefas poderão ser ativadas. Antes deste tempo, é certo que não poderão ser, o que permite fixar as respectivas variáveis com valor zero, de acordo com as Equações (3.9). Estas restrições são efetivas durante a etapa da Relaxação Linear, pois neste momento é possível que seja atribuído um valor fracionário indesejado (entre zero e um) às variáveis. Tal valor é geralmente atribuído quando não há mais recursos suficientes para pagar o

Algoritmo 3 MenorTempo

1: **for** $i = 1$ to n **do**
2: $menor(i) \leftarrow nivel(i, g)$
3: **end for**
4: $Prev \leftarrow Q_0$;
5: **for** $t = 1$ to H **do**
6: $D \leftarrow \{i \in G | menor(i) = t\}$;
7: **for** $i = 1$ to $|D|$ **do**
8: **if** $custo(D_i) > Prev$ **then**
9: $menor(D_i) = menor(D_i) + 1$;
10: $menor(D_j) = menor(D_j) + 1 \ \forall j \in Suc(D_i)$;
11: $D = D/\{D_i\}$;
12: **end if**
13: **end for**
14: $C \leftarrow L \leftarrow D$;
15: Ordena C pelo custo (crescente) e L pelo lucro (decrescente);
16: $custos \leftarrow lucros \leftarrow 0$;
17: $i \leftarrow 1$;
18: **while** $(i \leq |C|)e(custos \leq Prev)$ **do**
19: $custos \leftarrow custos + custo(C_i)$;
20: $lucros \leftarrow lucros + lucro(L_i)$;
21: $i \leftarrow i + 1$;
22: **end while**
23: $Prev \leftarrow Prev + lucros$;
24: **end for**
25: **return** $menor(.)$

custo integral de uma tarefa, assim o recurso restante é aplicado na ativação "parcial" da tarefa visando obter desde já um lucro igualmente parcial, porém impossível de ocorrer em uma solução inteira viável.

$$\sum_{t=1}^{menor(i)-1} x_{it} = 0 \qquad \forall i = 1, ..., n \tag{3.9}$$

3.2 Número máximo de tarefas

De forma indireta, ao se calcular a estimativa de recursos para cada instante de tempo é possível também determinar a quantidade máxima de tarefas que podem ser ativadas nestes instantes. Esse número estimado E_t se refere à quantidade de tarefas analisadas nas linhas 18-22 do Algoritmo 3. A estimativa é precisa apenas para o instante de tempo $t = 1$, pois é conhecida a quantidade máxima de recursos neste momento, dada por Q_0.

Com bem menos precisão, ainda assim é possível se utilizar esta estimativa E_t para se limitar a ativação das tarefas conforme as Equações (3.10).

$$\sum_{i=1}^{n} x_{it} \leq E_t \qquad \forall t = 1, ..., H \tag{3.10}$$

3.3 Pares de Tarefas

Ainda tendo em vista a impossibilidade de ativação de tarefas em determinadas situações, é interessante impedir que duas tarefas comecem a ser ativadas ao mesmo tempo se não houver recursos disponíveis para ativá-las simultaneamente. É muito comum que a Relaxação Linear consuma recursos nos primeiros instantes de tempo com ativações parciais, ou seja, utilizando recursos para pagar uma fração do custo de uma tarefa e já poder usufruir da mesma fração de seu lucro no tempo seguinte. A Relaxação Linear não tem por objetivo verificar que não seria possível realizar uma ativação completa, pois esta verificação fica a cargo da etapa de fixação das variáveis que permanecerem com valor fracionário. Como no primeiro instante de tempo, é conhecida a quantidade exata de recursos que podem ser utilizados, a análise de todos os pares de tarefas que podem ser ativadas já no primeiro instante permite determinar quais pares não poderão ter suas tarefas ativadas simultaneamente em $t = 1$. Se a soma dos custos das duas tarefas for maior do que o valor de Q_0, no máximo uma das duas poderá ser ativada. É o que se consegue ao incluir as restrições (3.11), a seguir.

$$x_{i1} + x_{k1} \leq 1 \qquad \forall \{i, k = 1, ..., n | i \neq k, menor(i) = menor(k) = 1, c_i + c_k > Q_0\} \tag{3.11}$$

3.4 Soma das variáveis

Suponha uma tarefa i e sua sucessora imediata j. O que se espera da relação entre as duas é que a tarefa i tenha valores para as variáveis binárias mais altos do que os da tarefa j, já que o razoável que se ative primeiro (e completamente) a tarefa predecessora. Como não é possível obrigar, durante a Relaxação Linear, que uma tarefa seja ativada completamente antes de se começar a ativar outra, o que se pode fazer é criar restrições

para que em momento algum uma sucessora tenha valores somados maiores do que os de uma predecessora. A Tabela 3.2 traz exemplos de valores que se enquadram na situação descrita.

Tempo	Tarefa i	Tarefa j
1	0	0
2	0.43	0
3	0	0.42
4	0	0.21
5	0	0
soma	0.43	0.63

Tabela 3.2: Valores fracionários que justificam as restrições sobre somas das variáveis

Pelos valores apresentados, nenhuma restrição original do problema foi violada, mas o comportamento das variáveis não está de acordo com o que se supõe razoável por que, no final do horizonte de planejamento, a tarefa sucessora j está "mais ativada" do que sua predecessora i. Para sanar esta situação indesejada, as Equações (3.12) podem ser adicionadas ao problema original. Por meio delas, a cada instante de tempo pertinente, o somatório das variáveis da predecessora i deve ser maior que o da sucessora j.

$$\sum_{t=1}^{T-1} x_{it} \geq \sum_{t=1}^{T} x_{jt} \qquad \forall i = 1, ..., n \quad \forall j \in Suc(i) \quad \forall \{T | menor(j) \leq T \leq H\} \tag{3.12}$$

O principal motivo de adicionar este conjunto de restrições à formulação original é buscar um limite dual mais próximo do valor ótimo. Como várias configurações de variáveis não serão mais possíveis, espera-se que o resultado da Relaxação Linear apresente um grau de inviabilidade menor. Como consequência disto, um menor número de nós da árvore do *branch-and-bound* deve ser analisado e mais rapidamente devemos encontrar a solução ótima.

3.5 Nova formulação

Um ponto negativo na formulação original é a possibilidade das variáveis de uma tarefa apresentarem valores que podem aumentar ou diminuir de um instante de tempo para outro (quando uma tarefa é ativada o valor da variável daquele instante passa a ser igual

a um, depois este valor volta a ser zero nas variáveis dos tempos seguintes). Nem sempre é possível saber, olhando para o valor de uma variável apenas, se aquela tarefa foi ou não ativada, porque ela pode ter sido anteriormente e apresentar valores zero a partir de então. Para determinar se a tarefa está ativada é preciso fazer um somatório envolvendo todas as variáveis relativas a tempos anteriores.

Tentando resolver esta situação, vale a pena recordar que o PEPRRD pressupõe que uma tarefa permanecerá ativada até o final do horizonte de planejamento. Desta forma, podemos mudar a semântica da variável binária para que ela indique não somente *quando* uma tarefa foi ativada, mas *se* ela já foi ativada ou não. Usando este raciocínio, desenvolvemos uma nova formulação (F2) que pretende manter o estado de ativação de uma tarefa em todas as variáveis binárias a partir de um momento escolhido para tanto. A formulação F2 é mostrada a seguir.

$$(F2) \;\; Max \;\; Q_H + P_H \tag{3.13}$$

Sujeito à

$$y_{it} \leq y_{it+1} \quad \forall i = 1, .., n \quad \forall t = 1, ..., H-1 \tag{3.14}$$

$$y_{it} \leq y_{jt-1} \quad \forall i = 1, .., n \quad \forall t = 1, ..., H \quad \forall j \in Pred(i) \tag{3.15}$$

$$Q_t = Q_{t-1} + P_{t-1} - \sum_{i=1}^{n} c_i(y_{it} - y_{it-1}) \quad \forall t = 1, ..., H \tag{3.16}$$

$$P_t = \sum_{i=1}^{n} p_i y_{it} \quad \forall t = 0, ..., H \tag{3.17}$$

$$y_{i0} = 0 \quad \forall i = 1, ..., n \tag{3.18}$$

$$y_{it} \in \{0, 1\} \quad \forall i = 1, ..., n \quad \forall t = 1, ..., H \tag{3.19}$$

$$Q_t, P_t \in N \quad \forall t = 0, ..., H \tag{3.20}$$

A variável binária y_{it} indica se a tarefa i está ativada no tempo t (valor 1) ou não (valor 0). A partir do momento em que a tarefa é ativada, as variáveis binárias referentes aos tempos seguintes também serão fixadas em 1, através das restrições (3.14). Se uma tarefa ainda não tiver sido ativada, as variáveis dos tempos seguintes podem ser 0 ou 1. As restrições (3.15) garantem que uma tarefa i só poderá ser ativada caso todas as predecessoras j já estejam ativadas anteriormente, pelo menos no tempo $t-1$. As

restrições (3.16) mostram como é definida a quantidade recursos ao final de cada instante de tempo t. É interessante notar que a diferença $y_{it} - y_{it-1}$ indica se uma tarefa acabou de ser ativada ou não. Se esta diferença é zero, ou a tarefa já estava ativada antes e permanece neste estado (1-1=0), ou a tarefa não estava ativada e assim continua (0-0=0). A diferença só será igual a 1 se a tarefa não estava ativada, mas passou a estar (1-0=1). As restrições (3.20) impedem que a diferença seja negativa. O cálculo do lucro acumulado é feito pelas restrições (3.17), indicando que o este valor é dado pela soma dos lucros de todas as tarefa já ativadas até o momento. As restrições (3.18) indicam que no início do problema nenhuma tarefa está ativada. As restrições (3.19) e (3.20) definem o domínio das variáveis. O valor de Q_0 é dado de entrada do problema.

A grande vantagem desta formulação é que ela é constituída na sua quase totalidade de restrições do tipo $a \leq b$, que são aritmeticamente muito simples e deixam a matriz de coeficientes muito esparsa. Esta característica permite que o otimizador descarte rapidamente as restrições redundantes e economize bastante memória. Além disso, a busca por uma base ótima no método Simplex pode ser mais eficiente. A configuração das variáveis relativas ao exemplo do Capítulo 2, para esta segunda formulação, pode ser vista na Tabela 3.3.

Tempo	y_{1t}	y_{2t}	y_{3t}	y_{4t}	y_{5t}	y_{6t}	Q_t	P_t
0	0	0	0	0	0	0	4	0
1	0	1	0	0	0	0	1	2
2	1	1	0	1	0	0	0	5
3	1	1	1	1	0	0	1	9
4	1	1	1	1	1	1	4	17

Tabela 3.3: Configuração das variáveis para o exemplo de escalonamento do Capítulo 2 - formulação F2

3.6 Restrições adicionais para a segunda formulação

De forma semelhante à formulação original para o PEPRRD, é possível incluir restrições adicionais também em F2. O menor tempo de ativação e a restrição aos pares de tarefas com custos somados maior do que a quantidade de recursos iniciais são de aplicação imediata. Podem ser utilizadas as Equações (3.9) e (3.11) sem maiores alterações.

O objetivo das restrições que limitam o número de tarefas a ser ativadas em cada

instante de tempo é impedir que a Relaxação Linear ative parcial ou integralmente mais tarefas do que poderiam estar presentes em uma solução viável, de acordo com estimativas de melhor caso. Na segunda formulação a aplicação das Equações (3.10) deve ser ligeiramente alterada, pois o valor de uma variável igual a um indica que ela pode ter sido ativada no instante em questão ou anteriormente. Para saber se uma tarefa foi ativada exatamente num instante t, é necessário realizar a subtração com o valor da variável no instante $t-1$. Analogamente, para saber quantas tarefas foram ativadas a cada instante é necessário realizar uma subtração de somas como nas Equações (3.21), a seguir, onde E_t representa a estimativa de quantas tarefas poderão ser ativadas no máximo a cada instante de tempo t.

$$\sum_{i=1}^{n} y_{it} - \sum_{i=1}^{n} y_{it-1} \leq E_t \qquad \forall t = 1, ..., H \tag{3.21}$$

Já as restrições referentes às somas das variáveis são um caso à parte. O problema que existia em F1, onde era possível que uma tarefa sucessora apresente valores para as variáveis maiores do que os da predecessora, não existe na formulação F2. As restrições (3.14) e (3.15) conjuntamente já obrigam que a predecessora tenha sempre valores iguais ou maiores do que a sucessora, com a diferença de que a predecessora deve começar a ser ativada antes. Logo, as restrições das somas já estão incorporadas na própria formulação, o que é um indício de que esta segunda formulação tende a ser melhor do que a primeira.

3.7 Relação entre as formulações

As variáveis inteiras Q_t e P_t têm o mesmo significado nas duas formulações apresentadas. O mesmo já não pode ser dito sobre as variáveis binárias x_{it} e y_{it}. No entanto, em ambas as formulações, uma solução pode ser reproduzida utilizando apenas o valor destas variáveis binárias - através delas é possível recalcular todos os valores de Q_t e P_t. É interessante, portanto, estabelecer uma relação entre as variáveis binárias das duas formulações para mostrar que toda solução viável obtida por qualquer formulação é válida também para a outra.

$$y_{it'} = \sum_{t=1}^{t'} x_{it} \qquad \forall i = 1, ..., n \quad \forall t' = 1, ..., H \tag{3.22}$$

As Equações 3.22 mostram que a variável $y_{it'}$ pode ser descrita como a soma das variáveis x_{it} até o instante t'. De fato, no momento da ativação da tarefa i, na primeira formulação a variável passa a valer um e depois volta a zero. Na segunda formulação, ela também passa a valer um, mas depois precisa manter este valor. Já a contra-parte das Equações 3.22 pode ser vista pelas Equações 3.23. Nelas, a ativação é dada pela subtração em relação ao tempo anterior. As Tabelas 3.1 e 3.3, mostradas anteriormente, servem como exemplo para as Equações apresentadas.

$$x_{it} = y_{it} - y_{it-1} \qquad \forall i = 1, ..., n \quad \forall t = 1, ..., H \tag{3.23}$$

Para que fique ainda mais clara a possibilidade de transformação de uma formulação em outra, será mostrado, a seguir, como cada restrição pode ser convertida para a formulação desejada. Para que esta conversão fique correta, será necessário adotar as restrições 3.12 no lugar das restrições 3.2. Isto ocorre, porque as restrições de soma das variáveis são uma versão mais forte das restrições de precedência expressas na formulação F1.

Desta forma, tomando as equações 3.12 e 3.23, temos:

$$\sum_{t=1}^{T-1}(y_{it} - y_{it-1}) \geq \sum_{t=1}^{T}(y_{jt} - y_{jt-1}) \quad \forall i = 1, ..., n \quad \forall j \in Suc(i) \quad \forall\{T|menor(j) \leq T \leq H\} \tag{3.24}$$

$$y_{iT-1} \geq y_{jT} \forall i = 1, ..., n \quad \forall j \in Suc(i) \quad \forall\{T|menor(j) \leq T \leq H\} \tag{3.25}$$

A equação 3.25 é semelhante à equação 3.14, com os índices i e j trocados. A conversão da equação 3.4, na equação 3.16 é imediata com a aplicação da equação 3.23. Tomando as equações 3.5 e 3.23, temos:

$$P_t = P_{t-1} + \sum_{i=1}^{n} p_i(y_{it} - y_{it-1}) \quad \forall t = 1, ..., H \tag{3.26}$$

$$P_t = P_{t-1} + \sum_{i=1}^{n} p_i y_{it} - \sum_{i=1}^{n} y_{it-1} \quad \forall t = 1, ..., H \tag{3.27}$$

Trabalhando a recursão em 3.27, chegamos a:

$$P_t = \sum_{i=1}^{n} p_i y_{it} \quad \forall t = 1, ..., H \tag{3.28}$$

que é exatamente a equação 3.17.

A volta das equações, isto é, a conversão da formulação F2 na formulação F1 é imediata aplicando-se a equação 3.22 a algum algebrismo simples. A conversão da equação 3.17 na equação 3.5 é mais complicada e será demonstrada seguir.

$$P_t = \sum_{i=1}^{n} (p_i \sum_{t'=1}^{t} x_{it'}) \quad \forall t = 1, ..., H \tag{3.29}$$

$$P_t = \sum_{i=1}^{n} p_i (x_{it} + \sum_{t'=1}^{t-1} x_{it'}) \quad \forall t = 1, ..., H \tag{3.30}$$

$$P_t = \sum_{i=1}^{n} p_i x_{it} + \sum_{i=1}^{n} p_i (\sum_{t'=1}^{t-1} x_{it'}) \quad \forall t = 1, ..., H \tag{3.31}$$

Pela equação 3.29 temos:

$$P_t = \sum_{i=1}^{n} p_i x_{it} + P_{t-1} \quad \forall t = 1, ..., H \tag{3.32}$$

que é idêntica à equação 3.5.

Resumindo a transformações:

$3.12 \overset{3.23}{\rightarrow} 3.15 \overset{3.22}{\rightarrow} 3.12$

$3.4 \overset{3.23}{\rightarrow} 3.16 \overset{3.22}{\rightarrow} 3.4$

$3.5 \overset{3.23}{\rightarrow} 3.17 \overset{3.22}{\rightarrow} 3.5$

As demais restrições não precisam ser convertidas ou levam a equações redundantes. O potencial de cada formulação bem como o impacto da inclusão das respectivas restrições

adicionais serão discutidos no Capítulo 6, sobre resultados computacionais.

3.8 Prova da complexidade do PEPRRD

O objetivo desta seção é mostrar que o PEPRRD pode ser transformado no problema da Mochila (*Knapsack Problem*), bastante conhecido [30]. Tomemos, inicialmente, a formulação matemática F2, reproduzida abaixo.

$$(F2) \;\; Max \;\; Q_H + P_H \tag{3.33}$$

Sujeito à

$$y_{it} \leq y_{it+1} \quad \forall i = 1, .., n \quad \forall t = 1, ..., H-1 \tag{3.34}$$

$$y_{it} \leq y_{jt-1} \quad \forall i = 1, .., n \quad \forall t = 1, ..., H \quad \forall j \in Pred(i) \tag{3.35}$$

$$Q_t = Q_{t-1} + P_{t-1} - \sum_{i=1}^{n} c_i(y_{it} - y_{it-1}) \quad \forall t = 1, ..., H \tag{3.36}$$

$$P_t = \sum_{i=1}^{n} p_i y_{it} \quad \forall t = 0, ..., H \tag{3.37}$$

$$y_{i0} = 0 \quad \forall i = 1, ..., n \tag{3.38}$$

$$y_{it} \in \{0, 1\} \quad \forall i = 1, ..., n \quad \forall t = 1, ..., H \tag{3.39}$$

$$Q_t, P_t \in N \quad \forall t = 0, ..., H \tag{3.40}$$

Para o caso específico em que $H = 1$ e $(i, j) = 0 \quad \forall\{(i, j) \in A\}$ (sem restrição de precedência), temos a situação onde todas as tarefas poderão ser ativadas no primeiro e único instante de tempo. Assim, o objetivo do nosso problema poderá ser reescrito da seguinte forma:

$$Max \quad Q_1 + P_1 \tag{3.41}$$

Substituindo Q_1 e P_1 pelas respectivas definições dadas por 3.36 e 3.37, temos:

$$Max \quad Q_0 + P_0 - \sum_{i=1}^{n} c_i(y_{i1} - y_{i0}) + \sum_{i=1}^{n} p_i y_{i1} \tag{3.42}$$

Pelas restrições 3.38, podemos retirar as variáveis y_{i0}, já que são iguais a zero. Chamando $p_i - c_i = b_i$, temos:

$$Max \quad Q_0 + P_0 + \sum_{i=1}^{n} b_i y_{i1} \tag{3.43}$$

Como Q_0 é um dado de entrada do problema (constante) e P_0 é igual a zero, podemos deixar o objetivo do problema da seguinte maneira, desde que lembremos do termo Q_0 que deverá ser somado à função objetivo:

$$Max \sum_{i=1}^{n} b_i y_{i1} \tag{3.44}$$

Se não há arcos relacionando as tarefas (precedências), as restrições 3.35 passam a não fazer sentido, bem como as restrições 3.34, uma vez que há apenas um instante de tempo $H = 1$ e $y_{i0} = 0$ (restrições 3.38 também podem ser desprezadas).

A única restrição que permanece é dada pela equação 3.36, que pode ser reescrita assim:

$$Q_1 = Q_0 + P_0 - \sum_{i=1}^{n} c_i y_{i1} \tag{3.45}$$

Retirando $P_0 = 0$ e trocando Q_1 e o somatório de lado, temos:

$$\sum_{i=1}^{n} c_i y_{i1} = Q_0 - Q_1 \tag{3.46}$$

Como Q_1 tornou-se uma variável livre, mas ainda inteira não-negativa, podemos transformar a equação para a seguinte forma, se chamarmos $Q = Q_0$,

$$\sum_{i=1}^{n} c_i y_{i1} \leq Q \tag{3.47}$$

Combinando esta restrição com o objetivo dado pela equação 3.44, temos o seguinte problema que é matematicamente equivalente ao problema da Mochila mencionado em [30]. Este problema, em sua versão de otimização, é considerado NP-árduo.

$$Max \sum_{i=1}^{n} b_i y_{i1} \tag{3.48}$$

$$\sum_{i=1}^{n} c_i y_{i1} \leq Q \tag{3.49}$$

$$y_{i1} \in \{0,1\} \quad \forall i = 1, ..., n \tag{3.50}$$

Obviamente, o segundo índice das variáveis y_{i1} pode ser retirado, já que se trata sempre do valor 1. No caso específico de b_i ser negativo ($p_i < c_i$, pela definição), a tarefa não trará benefício algum em ser ativada. É o equivalente a considerar itens do problema da mochila com valor negativo. Pode ser feito um pré-processamento, em tempo polinomial, que retire da instância tarefas (itens) que se encontrem nesta situação, para que a formulação fique ainda mais próxima da variante mais usual do Problema de Mochila 0-1 (com valores positivos).

De fato, a redução para o Problema da Mochila faz sentido, pois considerando apenas um instante de tempo, o objetivo de conseguir o maior lucro possível juntamente com a restrição de não consumir mais recursos do que aqueles inicialmente disponíveis se assemelha bastante aos elementos que compõe tal problema.

Se considerarmos a generalização $H \geq 1$, mas ainda sem a existência de precedências, teremos um problema semelhante ao das Múltiplas Mochilas, onde é possível colocar objetos em mais de uma mochila, ainda mantendo o objetivo de acumular o maior valor possível através dos objetos selecionados. A diferença residirá no fato de que a segunda mochila terá capacidade para comportar os objetos em função daquilo que foi escolhido para a primeira mochila - os recursos disponíveis no segundo instante de tempo do PEPRRD dependem das tarefas que foram ativadas no primeiro instante.

Ao permitirmos a existência de precedência entre as tarefas ($1 \geq (i,j) \geq 0 \ \forall \{i,j \in V | \bigcup (i,j) \ \ seja \ \ um \ \ Grafo \ Aciclico \ Direcionado\}$), não alteramos a complexidade teórica do problema, mas sim reduzimos a quantidade de tarefas que estarão disponíveis a cada instante de tempo. Evidentemente a complexidade prática do problema vai depender, entre outros parâmetros, da topologia do grafo e dos valores de custo e lucro das tarefas. Portanto, se o caso específico onde $H = 1$ e $(i,j) = 0$ é NP-árduo, o caso geral onde $H \geq 1$ e $1 \geq (i,j) \geq 0$ também o é.

Capítulo 4

Métodos heurísticos para o PEPRRD

Como o problema abordado neste trabalho é considerado difícil de ser resolvido em tempo computacional razoável, é justificada a abordagem por métodos heurísticos. Tais métodos devem ter o compromisso de apresentar soluções de boa qualidade em um curto espaço de tempo, mesmo que, para isso, tenham que sacrificar um pouco a garantia da otimalidade.

Os métodos heurísticos mais elementares consistem na geração iterativa de uma solução para o problema, geralmente apoiando-se em algum critério guloso, e na modificação de soluções já existentes, visando a melhoria da qualidade destas soluções. O problema da aplicação destas heurísticas elementares é que, frequentemente, o algoritmo planejado para o problema recai sobre uma solução cuja melhoria é impossível, embora nem sempre seja a melhor solução possível para o problema. Esta solução, chamada de ótimo local, acaba sendo a resposta encontrada pelo algoritmo heurístico.

Mecanismos para fugir deste ótimo local, ainda distante de um ótimo global, são a proposta dos chamados métodos meta-heurísticos para apresentar soluções de melhor qualidade. Tal mecanismo pode ser realizado de diversos modos e cada meta-heurística vai apresentar um método próprio para realizá-lo. Entre os métodos meta-heurísticos mais comuns temos os Algoritmos Genéticos ou Evolutivos, o GRASP (*Greedy Randomized Adaptive Search Procedure*), a Busca Tabu, o VNS (*Variable Neighborhood Search*), para citar alguns exemplos.

4.1 Algoritmos Evolutivos

Os Algoritmos Evolutivos são desdobramentos dos Algoritmos Genéticos clássicos [33], que apresentavam soluções codificadas por valores binários e operadores genéticos simples. Embora o objetivo de manter bons padrões (genes) em várias soluções simultâneas (indivíduos) seja o mesmo, os Algoritmos Evolutivos introduzem novos operadores de recombinação de soluções, buscas locais e refinamentos para fazer com que o aproveitamento dos padrões seja potencializado. Uma série de outras meta-heurísticas evolutivas também utilizam princípios semelhantes [5].

Normalmente os Algoritmos Genéticos ou Evolutivos operam sobre dezenas ou centenas de indivíduos simultaneamente, o que pode trazer dificuldades se os operadores não forem rápidos o suficiente ou permitirem a geração de soluções infactíveis que precisem de correção para que possam ser consideradas soluções viáveis para o problema.

4.1.1 Representação das soluções

Alguns Algoritmos Evolutivos para o PEPRRD já tinham sido propostos antes da realização deste trabalho, mas eles apresentavam um maneira de representar a solução chamada de Representação Direta [24, 27]. Nesta representação, cada elemento i de um vetor indicava o tempo em que a i-ésima tarefa deveria ser ativada, como na Figura 4.1. Embora esta representação seja bastante objetiva, a principal dificuldade em se trabalhar com ela é que alterações nos seus valores frequentemente resultam em solução infactíveis. Os cálculos necessários para que estas infactibilidades não existam são computacionalmente tão custosos que também não valem a pena serem executados rotineiramente.

Tempos	1	1	2	5	3	4	4	2	3	5
Tarefas	1	2	3	4	5	6	7	8	9	10

Figura 4.1: Exemplo de solução com representação direta

Uma das propostas deste trabalho é empregar uma nova forma de representação chamada de Representação Indireta, pois agora, a cada tarefa, será dada uma prioridade de ativação que deve ser administrada por um algoritmo de escalonamento diferente. As

Prioridades	1.1	0.3	2.1	5.8	4.2	7.9	6.3	4.5	2.3	4.4
Tarefas	1	2	3	4	5	6	7	8	9	10

Figura 4.2: Exemplo de solução com representação indireta

tarefas com maior prioridade deverão ser ativadas primeiro, desde que respeitem as restrições de precedência e de recursos disponíveis. A Figura 4.2 traz uma configuração de prioridades. Vale destacar que, para esta representação, quaisquer valores reais (ou mesmo inteiros) podem gerar uma solução viável, desde que o algoritmo de escalonamento respeite as restrições do problema.

Neste contexto, o novo algoritmo de escalonamento verifica, a cada instante de tempo, quais tarefas estão disponíveis e as ordena de acordo com a prioridade. Feito isto, as ativações acontecem, respeitando a ordenação, mas só se a tarefa em questão possuir custo menor do que os recursos disponíveis no momento. Após analisar todas as tarefas disponíveis e realizar as ativações possíveis, o algoritmo passa para a unidade de tempo seguinte.

4.1.2 Algoritmo Evolutivo com prioridades

A principal característica deste novo algoritmo é o uso de prioridades na representação de uma solução; ele foi chamado de EA_priority [34]. Como a representação da solução é diferente, assim também deverá ser a forma de ser operar sobre ela. Portanto, novos operadores evolutivos precisam ser definidos para que o potencial da representação seja alcançado. As subseções a seguir apresentam propostas de operadores. Cada parâmetro utilizado como o tamanho da população inicial, a chance de mutação e o número de filhos gerados foram definidos após testes preliminares com vários valores.

4.1.2.1 População inicial

Todo Algoritmo Genético ou Evolutivo precisa gerar uma população inicial para que os demais operadores possam entrar em ação. Não existe regra para a geração desta população, mas a qualidade dela pode ser crucial para que o algoritmo apresente um bom resultado final. Assim sendo, é interessante que a população inicial seja gerada por meio

de um critério guloso que faça sentido ao problema.

Tarefas que devem ser preferencialmente ativadas devem possuir: (i) elevado lucro, (ii) baixo custo ou (iii) muitas tarefas sucessoras. Certamente os dois itens mais importantes são (i) e (ii), pois o fato de uma tarefa possuir muitas sucessoras não assegura que estas serão ativadas nem que são tarefas lucrativas. A regra para se atribuir uma prioridade inicial a cada tarefa foi definida pela Equação 4.1 a seguir, onde λ é um número real no intervalo [-1;1], escolhido para cada tarefa de cada indivíduo, c_i e p_i são, respectivamente, o custo e lucro da tarefa i. O intervalo de λ foi definido de acordo com testes preliminares.

$$Prioridade_i = \frac{p_i}{c_i} + \lambda \tag{4.1}$$

Após serem gerados todos os N indivíduos, a população é ordenada pela aptidão de seus membros de forma decrescente. Testes preliminares mostraram que $N = 100$ produzia bons resultados sem consumir tempo computacional demasiadamente.

4.1.2.2 Recombinação

A Recombinação vai procurar escolher e combinar dois indivíduos já existentes de forma que sejam gerados filhos com características semelhantes a um pai ou a ambos. Para que as soluções geradas sejam interessantes ao problema, é bom que os pais não sejam idênticos ou apresentem grau de semelhança muito elevado. Por outro lado, pais com qualidade superior deveriam gerar filhos melhores. Uma proposta para atender a estes requisitos é dividir a população em classes, por exemplo: classe A - composta pelos 20% melhores indivíduos, classe C - composta pelos 20% piores indivíduos, classe B - composta pelos demais indivíduos. Um pai será sempre escolhido da classe A e outro sempre da classe B para tentar evitar que pais muito semelhantes sejam escolhidos, mas que tenham boa qualidade.

A combinação em si vai procurar gerar uma configuração de prioridades analisando os valores presentes em cada um dos pais. Um critério simples e bastante razoável é aplicar a média aritmética das prioridades dos pais. No operador de recombinação proposto, dois filhos são gerados a cada recombinação. É necessário que se estabeleça alguma forma de diferenciá-los para que não se tenha os indivíduos idênticos. A aplicação de um λ

semelhante ao utilizado na população inicial pode ajudar a resolver este percalço. Um total de N filhos é produzido a cada geração.

4.1.2.3 Seleção Natural

A Seleção Natural é um operador que tem por objetivo definir quais indivíduos poderão continuar existindo na geração seguinte e quais deverão ser removidos e eliminados. O princípio de que os indivíduos mais aptos, isto é, com maior qualidade, têm mais chance de sobrevida é bastante razoável, pois se deseja que bons padrões genéticos permaneçam mais tempo disponíveis para serem utilizados em recombinações com outros indivíduos. O critério elitista, no qual os indivíduos que possuírem os melhores valores de aptidão são privilegiados, atende bem ao princípio de prevalecimento dos mais aptos. Desta maneira, apenas os N melhores indivíduos, dentre pais e filhos, permanecerão na geração seguinte.

4.1.2.4 Mutação

O operador de mutação consiste em produzir modificações em um indivíduo já existente, para que ele ou melhore de qualidade ou transmita seus genes mutantes a outros indivíduos por meio da recombinação. A mutação deve causar perturbações de forma que os indivíduos da população não se tornem muito semelhantes. Uma forma de permitir que mutações mais intensas ocorram conforme as gerações se sucedam é atrelar o grau de mutação ao número da geração em questão. Assim, nas primeiras gerações a mutação será pequena e nas últimas gerações será bem maior. Entre muitas fórmulas testadas preliminarmente, a Eq. 4.2 apresentou bom resultado. Nela, λ é um número real no intervalo [-1;1] e o termo *geracao* indica em qual geração está sendo aplicada a mutação, a partir da população inicial que tem valor igual a um. O valor *Grau* será calculado para cada tarefa do indivíduo a ser mutacionado e será somado à prioridade dela.

$$Grau = geracao^2 * \lambda \tag{4.2}$$

Cada indivíduo terá uma chance de 5% de sofrer mutação a cada geração.

4.1.2.5 Critério de Parada

Embora não seja um operador propriamente dito, é um dos principais componentes dos Algoritmos Genéticos e Evolutivos, bem como das demais meta-heurísticas. Um bom critério de parada permite que o algoritmo tenha condições de aprimorar as soluções encontradas sem que seja consumido tempo desnecessariamente. O critério de parada inicial desta primeira versão de Algoritmo Evolutivo foi o número máximo de gerações.

4.1.3 Algoritmo Evolutivo com reconstruções

O Algoritmo Evolutivo EA_priority apresentou grandes melhorias em relação aos evolutivos anteriores [22, 27]. No entanto, o limite de gerações foi bastante restritivo em algumas instâncias de médio e grande porte. Ao se expandir este limite, recaía-se em outro problema que era a estagnação da população de indivíduos. Uma população é considerada estagnada, quando, por exemplo, não consegue gerar indivíduos melhores após algumas gerações consecutivas. A estagnação é um problema comum aos Algoritmos Genéticos e Evolutivos, embora bastante indesejável (principalmente se ocorrer nas primeiras gerações). Como a substituição dos operadores evolutivos por outros métodos não se mostrou eficiente em testes preliminares, uma saída foi incorporar novos mecanismos que permitissem aos operadores já existentes uma prorrogação de sua capacidade de gerar indivíduos que não causem a estagnação da população. As subseções seguintes explicam estes mecanismos, partindo do princípio que a nova versão do algoritmo evolutivo tem por base o algoritmo EA_priority.

4.1.3.1 População inicial

Como dito anteriormente, três características das tarefas são preponderantes para que elas tenham prioridades maiores. O algoritmo EA_priority não levou em consideração o número de tarefas sucessoras no cálculo da prioridade inicial de cada tarefa. Em muitas das instâncias testadas não houve diferença significativa em relação ao uso ou não deste parâmetro adicional. Em apenas 4 das 20 instâncias mais testadas houve uma melhoria de menos de 5%. Como o cálculo do número de sucessoras para cada tarefa não é computacionalmente demorado, a nova versão do algoritmo evolutivo incorporou este componente

no cálculo das prioridades iniciais adaptando a fórmula da Eq. 4.1 para a Eq. 4.3, a seguir, onde $Suc(i)$ é o conjunto de tarefas imediatamente sucessoras da tarefa i, λ é um número real entre -1 e 1, c_i e p_i são, respectivamente, o custo e o lucro da tarefa i:

$$Prioridade_i = \frac{p_i^2(|Suc(i)| + 1)}{c_i} + \lambda \tag{4.3}$$

Como a relação entre o custo e o lucro de uma tarefa pode ter o mesmo valor para diversas combinações (por exemplo: 2/5, 4/10, 6/15), ao elevar o lucro p_i ao quadrado, espera-se privilegiar as tarefas que, tendo a mesma relação custo-benefício, produzam mais recursos no decorrer do escalonamento.

4.1.3.2 Refinamento de Soluções

Uma desvantagem da representação por prioridades é que o escalonamento é realizado levando em conta apenas o valor relativo a cada tarefa e a possibilidade ou não dela ser ativada num determinado instante de tempo, já que as restrições de precedência e de recursos devem ser respeitadas. É muito comum que o algoritmo de escalonamento permita a ativação de um tarefa que consuma mais recursos do que produza. Isto ocorre porque a tarefa pode ter recebido uma prioridade mais alta do que deveria ou porque todas as tarefas com maior prioridade já tenham sido ativadas e ainda haja recursos para proceder com outras ativações. Em ambos os casos, só é possível perceber o equívoco após o escalonamento ser completamente realizado, já que tentar prever situações como esta pode ser muito impreciso e até muito custoso computacionalmente.

Uma vez realizado o escalonamento, é fácil analisar se cada uma das tarefas contribuiu de forma positiva ou não para o resultado final do processo. Começando pelas tarefas ativadas no último instante de tempo e regredindo até o primeiro instante, analisamos cada uma das tarefas ativadas e suas sucessoras que também foram ativadas. Se este grupo todo estiver consumindo mais recursos do que produzindo, todo ele é removido do escalonamento; caso contrário, mantido. Toda vez, contudo, que uma única tarefa ou um grupo forem removidos é preciso voltar ao último instante de tempo e recomeçar o processo, porque a retirada de uma tarefa pode fazer com que outra já não valha mais a pena permanecer ativada. Este Refinamento é realizado a cada novo indivíduo gerado.

4.1.3.3 Busca Local

Outra forma de tentar melhorar a qualidade de uma solução é aplicar sobre ela algum método de busca local. Uma busca local é um método que recebe uma solução para o problema e analisa soluções chamadas de vizinhas, isto é, que apresentam algum tipo de semelhança em relação à solução original. Este conjunto de soluções é chamado de vizinhança.

Algoritmo Evolutivo EA_ages trabalha com uma relação de vizinhança baseada na troca de duas prioridades. Em outras palavras, uma solução S' é considerada vizinha de S se e somente se $n-2$ tarefas tiverem as mesmas prioridades e as outras duas tarefas tiverem prioridades trocadas, onde n é o número total de tarefas de S. A busca local chamada LS_swap tem seu pseudo-código mostrado pelo Algoritmo 4.

Algoritmo 4 LS_swap

1: **for** $i = 1$ to $n-1$ **do**
2: **for** $j = i+1$ to n **do**
3: **if** $(menor(i) = menor(j))$ e $(tempo(i) \neq tempo(j))$ **then**
4: $S' \leftarrow S$;
5: Troca a prioridade de i pela de j em S';
6: **if** $aptidao(S') > aptidao(S)$ **then**
7: $S \leftarrow S'$;
8: **end if**
9: **end if**
10: **end for**
11: **end for**
12: **return** S;

É interessante notar que, visando um gasto menor de tempo, alguns critérios foram adotados para que nem todos os pares de tarefas fossem testados. São testadas apenas as tarefas que possuírem o mesmo menor de tempo de ativação, mas que tiverem sido ativadas em tempos distintos (linha 3). Estas tarefas são particularmente interessantes pois, em princípio, poderiam ter sido ativadas no mesmo instante mas não o foram. Em teoria, a troca das prioridades destas tarefas permite que novas escolhas, que não foram contempladas pelo escalonamento original sejam feitas, durante o re-escalonamento. Também é importante notar que a troca somente será mantida se o escalonamento resultante for melhor do que o original (linha 7).

Mesmo adotando o critério que restringe o número de trocas, se a busca local for apli-

cada muito frequentemente pode deixar o algoritmo pouco eficiente em relação ao tempo computacional. Logo, ela só será aplicada em duas situações: (i) quando uma nova melhor solução for encontrada, para que ela seja explorada ao máximo; (ii) quando a população não conseguir produzir melhores resultados por 30 gerações seguidas, nesta circunstância, a população é considerada estagnada e a busca local será aplicada aos melhores indivíduos, tentando fazer com que estes melhorem pelo menos um pouco suas aptidões. Testes preliminares mostraram que a aplicação apenas na classe A (20% melhores indivíduos) apresenta, em geral, uma boa relação de tempo consumido e melhorias obtidas.

4.1.3.4 Critério de Parada

Um dos principais componentes que podem fazer o algoritmo estagnar ou perder muito tempo sem conseguir evoluir as soluções é o critério de parada. O que se pôde perceber nos testes realizados é que o critério de número fixo de gerações funciona bem nas instâncias onde rapidamente se produz uma solução próxima ao ótimo global ou mesmo ele próprio. Nas instâncias onde o algoritmo vai gradativamente melhorando a solução, este limite fixo acaba por encerrar o algoritmo sem que ele consiga esgotar toda sua capacidade de melhoria das soluções correntes.

A idéia de mudar o critério de parada para um limite de gerações mais flexível foi uma das modificações implementadas nesta segunda versão. Inicialmente, o limite de gerações é o mesmo daquele usado na versão anterior, mas conforme o algoritmo consegue gerar soluções melhoradas, o limite é estendido por meio de gerações extras que são adicionadas a ele.

Porém, é interessante notar que as melhorias que ocorrem ao longo da execução do algoritmo não possuem o mesmo contexto. As primeiras melhorias ocorrem, geralmente, no começo da execução do algoritmo e atuam sobre soluções de qualidade inferior se comparadas àquelas que podem ocorrer nos instantes finais da execução, quando a qualidade das soluções tende a ser bem mais elevada e, portanto, mais difícil de ser aprimorada. Como consequência direta disto, a frequência de melhorias no final do algoritmo é bem menor.

Se um algoritmo consegue, após muito tempo, aprimorar sua melhor solução (solução incumbente), é bastante razoável que ele possa ter bem mais tempo para que consiga

outra(s) melhoria(s). Logo, a extensão do limite de gerações deve ser maior neste caso do que a extensão obtida pelas primeiras melhorias. Uma fórmula simples que permite que este raciocínio seja posto em prática é apresentada pela Eq. 4.4, onde *geracao* é número indicativo de em qual geração houve a referida melhoria. Os valores 4 e 20 foram fixados após testes preliminares.

$$\beta = \left\lceil \frac{geracao}{20} \right\rceil + 4 \tag{4.4}$$

Quanto maior o valor de *geracao*, maior o valor de β, que representa a quantidade de gerações extras que serão incorporadas ao limite estabelecido.

4.1.3.5 Reconstrução da População

O único caso onde o critério de parada não é bem explorado ocorre quando o algoritmo evolutivo encontra, nas primeiras gerações, soluções cada vez melhores, mas não consegue se aproximar do ótimo global. O número de gerações extras aumenta bastante mas a população ainda assim permanece estagnada, mesmo sendo aplicada a busca LS_swap algumas vezes. Nesta situação, não há indícios de que manter a execução normal do algoritmo produzirá resultados melhores, mesmo se houver um grande número de gerações extras, ou seja, se ainda houver muitas gerações a serem computadas.

A proposta feita em [35] foi uma mudança mais radical na população estagnada: eliminar todos os indivíduos que se mostram incapazes de prover continuadas melhorias e reconstruir a população a partir do melhor indivíduo obtido até o momento. O objetivo é produzir uma população mais semelhante a este indivíduo como forma de intensificar as buscas na vizinhança do que se conhece de melhor. A criação de um indivíduo parecido se dará com a adição de um valor λ a cada prioridade do melhor indivíduo. Assim, teremos uma nova população de indivíduos distintos, porém com qualidade próxima ao melhor conjunto de genes já produzido. Este λ deve ter uma amplitude um pouco maior do que a usada nos operadores anteriores, caso contrário, os indivíduos serão tão semelhantes que nenhum benefício seria obtido. Um valor real no intervalo [-10;10] mostrou-se bastante satisfatório.

Esta nova população será considerada uma população inicial e o algoritmo procederá

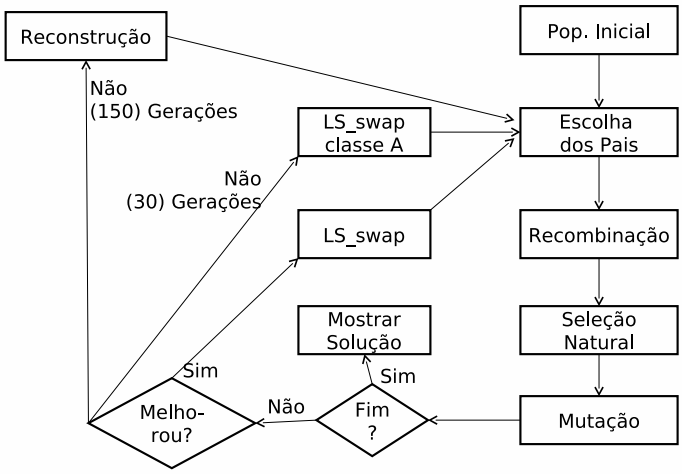


Figura 4.3: Esquema estrutural do algoritmo EA_ages

Parâmetro	EA3	EA_priority	EA_ages
Repres. da Solução	Direta	Indireta	Indireta
Tam. da População	20	100	100
Prob. de Mutação	0.20	0.05	0.05
Estagnação	4 Gerações(Intensif.)	-	30 Gerações (LS_swap na Classe A)
	4 Gerações(Diversif.)	-	150 Gerações (Reconstrução)
λ	-	[-1;+1] (Recombinação)	[-1;+1] (Recomb. e Mutação)
	-	-	[-10;+10] (Reconstrução)
Crit. de Parada	50 Gerações	50 Gerações	50 Gerações(com bônus)

Tabela 4.1: Principais parâmetros dos Algoritmos Evolutivos estudados e seus valores

como se tivesse reiniciado sua execução. A exceção é que o número de gerações restantes será preservado, ou seja, não voltará a valer o mesmo número de gerações como na primeira população. O conjunto de gerações entre uma reconstrução e outra será chamada de Era (*Age*, em inglês), dando nome ao algoritmo evolutivo de EA_ages. As Eras terminarão quando forem computadas 150 gerações consecutivas sem mudanças na melhor solução encontrada. A Figura 4.3 traz um esquema de como os operadores deste algoritmo são executados. Já a Tabela 4.1 mostra os principais parâmetros e os valores utilizados para as três versões de Algoritmos Evolutivos analisadas neste trabalho (EA3, EA_priority e EA_ages).

4.1.4 Versão Paralela do EA_ages

Também em [35] foi proposta uma versão paralela para o algoritmo EA_ages. Diferentemente do paralelismo mais usual, com processos sendo executados em máquinas distintas e trocas de mensagens via infra-estrutura de rede [36, 37], a proposta apresentada foi a

utilização de processadores com múltiplos núcleos e memória compartilhada. Esta arquitetura vem se tornando cada vez mais popular com a diminuição dos custos envolvidos na compra deste tipo de equipamento. Praticamente todos os processadores vendidos atualmente têm capacidade de executar simultaneamente dois ou mais processos sem que haja a necessidade de troca de mensagens via rede.

O objetivo do emprego desta arquitetura foi o de reduzir o tempo computacional gasto para se gerar uma solução de mesma qualidade. O ambiente de testes, descrito com mais detalhes no Capítulo 6, tem capacidade para executar até 4 processos simultaneamente, o que permite que a versão paralela do algoritmo EA_ages tenha sua carga de trabalho dividida até este grau, sem perda de desempenho. Vale lembrar que a versão sequencial do EA_ages descrita na seção anterior utiliza apenas um núcleo de processamento, o equivalente a apenas 25% da capacidade total da máquina - outro motivo para o emprego de versões paralelas.

A paralelização foi implementada por meio de alguns dos operadores evolutivos utilizados. A seleção dos pais, a recombinação, a mutação, a aplicação da busca LS_swap quando a população estagna e a reconstrução da população tiveram suas cargas de trabalho divididas por 2 ou 4 núcleos, nas versões com 2 ou 4 *threads* respectivamente. A ordenação da população, a aplicação da LS_swap quando um melhor indivíduo é encontrado e a seleção natural permanecem sequenciais. No caso dos operadores paralelizados, metade ou quarto da população é entregue a cada núcleo para que possam ser aplicados os devidos operadores. No caso da recombinação, metade ou um quarto dos filhos é gerado por cada núcleo.

4.2 GRASP - *Greedy Randomized Adaptive Search Procedure*

O GRASP é uma das meta-heurísticas mais utilizadas em Otimização Combinatória. É aplicada a problemas dos mais diversos tipos: configurações de redes de computadores, sequenciamento de produção, problemas de atribuição, entre outros. Foi proposta em [23] e possui uma estrutura muito simples, composta de uma heurística construtiva randomizada e uma busca local executadas repetidas vezes. A adição de mecanismos como

recombinação de caminhos [26, 38] e utilização de algum tipo de memória de soluções [39, 40] têm permitido que a estrutura do GRASP seja expandida e consiga resultados melhores.

Para que haja uma boa comparação do comportamento das diversas versões de metaheurísticas aplicadas ao PEPRRD, os métodos utilizados nas versões GRASP aqui propostas são idênticos aos das versões de Algoritmos Evolutivos. Embora seja mais comum encontrar versões do GRASP que tenham métodos construtivos capazes de gerar soluções sem a necessidade de dados adicionais, o método construtivo proposto neste trabalho será semelhante à reconstrução realizada pelo EA_ages. Partindo de uma configuração de prioridades, novas soluções serão geradas através da adição de um valor real λ no intervalo [-10;10]. O Algoritmo 5 traz um pseudo-código para esta versão do GRASP.

Algoritmo 5 GRASP1

1: $S^* \leftarrow prioridadesIniciais()$;
2: $max \leftarrow valor(S^*)$;
3: **while** $tempoGasto < tempoLimite$ **do**
4: **for** $i = 1$ to n **do**
5: $\lambda \leftarrow aleatorio(-10, +10)$;
6: $S_i \leftarrow S_i^* + \lambda$;
7: **end for**
8: $S \leftarrow refinamento(S)$;
9: $S \leftarrow LS_swap(S)$;
10: **if** $valor(S) > max$ **then**
11: $max \leftarrow valor(S)$;
12: $S^* \leftarrow S$;
13: **end if**
14: **end while**
15: **return** S^*;

A primeira configuração será produzida da mesma forma que um indivíduo da população inicial (linha 1), pela Eq. 4.3. A partir de então começa a parte iterativa do GRASP1 - a solução será adicionada de valores λ (linhas 4-6), passará pelo refinamento utilizado no EA_ages (linha 7), pela busca local LS_swap (linha 8) e, se apresentar um escalonamento de maior qualidade do que o melhor encontrado até o momento, suas prioridades servirão de base para as próximas iterações (linhas 9-12).

Uma das características que chama a atenção ao analisar a estrutura desta versão GRASP é que uma boa solução não é completamente descartada como na maioria das versões básicas do GRASP, ou seja, existe uma memória que permanece em constante

atualização enquanto as soluções geradas forem melhorando de qualidade. Como a estrutura e a quantidade de métodos é relativamente simples e pequena, fica muito difícil estabelecer outro critério de parada que não o de tempo limite, já que de uma instância para outra a quantidade de iterações pode ter diferença de algumas ordens de grandeza. Além disso, o critério de tempo limite pode ser facilmente deslocado para qualquer outra meta-heurística, permitindo comparações mais justas.

4.2.1 GRASP com recombinação de soluções

Uma segunda versão da meta-heurística GRASP, chamada GRASP2, é proposta neste trabalho. Esta segunda versão apresenta todos os componentes da versão anterior, porém é mantido ao longo da execução do algoritmo um conjunto contendo as 10 melhores soluções distintas geradas até o momento, onde 10 é parâmetro de entrada. Este *pool* de soluções é atualizado ao final cada iteração do GRASP2. De tempos em tempos, o fluxo de execução normal do GRASP é interrompido e uma etapa de intensificação é realizada no *pool* onde todas as soluções armazenadas são recombinadas entre si, gerando um conjunto extra de soluções. Esta etapa é feita através do mesmo método de recombinação adotado nos Algoritmos Evolutivos EA_priority e EA_ages. Quando todas as recombinações encerrarem, o *pool* é novamente atualizado, contemplando também as soluções recombinadas. As soluções que não permanecerem no *pool* são descartadas bem como as que porventura tiverem saído de lá; o algoritmo então prossegue, verificando se uma nova melhor solução foi encontrada e passando para a iteração seguinte. Um exemplo de pseudo-código do GRASP2 é delineado pelo Algoritmo 6. Foi utilizado como parâmetro para a frequência de realização da intensificação um valor de 10 iterações consecutivas, independentemente de haver atualização na melhor solução ou não.

4.3 ILS - *Iterated Local Search*

A meta-heurística ILS (*Iterated Local Search*, [41, 42, 43]) possui uma estrutura bastante semelhante à da Busca Tabu. Uma solução S é inicialmente gerada, a cada iteração, ela sofre algum tipo de perturbação e passa por uma busca local e/ou refinamento, transformando-se em uma solução S'. Se esta solução for a melhor já encontrada, ela

Algoritmo 6 GRASP2

1: $S^* \leftarrow prioridadesIniciais()$;
2: $max \leftarrow valor(S^*)$;
3: $iteracao \leftarrow 0$;
4: $inicia(pool)$;
5: **while** $tempoGasto < tempoLimite$ **do**
6: $iteracao \leftarrow iteracao + 1$
7: **for** $i = 1$ to n **do**
8: $\lambda \leftarrow aleatorio(-10, +10)$;
9: $S_i \leftarrow S_i^* + \lambda$;
10: **end for**
11: $S \leftarrow refinamento(S)$;
12: $S \leftarrow LS_swap(S)$;
13: **if** $valor(S) > max$ **then**
14: $max \leftarrow valor(S)$;
15: $S^* \leftarrow S$;
16: **end if**
17: Tenta inserir S em *pool*;
18: **if** iteracao **mod** $10 = 0$ **then**
19: Recombina soluções no pool;
20: $S' \leftarrow melhorSolucao(pool)$;
21: **if** $valor(S') > max$ **then**
22: $max \leftarrow valor(S')$;
23: $S^* \leftarrow S$;
24: **end if**
25: **end if**
26: **end while**
27: **return** S^*;

passa a ser a solução incumbente para as iterações seguintes.

A perturbação à qual a solução S será submetida pode, a princípio, ser qualquer tipo de modificação nesta solução. No entanto, a representação indireta traz uma complicação adicional: a alteração de pequenas porções da solução dificilmente causa alguma alteração significativa no escalonamento resultante. Em contrapartida, alterar um grande conjunto de prioridades frequentemente causa uma degradação na qualidade da solução corrente. Definir um meio termo é uma tarefa difícil, já que, de instância para instância, o grau de perturbação pode variar substancialmente.

Uma proposta para dirimir este impasse é aplicar vários tipos de perturbações sobre a mesma solução corrente S e analisar qual solução resultante S' apresenta melhor qualidade.

4.3.1 Perturbações

A primeira perturbação (Pert1) consiste em dobrar o valor da prioridade das predecessoras diretas de, no máximo, 10% da tarefas da instância em questão. As tarefas são escolhidas aleatoriamente e podem ser escolhidas repetidas vezes. Após a escolha de uma tarefa, todas as suas predecessoras têm a prioridade dobrada e ocorre então um re-escalonamento levando em conta os novos valores. A segunda perturbação (Pert2) é executada de forma bem similar à primeira, porém dividindo por 2 a prioridade das predecessoras da tarefa escolhida.

A terceira perturbação (Pert3) cria uma lista L contendo também 10% das tarefas do problema; neste caso, as tarefas são todas distintas. Após esta etapa, a prioridade da primeira tarefa de L é trocada com a última tarefa da lista; a segunda com a penúltima e assim por diante. Terminadas as trocas, ocorre o re-escalonamento com as novas prioridades.

Por fim, a última perturbação (Pert4) procura dar mais prioridade a tarefas que se encontrem nas seguintes situações: (i) a tarefa foi ativada, no escalonamento original, no menor tempo possível ou (ii) a tarefa não está ativada no escalonamento original. Estas duas situações são extremas porque se uma tarefa é ativada no menor tempo possível, significa que ela deve contribuir muito positivamente para o resultado do escalonamento. Se ela não foi ativada, pode significar que ela tem um impacto negativo na solução do problema ou foi preterida por outra tarefa considerada mais interessante durante o escalonamento.

Apostando nesta última situação, pode ser interessante verificar se a presença dela é mesmo prejudicial ou não. Ao aumentar sua prioridade, se a solução resultante for melhor do que a original, pode ser um indício de que o escalonamento equivocou-se ao evitá-la; se a solução resultante for pior, ela será desprezada pelo próprio ILS e não comprometerá o desempenho do algoritmo. Em ambos os casos, a prioridade das tarefas que se enquadrarem nas duas condições mencionadas será multiplicada por 5. De fato, não haveria necessidade de se multiplicar a prioridade da tarefa que foi ativada no menor tempo possível, mas isso será feito para evitar que futuramente ela seja escolhida por outra perturbação e acabe ficando com um valor de prioridade muito abaixo do original.

Antes de cada sequência de perturbações a solução corrente sofre uma variação de suas prioridades através da soma de um λ de forma análoga às versões de GRASP propostas. Esta variação foi adicionada para tentar evitar que as perturbações trabalhem sobre a mesma solução por muitas iterações, caso nenhuma delas consiga gerar uma solução de qualidade melhor. Um pseudo-código para esta primeira versão do ILS é apresentado a seguir. A solução inicial do ILS é computada da mesma forma que a solução original do GRASP (linha 1). A função *melhorSolucao* escolhe dentro das quatro soluções passadas qual apresenta melhor resultado (linha 7).

Algoritmo 7 ILS1

1: $S^* \leftarrow prioridadesIniciais();$
2: **while** $tempoGasto < tempoLimite$ **do**
3: **for** $i = 1$ to n **do**
4: $\lambda \leftarrow aleatorio(-10, +10);$
5: $S'_i \leftarrow S^*_i + \lambda;$
6: **end for**
7: $S^1 \leftarrow Pert1(S');$
8: $S^2 \leftarrow Pert2(S');$
9: $S^3 \leftarrow Pert3(S');$
10: $S^4 \leftarrow Pert4(S');$
11: $S \leftarrow melhorSolucao(S^1, S^2, S^3, S^4);$
12: **if** $valor(S) > valor(S^*)$ **then**
13: $S^* \leftarrow S;$
14: **end if**
15: **end while**
16: **return** $S^*;$

4.3.2 ILS com recombinação de soluções

De forma bem semelhante ao que foi testado no GRASP, propomos também uma recombinação das soluções provenientes das perturbações, como forma de verificar se o operador de recombinação é capaz de aprimorar a qualidade das soluções produzidas também em outras meta-heurísticas que não os Algoritmos Evolutivos. Cada solução S^k gerada por uma perturbação é recombinada com outra e, ao final desta etapa, cada uma das soluções perturbadas ou recombinadas é comparada com a melhor solução já encontrada. O Algoritmo 8 traz um pseudo-código para esta versão do ILS.

Algoritmo 8 ILS2

1: $S^* \leftarrow prioridadesIniciais();$
2: **while** $tempoGasto < tempoLimite$ **do**
3: **for** $i = 1$ to n **do**
4: $\lambda \leftarrow aleatorio(-10, +10);$
5: $S'_i \leftarrow S^*_i + \lambda;$
6: **end for**
7: $S^1 \leftarrow Pert1(S');$
8: $S^2 \leftarrow Pert2(S');$
9: $S^3 \leftarrow Pert3(S');$
10: $S^4 \leftarrow Pert4(S');$
11: $pool \leftarrow recombinaTodas(S^1, S^2, S^3, S^4);$
12: $pool \leftarrow pool \cup \{S^1, S^2, S^3, S^4\};$
13: $S \leftarrow melhorSolucao(pool);$
14: **if** $valor(S) > valor(S^*)$ **then**
15: $S^* \leftarrow S;$
16: **end if**
17: **end while**
18: **return** $S^*;$

Capítulo 5

Algoritmos Híbridos para o PEPRRD

O termo híbrido serve para designar aquilo que é proveniente da combinação de duas ou mais coisas distintas. Assim, é possível encontrar na literatura algoritmos híbridos de meta-heurísticas distintas [44, 45, 46]. Também é possível atribuir este termo a combinações de algoritmos que representam paradigmas diferentes como os métodos exatos e os heurísticos. É necessário, no entanto, fazer a ressalva de que, neste caso específico, o algoritmo resultante não poderá ser de natureza díspar a ambos os originais simultaneamente. Em outras palavras, ou o algoritmo terá um caráter final heurístico ou terá um caráter exato, uma vez que estes predicados são mutuamente excludentes.

Esta tese apresenta alguns algoritmos híbridos para o PEPRRD. Na análise específica de cada um, será explicitado se o algoritmo poderá ser considerado exato ou heurístico.

5.1 EA_ages + CPLEX

O primeiro dos algoritmos híbridos apresenta uma combinação entre a heurística EA_ages e a formulação matemática F2, proposta no Capítulo 3 e executada através do *software* CPLEX. O objetivo deste algoritmo é utilizar o ponto forte de uma das abordagens para suprir a deficiência da outra.

No caso da execução via CPLEX, um dos grandes problemas é gerar a primeira solução factível como foi estudado em [47, 48]. Entretanto, a geração desta solução deve ser uma tarefa fácil para os métodos considerados heurísticos. No esquema proposto, após a geração da população inicial pelo algoritmo evolutivo, é feita uma chamada para que o CPLEX inicie sua execução. Além disso, é informada a melhor solução da população inicial

para que o CPLEX tenha, desde o seu início, um limite primal já estabelecido. Por fim, uma listagem contendo uma ordem prioritária para a fixação das variáveis presentes na formulação é criada e também entregue ao CPLEX. A confecção desta lista será explicada posteriormente, mas não se deve confundir a prioridade das tarefas que são trabalhadas pelo algoritmo evolutivo com esta nova lista de prioridades para a fixação das variáveis.

Por sua parte, a maior dificuldade encontrada pelo algoritmo evolutivo, ao criar o escalonamento através da lista de prioridades que representa cada indivíduo, é saber qual é o melhor momento para que se encerrem as ativações das tarefas em um determinado instante de tempo e se recomecem no instante seguinte. Isto ocorre porque o algoritmo de escalonamento tenta ativar a maior quantidade possível de tarefas o quanto antes. Em algumas situações, contudo, é preferível que não sejam gastos todos os recursos disponíveis em um instante, visando a ativação de tarefas mais lucrativas em um instante futuro, como mostrado na Figura 5.1.

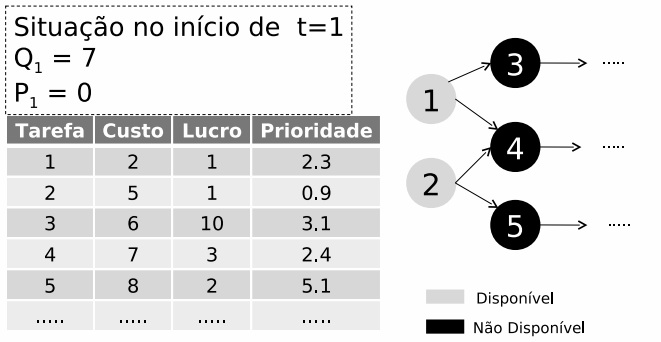


Figura 5.1: Exemplo de situação onde a ativação irrestrita é prejudicial

Se o algoritmo decidir ativar de início além da tarefa 1 também a 2, o instante de tempo $t = 2$ contará com apenas 2 unidades de recurso disponível provenientes dos lucros das tarefas ativadas. Neste caso, não será possível realizar a ativação da tarefa 3, bastante lucrativa, por pelo menos duas unidades de tempo, causando um perda de recursos que podem fazer muita diferença no resultado final do escalonamento. Se somente a tarefa 1 fosse ativada em $t = 1$, já no instante seguinte seria possível ativar a tarefa 3.

Uma tentativa de evitar este problema foi feita criando-se uma limitação para a quantidade de ativações que ocorrerão a cada instante de tempo. Um vetor de H números inteiros não-negativos determina qual a quantidade máxima de ativações podem ser realizadas em cada instante de tempo do horizonte de planejamento, independentemente

de quais tarefas forem ativadas. Desta forma, é possível resolver a situação exposta na Figura 5.1 e, de quebra, promover o refinamento tratado na subseção 4.1.3.2, já que será possível prevenir ativações desnecessárias se as prioridades forem bem computadas.

Este vetor de números inteiros, chamado de perfil de ativações (ou simplesmente perfil), não é simples de ser calculado heuristicamente, pois um mesmo perfil pode resultar em escalonamentos distintos de acordo com a lista de prioridades que for utilizada. A definição de um perfil de boa qualidade, ou seja, que permita a geração de uma solução de boa qualidade, pode ser um elemento fundamental para que o algoritmo evolutivo possa produzir resultados bem melhores.

O esquema híbrido proposto prevê que ao encontrar uma nova solução incumbente o CPLEX deve repassar ao algoritmo evolutivo, que já está rodando paralelamente, o perfil daquela solução para que possa ser aplicado a todas as soluções posteriormente geradas pelo evolutivo. Obviamente, aquela solução também é entregue ao evolutivo, codificada e inserida na população corrente na forma de um indivíduo extra. Espera-se que este indivíduo possa transferir parte de sua carga genética a seus sucessores e que estes tenham uma aptidão melhorada. A codificação é algo bastante abstrato pois, para o CPLEX, não existe a questão da prioridade entre as tarefas, mas apenas a solução final. Alguns mecanismos foram testados e um dos que apresentaram resultado bem satisfatório foi o de multiplicar por uma constante positiva K a diferença $H - S_i$, onde S_i é o tempo de ativação da tarefa i na solução encontrada pelo CPLEX. Se uma tarefa não estiver ativada nesta solução, seu valor S_i será considerado igual a $H + 1$, o que resultará em uma prioridade negativa, ou seja, muito baixa.

Em resumo, inicialmente o evolutivo fornecerá ao CPLEX uma solução que servirá de limite primal e uma ordem de prioridade para a fixação das variáveis. Quando o CPLEX encontrar uma nova solução incumbente, fornecerá ao evolutivo um perfil e a solução incumbente que será utilizada como um indivíduo extra. Em resposta, sempre que o evolutivo gerar uma solução de qualidade ainda superior também a entregará ao CPLEX como novo limite primal. É importante notar que um limite primal melhor tem dois efeitos positivos para o CPLEX: (i) subproblemas que tiverem um valor da relaxação linear pior do que o da solução passada serão automaticamente eliminados; (ii) permite-se que o CPLEX utilize suas técnicas de polimento e inferência de cortes sobre uma solução

mais próxima do valor ótimo. Portanto, é provável que sejam obtidos resultados melhores por meio destas técnicas.

O critério de parada deste algoritmo híbrido será o término da otimização realizada pelo CPLEX. Logo, este algoritmo pode ser considerado como um método exato, já que, se for dado a ele tempo suficiente, sempre fornecerá uma ou mais soluções ótimas. Infelizmente na prática, não é viável permitir que ele execute até que prove a otimalidade da solução nas instâncias de maior porte - neste(s) caso(s) será dado um tempo limite para a execução do método híbrido.

5.1.1 Prioridade para a fixação das variáveis

O PEPRRD apresenta uma característica muito interessante. Tanto as variáveis binárias quanto as inteiras das formulações F1 e F2 possuem uma relação entre si dada pelo índice que identifica a unidade de tempo à qual estão relacionadas. Em consequência disso, ao se fixar o valor das variáveis relacionadas aos primeiros instantes de tempo, é possível reduzir o intervalo de valores que podem ser atribuídos a variáveis de tempos posteriores. Como as variáveis inteiras são computadas em função das variáveis binárias e como a mencionada redução é mais significativa nestas mesmas variáveis binárias (principalmente na formulação F2), a ordem de fixação das variáveis binárias levará em consideração apenas a unidade de tempo à qual elas estão relacionadas. Quanto mais cedo for o tempo, maior a sua prioridade.

Para o cálculo da prioridade, será utilizada uma constante inteira e positiva W, de acordo com a equação a seguir, onde H é o tamanho do horizonte de planejamento do problema.

$$Prioridade(x_{it}) = (H - t + 1) * W \quad \forall i = 1, ..., n \quad \forall t = 1, ..., H \tag{5.1}$$

Alterações na ordem de grandeza de W bem como a utilização do termo i no lado direito da equação foram testadas, mas não apresentaram diferença significativa no tempo de execução das formulações.

5.2 CPLEX como busca local

Uma forma muito interessante de utilizar o otimizador CPLEX é realizar uma pré-fixação do valor de variáveis que estejam relacionadas a uma solução gerada heuristicamente. O algoritmo *Local Branching* [49, 50] é um exemplo desta técnica.

Para o PEPRRD, foi desenvolvido um algoritmo que utiliza novamente a heurística EA_ages, porém em sua versão paralela. Toda vez que esta heurística encontrar uma nova melhor solução S, será criada uma formulação que levará em conta o tempo de ativação de cada tarefa em S para que o CPLEX possa examinar a vizinhança desta solução. Aqui, uma solução S' será considerada vizinha de S se $S_i - Z \leq S'_i \leq S_i + Z \;\; \forall i = 1, .., n$, onde Z é outra constante inteira positiva e S_i (S'_i) é o tempo de ativação da tarefa i na solução S (S').

Se $Z = 1$, um problema surge quando uma tarefa i é ativada no último ou no primeiro instante de tempo, pois a estrutura de vizinhança levará em consideração o instante de tempo $t = H+1$ ou $t = 0$, respectivamente. Ambos os instantes estão fora do horizonte de planejamento e, portanto, não faz sentido fixar um valor para estas variáveis. A melhor maneira de resolver isto é simplesmente ignorar tais variáveis. Se $Z > 1$ o problema também ocorre, mas, felizmente, a solução de ignorar as variáveis que estiverem fora de escopo é a mesma. As Equações (5.2) e (5.3) devem ser incluídas na formulação para que a busca local possa ocorrer. Estas restrições asseguram que a tarefa i não poderá ser ativada antes de $S_i - Z$, mas deverá estar ativada depois de $S_i + Z$.

$$y_{it} = 0 \quad \forall i = 1, ..., n \quad \forall t = 1, ..., S_i - Z - 1 \tag{5.2}$$

$$y_{it} = 1 \quad \forall i = 1, ..., n \quad \forall t = S_i + Z, ..., H \tag{5.3}$$

Quando o CPLEX realizar a busca local, provavelmente encontrará soluções piores e melhores que S. Para acelerar o execução da busca local, será informado ao CPLEX o valor da solução S que servirá de limite primal para o algoritmo, assim, nenhuma solução de qualidade inferior será considerada. Infelizmente, mesmo com as várias restrições que forem adicionadas, a busca local pode demorar muito tempo, principalmente em instâncias com muitas tarefas. Como o objetivo da busca local não é encontrar a solução ótima do

problema como um todo, mas apenas na vizinhança de uma solução, será estabelecido um limite de tempo para que a busca seja executada.

Ao final dela, a melhor solução encontrada será devolvida ao EA_ages que a utilizará como um indivíduo de sua população. Se outro melhor indivíduo for gerado, o CPLEX será novamente acionado para tentar aprimorar esta nova solução. O operador que realiza a busca local LS_swap nos indivíduos da classe A será substituído pela aplicação desta busca local, chamada LS_cplex, em um indivíduo escolhido aleatoriamente dentro desta subpopulação. Isto será feito para que a LS_cplex não seja aplicada sempre sobre o mesmo indivíduo, nem que seja aplicada tantas vezes que torne o algoritmo muito custoso computacionalmente.

Diferentemente da versão anterior, esta versão híbrida não pode ser considerada exata, pois não existe garantia de que a solução ótima seja encontrada. Embora a busca local possa encontrar o melhor vizinho de uma solução qualquer, não é possível afirmar que se trate de uma solução ótima. Se $Z \geq H$, isto seria possível, mas seria também equivalente a realizar a otimização do problema todo, independente da solução S. Somente neste caso, o algoritmo poderia ser considerado exato.

5.3 Escalonamentos Parciais

O algoritmo híbrido CPLEX+EA3 proposto em [27] introduziu uma idéia bem interessante: dividir o horizonte de planejamento em duas metades e realizar o escalonamento destas partes em sequência, pelo método exato CPLEX e pelo evolutivo EA3, respectivamente. O algoritmo apresentou resultados significativos em instâncias de médio porte ou que possuam um curto horizonte de planejamento.

Para instâncias maiores, principalmente as detentoras de um extenso horizonte de planejamento, o algoritmo consumia muito tempo para produzir o escalonamento parcial da primeira metade (sob responsabilidade do CPLEX) e, consequentemente, não conseguia terminar em tempo aceitável. O formulação utilizada então era ainda a F1.

A proposta agora é, além de utilizar a formulação F2 com as restrições adicionais, dividir o horizonte em partes menores para que um resultado parcial seja produzido mais rapidamente. Esta solução é, então, utilizada como base para que outra solução parcial

(que contemple mais unidades de tempo) seja também produzida até que todo o horizonte de planejamento seja computado. A limitação de ativação dentro do horizonte de planejamento é feita por meio da inclusão de restrições que impeçam a ativação após a unidade de tempo desejada. Sempre que uma solução parcial é produzida, as tarefas ativadas têm suas variáveis fixadas na formulação de acordo com o tempo de ativação da tarefa. Isto deve ser feito para que o problema não se torne cada vez mais difícil com a possibilidade de realizar o escalonamento em intervalos cada vez maiores. A divisão do horizonte pode ser feita de diversas maneiras. Foram propostas três formas de divisão, utilizando dois critérios distintos, que serão explicados a seguir.

- Intervalos de tamanho fixo: nesta forma de divisão, o horizonte de planejamento será seccionado em intervalos de K unidades de tempo. Por exemplo, se $H = 7$ para uma instância qualquer e $K = 2$, serão executados quatro intervalos: os três primeiros com duas unidades e o último com apenas a unidade de tempo restante.

- Intervalos de tamanhos variáveis: outra forma de realizar a divisão é definir a quantidade de intervalos mas permitir que estes tenham tamanhos distintos. Embora o ideal seja que os intervalos tenham o mesmo tamanho, o que aumenta ou diminui a dificuldade de execução é, principalmente, a quantidade de tarefas (e por consequência de variáveis) que devem ser tratadas em cada intervalo. Assim, intervalos cujos tamanhos tenham uma pequena diferença são perfeitamente aceitáveis.

  Entretanto, testes preliminares mostraram que levar em consideração apenas a quantidade de variáveis binárias não apresenta resultados tão bons, pois os primeiros instantes de tempo terão menos variáveis binárias devido às restrições de menor tempo. Isto acarreta em intervalos que precisam reunir mais unidades de tempo, demorando mais para serem executados. De fato, a contagem que vai determinar o tamanho de cada intervalo será baseada na quantidade de tarefas que têm o menor tempo de ativação dentro do intervalo em questão.

  O pré-processamento computará, em cada tempo t, a quantidade de tarefas i que têm $menor(i) = t$, esta quantidade será denotada por $w(t)$. O valor de n (número total de tarefas) será dividido pela quantidade de intervalos v para saber o tamanho médio m dos intervalos. Os intervalos agregarão consecutivas unidades de tempo até que a soma dos valores $w(t)$ seja o mais próximo possível de m. Se não for possível

atingir exatamente o valor de m, a quantidade que sobrar ou que faltar será levada em consideração no cálculo do próximo intervalo. É importante mencionar que a quantidade de intervalos v é um dado passado pelo usuário do algoritmo; só assim será possível realizar os cálculos necessários para esta forma de divisão do horizonte de planejamento. As Figuras 5.2 e 5.3 mostram como particionar um grafo em 4 ou 6 intervalos, respectivamente, no qual as tarefas podem ser iniciadas até o tempo $t = 8$. Não foram mostrados os arcos para facilitar a visualização das figuras, as tarefas são representadas pelos círculos cinzas.

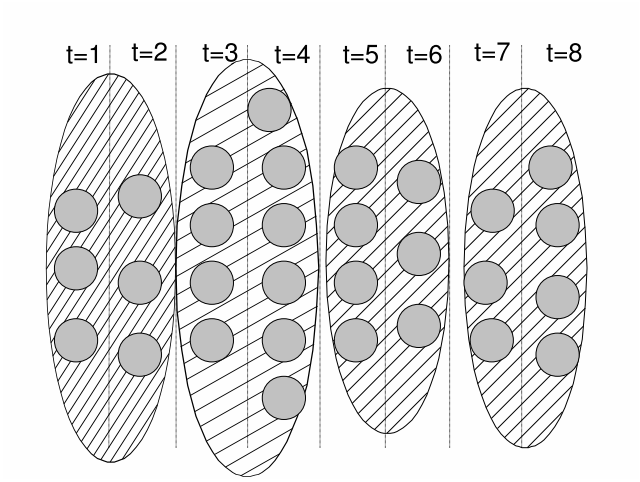


Figura 5.2: Exemplo de divisão do horizonte de planejamento em 4 partições

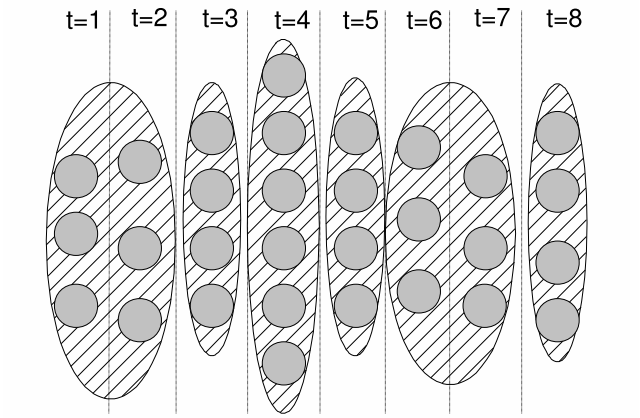


Figura 5.3: Exemplo de divisão do horizonte de planejamento em 6 partições

Como o grafo contém 30 tarefas, a divisão em 4 intervalos na Figura 5.2 nós dá uma média de 7.5 tarefas por intervalo. A primeira unidade de tempo possui apenas 3 tarefas, o que fica muito longe do nosso objetivo (7.5). Incluindo o tempo 2, ficamos

com 6 tarefas. Se incluirmos o tempo 3, teríamos 10 tarefas - mais distante de 7.5 do que o valor 6. Portanto, nosso primeiro intervalo termina em $t = 2$, faltando "uma tarefa e meia" para o objetivo. O segundo intervalo vai levar isto em consideração e buscar por um grupo que chegue perto de $7.5 + 1.5 = 9$ tarefas. A melhor escolha é agrupar os tempos 3 e 4 em outro intervalo, agora com uma tarefa de sobra. O algoritmo prossegue até chegar a $t = 8$, neste caso.

Para a divisão em 6 intervalos, temos um valor médio de 5 tarefas para cada partição. Na Figura 5.3, podemos ver como ficariam estas partições utilizando o mesmo raciocínio empregado no exemplo anterior.

- Múltiplos intervalos de tamanhos variáveis: a idéia é bem semelhante à anterior, mas ao final de cada execução completa, ou seja, após realizar o escalonamento em todo o horizonte de planejamento, os intervalos já definidos serão modificados de forma a ficarem uma unidade de tempo mais longo ou mais curto. Após cada modificação, o escalonamento será realizado levando em consideração a nova configuração. Portanto, este mecanismo executa diversos escalonamentos semelhantes ao item anterior, porém com pequenas alterações no que diz respeito ao tamanho dos intervalos. A Figura 5.4 mostra um exemplo com 5 intervalos ($v = 5$) e $H = 24$. Cada unidade de tempo é representada por um quadradinho e a divisão dos intervalos por uma linha vertical tracejada. Quadradinhos que apresentam o mesmo estilo de fundo estão no mesmo intervalo. As execuções seguem a ordem estabelecida pela figura, de cima para baixo. O particionamento original, no caso com intervalos de tamanho 4, 5, 3, 5 e 7, é definido como no critério anterior (intervalos de tamanho variável).

Estas três formas de seccionamento também fazem com que a solução ótima possa não ser encontrada mais, pois a solução ótima de um escalonamento parcial pode não fazer parte da solução ótima do escalonamento por completo. Assim, estas versões híbridas não podem ser consideradas algoritmos exatos para o PEPRRD.

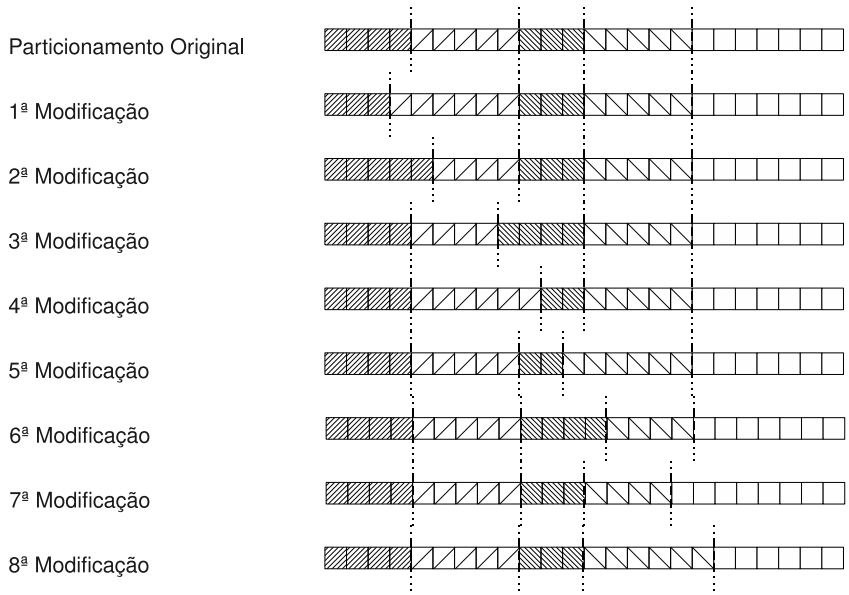


Figura 5.4: Esquema de múltiplos intervalos

Capítulo 6

Resultados Computacionais

Esta seção traz resultados dos principais testes realizados com as heurísticas e com os métodos exatos discutidos nos capítulos anteriores. Os testes foram realizados em computadores Intel Quad-core 9550, com 8 Gigabytes de memória RAM. A versão do CPLEX utilizada foi a 11.2 paralela (com até 4 *threads*). O código dos métodos heurísticos e híbridos foi escrito em linguagem C. Antes, porém, de passar à apresentação dos resultados, faz-se necessário descrever as principais características das instâncias utilizadas.

6.1 Instâncias

Todas as instâncias utilizadas fazem parte do Projeto Labic, disponíveis em [51]. São as únicas instâncias existentes para o PEPRRD, já que as instâncias para outros problemas de escalonamento de projetos não contêm uma característica fundamental que é o lucro produzido pelas tarefas.

A topologia dos grafos foi elaborada para que 10% das tarefas não tenham predecessor algum e as demais tenham de 1 a 5 predecessores aleatoriamente escolhidos, mas de forma a não permitir grafos direcionados cíclicos. O custo e o lucro das tarefas foram aleatoriamente escolhidos dentro dos intervalos [1;50] e [1;10], respectivamente. Custos maiores e/ou lucros menores resultariam em uma grande diminuição do número de tarefas ativadas o que tornaria as instâncias mais fáceis. Por outro lado, custos menores e/ou lucros maiores permitiriam que todas as tarefas sempre valessem a pena serem ativadas o que poderia diminuir a dificuldade da mesma forma. No intervalo em que foram definidos, custos e lucros permitem a confecção de instâncias interessantes, sempre com soluções não

triviais.

A quantidade inicial de recursos Q_0 foi aleatoriamente escolhida no intervalo [MinCusto;50], onde MinCusto é o menor custo entre as tarefas sem precedência. O horizonte de planejamento foi estabelecido com tamanho igual à raiz quadrada do número de tarefas.

Os principais fatores de determinam a dificuldade de uma instância são o número de tarefas e o tamanho do horizonte de planejamento. Uma instância com 5000 tarefas, por exemplo, tende a ser mais difícil do que outra com 2000 tarefas. Uma instância com 20 unidades de tempo também tende a ser mais difícil do que outra com apenas 8 unidades. A distribuição de custos e lucros, como já mencionado, como também a topologia do grafo podem influenciar na dificuldade da instância, porém de forma menos direta.

Nos testes, as instâncias serão indicadas de acordo com a seguinte notação: $xxxay$, onde xxx indica a quantidade de tarefas da instância e y (opcional) indica que se trata da y-ésima instância com aquela quantidade de nós. Por exemplo, 300a5 significa que é a quinta instância com 300 tarefas; 450a indica que é a primeira instância com 450 tarefas (o valor de y foi omitido, por questão de simplificação da notação). As instâncias com a mesma quantidade de tarefas (300a, 300a2, 300a3, 300a4, por exemplo) foram geradas separadamente e não possuem, obrigatoriamente, características em comum, ou seja, não foram geradas a partir de outras.

6.2 Formulações Matemáticas

O primeiro teste analisa o impacto causado por cada uma das restrições adicionais quando incluídas nas formulações F1 e F2. A Tabela 6.1 mostra, para cada instância, o valor da relaxação linear sem qualquer restrição adicional, e a redução percentual causada por cada conjunto de restrições (C1-C4). A coluna C1 indica as restrições de menor tempo de ativação (Seção 3.1), C2 representa as restrições de número máximo de tarefas (Seção 3.2), C3 indica as restrições de pares de tarefas (Seção 3.3) e C4 representa as restrições de soma de variáveis (Seção 3.4). Em negrito está destacado o tipo de restrição que produziu melhor resultado para a formulação F1 e, em itálico, para a formulação F2.

A primeira constatação que pode ser feita é que o valor da relaxação linear da formulação F2 é bem menor e, portanto, melhor do que o valor da formulação F1 (recordando,

	F1					F2			
Inst.	RL	C1	C2	C3	C4	RL	C1	C2	C3
50a	59.1694	**10.05**	3.48	0.00	1.33	57.4709	*7.39*	0.00	0.00
100a	398.4813	5.31	0.95	0.00	**17.43**	322.8573	*0.25*	0.00	0.00
150a	684.8494	3.13	0.00	0.00	**14.02**	588.8171	0.00	0.00	0.00
200a	1208.6459	6.94	1.62	0.94	**39.69**	710.7612	0.00	*2.61*	1.30
250a	1721.6905	5.16	4.62	0.77	**33.45**	1158.6544	1.11	*3.68*	0.66
300a	3255.1332	4.06	1.92	0.00	**33.19**	2174.7781	0.32	*1.61*	0.00
350a	4373.5286	3.99	0.03	0.00	**39.30**	2654.5581	0.00	0.00	0.00
400a	10600.4006	5.79	0.39	0.00	**29.61**	7461.0944	0.00	*0.18*	0.00
450a	13682.4747	0.83	0.02	0.00	**28.27**	9814.6837	0.00	0.00	0.00
500a	18538.3810	0.47	0.00	0.00	**22.15**	14432.8957	0.00	0.00	0.00
550a	17925.9438	1.08	0.00	0.00	**36.93**	11305.3744	0.00	0.00	0.00
600a	18741.3853	2.99	0.00	0.11	**41.68**	10930.0444	0.00	0.00	*0.99*
650a	25945.1970	0.87	0.83	0.83	**33.83**	17168.3087	0.00	1.45	*1.54*
700a	39786.5320	0.66	0.00	0.38	**19.30**	32109.7174	0.00	0.00	*0.51*
750a	46409.8935	0.58	0.01	0.00	**21.39**	36483.4751	0.00	0.00	0.00
800a	48951.3293	0.57	0.00	0.00	**20.68**	38828.4671	0.00	0.00	0.00
850a	57781.0238	0.94	0.00	0.00	**18.47**	47106.5569	0.00	*0.12*	0.00
900a	53651.5671	3.61	2.20	0.00	**26.13**	40398.7892	*3.24*	*3.24*	0.00
950a	78146.6912	0.05	0.00	0.00	**12.98**	68005.4831	0.00	0.00	0.00
1000a	83379.8300	0.51	0.47	0.19	**12.85**	72666.5083	0.00	*0.99*	0.33
Média	-	2.88	0.83	0.16	**25.13**	-	0.62	*0.69*	0.27

Tabela 6.1: Impacto das restrições adicionais em cada formulação

o PEPRRD é um problema de maximização). O fato daquela formulação já garantir que tarefas sucessoras estejam sempre "menos ativadas" que as predecessoras faz uma grande diferença já em um limite dual muito comum de ser computado. Aliás, o impacto causado pela inclusão das restrições das somas das variáveis na formulação F1 também pode ser visto na grande redução percentual mostrada na sexta coluna da Tabela 6.1.

Para a formulação F1, as restrições de menor tempo também tiveram um bom desempenho com quase 3% de redução média. Por outro lado, para a formulação F2, as restrições adicionais causaram efeito bem menor. É importante destacar que as situações pontuais, ou seja, o subconjunto de tarefas, onde as restrições surtem efeito existem para as duas formulações, já que o grafo de entrada é o mesmo. O fato de um mesmo conjunto de restrições adicionais causar diferentes níveis de redução pode ser explicado pelo valor da relaxação linear que serviu de base para o cálculo. Tomemos por exemplo a instância 50a e o conjunto de restrições C1. O valor da relaxação linear para F1 é 59.1694. Uma redução de 10.05% leva este valor para 53.2229 que é o mesmo valor obtido pela relaxação da formulação F2 com a redução de 7.39% causada pelas restrições C1. As restrições adicionais, portanto, são menos efetivas para F2 porque o valor da relaxação neste caso já é menor.

As Tabelas 6.2 e 6.3 apresentam comparações dos limites primais e duais gerados por cada formulação isoladamente e com a inclusão de todas as restrições adicionais propostas para cada formulação (indicado por um "+" na tabela). Todos os parâmetros do CPLEX foram deixados com seus valores padrões e foi dado um tempo limite de 50000 segundos para cada execução.

Comparativamente, a formulação F2 obteve os melhores resultados. Foi a única versão capaz de produzir um limite primal, ou seja, uma solução factível para todas as instâncias dentro do limite de tempo estipulado (50000 segundos). A formulação F1 e sua versão com as restrições não conseguiram sequer produzir uma solução factível em pelo menos 25% das instâncias testadas. Uma das explicações pode estar na quantidade de nós explorados: nas instâncias até 500 tarefas onde o ótimo foi encontrado, o CPLEX com F2 explorou, em média, cerca de 1/9 da quantidade de nós em relação à formulação F1. Isto se reflete também no tempo computacional gasto que poderá ser visto na Tabela 6.4 mais adiante. Já nas intâncias onde o ótimo não pôde ser provado (com mais de 500 tarefas)

Instância	F1	F1+	F2	F2+
50a	47	47	47	47
100a	304	304	304	304
150a	576	576	576	576
200a	636	636	636	636
250a	1030	1030	1030	1030
300a	2073	2073	2073	2073
350a	2571	2571	2571	2571
400a	7265	**7271**	**7271**	**7271**
450a	9633	9574	**9705**	**9705**
500a	14222	**14336**	**14336**	**14336**
550a	0	11072	11087	**11108**
600a	0	10154	10157	**10167**
650a	0	16270	**16332**	16331
700a	0	31784	**31820**	31798
750a	0	0	36216	**36231**
800a	0	0	**38351**	0
850a	45240	0	**46840**	46806
900a	0	38556	38733	**38800**
950a	0	0	**67903**	67896
1000a	0	0	71864	**71879**

Tabela 6.2: Limites primais obtidos pelas formulações

Instância	F1	F1+	F2	F2+
50a	47.0000	47.0000	47.0000	47.0000
100a	304.0000	304.0000	304.0000	304.0000
150a	576.0000	576.0000	576.0000	576.0000
200a	636.0000	636.0000	636.0000	636.0000
250a	1030.0000	1030.0000	1030.0000	1030.0000
300a	2073.0000	2073.0000	2073.0000	2073.0000
350a	2585.2790	**2571.0000**	**2571.0000**	**2571.0000**
400a	7304.9527	**7271.0000**	**7271.0000**	**7271.0000**
450a	10061.5759	10174.7632	**9705.0000**	**9705.0000**
500a	15925.2149	**14336.0000**	**14336.0000**	**14336.0000**
550a	15401.4295	**11132.3064**	11136.4831	11133.4913
600a	14968.1397	10200.5773	10199.5660	**10187.8364**
650a	21800.2408	16431.0530	**16355.8210**	16383.4517
700a	36406.8434	31864.0367	**31836.2076**	31842.2301
750a	42069.0104	36317.8720	36296.8748	**36296.5570**
800a	45306.9727	38732.8392	**38729.7743**	38743.3329
850a	53206.7039	46934.7733	**46897.9824**	46911.6446
900a	47904.0815	**38827.7713**	38834.8116	38829.6680
950a	75062.9671	67939.5572	**67927.4055**	67928.4694
1000a	79287.9344	71924.7402	71911.8527	**71906.8074**

Tabela 6.3: Limites duais obtidos pelas formulações

a quantidade de nós explorados chega ser 4 vezes maior com a formulação F2. Isto pode ser um dos fatores que levou esta formulação a apresentar resultados melhores para as referidas instâncias.

Os limites duais das duas versões de F1 também foram piores em quase todas as instâncias, mostrando que estas versões devem ter uma dificuldade bem maior em provar a otimalidade das soluções caso fosse dado tempo suficiente. É interessante notar que a inclusão das restrições em F1 propiciou uma significativa melhora de desempenho, mas o mesmo não pode ser concluído em relação à F2. Para algumas instâncias os limites primais e duais melhoram (600a, por exemplo), enquanto em outras pioraram (850a, por exemplo).

Na instância 800a, a execução da formulação F2 foi tão alterada que a versão mais complexa não conseguiu produzir uma solução factível. Isto pode ocorrer em algumas situações porque qualquer inclusão, remoção ou alteração nas restrições do modelo afetam diretamente as escolhas de nós a serem explorados, de variáveis a serem fixadas ou mesmo da direção (para cima ou para baixo) que deve ser tomada por parte dos algoritmos *branch-and-bound* (e variações) utilizados pelos otimizadores. Assim, não foi possível perceber uma predominância de uma versão (F2 simples ou com restrições) sobre a outra de forma geral, embora o fato de que a versão com restrições não tenham produzido um limite primal, para uma instância, seja bastante negativo.

Instância	F1	F1+	F2	F2+
50a	0.0	0.0	0.0	0.0
100a	0.2	0.1	0.1	**0.0**
150a	0.5	**0.1**	0.3	0.2
200a	12.3	3.5	6.5	**2.0**
250a	88.2	13.2	25.3	**8.2**
300a	5674.3	67.9	196.3	**42.2**
350a	50000.0	224.7	211.3	**142.2**
400a	50000.0	3153.8	**467.8**	1101.7
450a	50000.0	50000.0	9843.2	**3399.4**
500a	50000.0	8430.5	6688.7	**3376.5**
550a-1000a	50000.0	50000.0	50000.0	50000.0

Tabela 6.4: Tempo gasto (seg) pelas formulações

A Tabela 6.4 mostra o tempo gasto por cada versão para provar a otimalidade da solução encontrada. Quando isto não foi possível, o limite de 50000 segundos foi indicado

na tabela. Novamente, é perceptível uma grande vantagem da formulação F2 em relação à F1. Somente nas instâncias com mais de 550 tarefas todas as versões não terminaram antes do prazo estabelecido.

A inclusão das restrições diminui bastante o tempo gasto em ambas as formulações porque consegue eliminar algumas variáveis por meio da fixação de seus valores, como é o caso da restrições de menor tempo. A redução média no tamanho da matriz de coeficientes com as restrições chega a ser de 20% para F1 e 46% para F2. Além disso a formulação F2, em suas duas versões, foi executada mais rapidamente do que as suas respectivas versões de F1.

De fato, a formulação F2 é composta por restrições mais simples, que não incluem muitos somatórios, como no caso das restrições de precedência de F1, por exemplo. Restrições mais simples, ou seja, com menos coeficientes diferentes de zero, conseguem ser tratadas com mais rapidez pelo algoritmo Simplex, quando uma base ótima é procurada nos diversos Programas Lineares resolvidos ao longo da execução do *branch-and-bound*. Quanto mais rápido o método Simplex encontra a base ótima, mais rápido os nós são processados e, consequentemente, maior a chance de se encontrar uma solução factível, inclusive a solução ótima.

Por apresentar um resultado muito fraco, a formulação F1 não será utilizada nos algoritmo híbridos. No entanto, os resultados obtidos nestes primeiros experimentos serão ainda comparados com algumas outras abordagens para o PEPRRD. Um último experimento levando em consideração apenas a formulação F2 e as restrições adicionais tentou verificar se a utilização destas através do mecanismo presente no CPLEX conhecido como "cortes do usuário" (*user-cuts*, em inglês) traria resultados melhores.

As restrições não seriam adicionadas diretamente ao modelo, como feito até então, mas ficariam armazenadas em uma memória específica do CPLEX e seriam utilizadas caso fosse gerada alguma solução fracionária que as violasse. A idéia por trás disto, é fazer com que a formulação fique mais simples e, consequentemente, possa ser executada mais rapidamente. O experimento mostrou que em apenas metade das instâncias o tempo de processamento foi reduzido. Outros parâmetros como o limite dual e a melhor solução encontrada, inclusive, também foram aprimorados em parte das instâncias, o que não permitiu comprovar um melhor desempenho ao se utilizar desta técnica de cortes do

usuário. A Tabela 6.5 apresenta os valores obtidos, utilizando-se os *user-cuts*.

Instância	Limite Primal	Tempo Total(s)	Limite Dual
50a	47	0.0	47.0000
100a	304	0.1	304.0000
150a	576	0.2	576.0000
200a	636	4.7	636.0000
250a	1030	24.6	1030.0000
300a	2073	193.5	2073.0000
350a	2571	212.0	2571.0000
400a	7271	460.5	7271.0000
450a	9705	9767.2	9705.0000
500a	14336	6555.0	14336.0000
550a	11106	50000.0	11136.5988
600a	10159	50000.0	10199.7436
650a	16332	50000.0	16355.5896
700a	31820	50000.0	31836.4917
750a	36220	50000.0	36297.1078
800a	0	50000.0	38733.4318
850a	46774	50000.0	46928.1209
900a	38733	50000.0	38834.8323
950a	67903	50000.0	67927.3888
1000a	71867	50000.0	71911.7650

Tabela 6.5: Resultados da utilização de *user-cuts*

6.3 Algoritmos Heurísticos

Um marco importante no desenvolvimento de métodos heurísticos foi a mudança na maneira de representar a solução. Os algoritmos evolutivos propostos em [34] e [35], respectivamente EA_priority e EA_ages, apresentaram grandes melhorias em relação aos algoritmos evolutivos previamente propostos. A Tabela 6.6 indica a média (de 30 execuções) dos resultados obtidos por aqueles evolutivos que utilizam a representação indireta (por prioridades) e a representação direta (EA3, proposto em [27]). O critério de parada utilizado foi o número máximo de gerações.

Instância	EA3	EA_priority	EA_ages
100a	**304.0**	303.2	**304.0**
200a	613.1	**636.0**	632.1
300a	1734.3	1958.5	**2004.2**
400a	5500.6	5962.2	**6908.5**
500a	11386.3	11954.3	**13523.7**
600a	7097.8	8616.0	**9358.0**
700a	23775.5	26217.7	**28500.6**
800a	31642.3	32354.4	**35001.9**
900a	28379.1	29912.6	**34355.7**
1000a	60751.9	63512.8	**66399.7**

Tabela 6.6: Média dos resultados dos Algoritmos Evolutivos

É possível perceber pela Tabela 6.6 que, em geral, os algoritmos evolutivos que utilizam a representação indireta são bem superiores em relação ao EA3. Na instância 600a, por exemplo, temos uma melhoria de mais de 30%. Isto se deve muito à forma de representação e ao algoritmo de recombinação utilizados em EA_priority e EA_ages. Para corroborar esta afirmação, um experimento foi realizado para analisar apenas os resultados destes operadores de recombinação. Duas populações iniciais A e B, foram geradas, respectivamente, pelos algoritmos EA3 e EA_priority. Os operadores de recombinação destes evolutivos foram aplicados a cada população, gerando outras duas populações com 10000 indivíduos cada. Neste ponto, temos quatro populações de filhos gerados: duas populações geradas pelo operador de recombinação antigo, sendo uma delas derivada da População A e a outra da População B. O mesmo vale para o operador de recombinação novo que também gerou outras duas populações de filhos.

A aptidão média das proles foi comparada entre os dois operadores. Ou seja, a prole gerada pelo operador antigo (a partir da População A) foi comparada com a prole gerada pelo operador novo. As proles geradas a partir da População B também foram compradas. A Tabela 6.7 mostra quanto a aptidão média da população gerada pelo operador do EA_priority foi melhor do que a da população gerada pelo EA3, em cada uma das populações de filhos derivadas das Populações A e B. Além disso, a Tabela 6.8 mostra o

tempo médio gasto em cada recombinação (em milisegundos).

Instância	Pop. A	Pop. B
100a	11.5%	-4.9%
200a	14.3%	11.7%
300a	25.9%	95.2%
400a	72.1%	87.7%
500a	61.4%	52.9%
600a	-2.4%	48.2%
700a	42.8%	61.4%
800a	21.1%	38.4%
900a	444.7%	385.1%
1000a	411.3%	272.5%

Tabela 6.7: Comparação dos resultados dos operadores de recombinação

	Pop. A		Pop. B	
Instância	EA3	EA_priority	EA3	EA_priority
100a	0.36	0.58	0.42	**0.31**
200a	1.72	1.16	2.63	**0.63**
300a	4.50	1.90	5.29	**1.10**
400a	10.84	2.70	12.09	**1.43**
500a	26.25	3.58	23.24	**1.87**
600a	31.71	4.61	29.73	**2.17**
700a	65.34	5.21	59.55	**2.91**
800a	147.38	5.19	91.61	**3.51**
900a	102.22	5.85	93.24	**4.65**
1000a	174.44	7.43	172.24	**5.74**

Tabela 6.8: Tempo médio gasto em cada recombinação (milisegundos)

É notável que o operador de recombinação do EA_ages e do EA_priority (ambos têm o mesmo operador) consegue gerar indivíduos muito melhores e em muito menos tempo. A explicação para isto vem do fato de que qualquer lista de prioridades é capaz

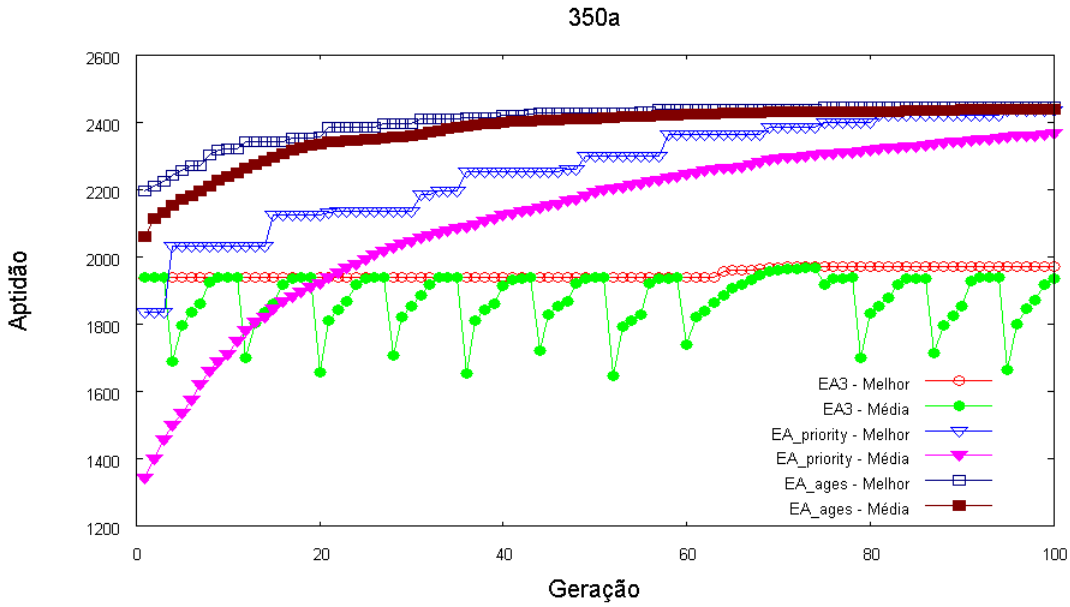
de gerar um escalonamento viável, na representação por prioridades, o que não é verdade na representação direta. Esta, por sua vez, precisa checar a viabilidade da solução, a cada tentativa de fixar um tempo de ativação para uma tarefa, consumindo muito tempo. Para manter a viabilidade da solução, muitas vezes é necessário abrir mão de bons genes provenientes dos pais, o que também faz com que a qualidade dos filhos não seja sempre boa. A Figura 6.1 ilustra situações como esta em duas instâncias (350a e 550a). Cada algoritmo rodou por 100 gerações e foram anotadas a qualidade da melhor solução encontrada e a aptidão média dos indivíduos de cada geração computada pelos evolutivos EA3, EA_priority e EA_ages.

O algoritmo EA3 conseguiu evoluir o melhor indivíduo poucas vezes. Seu operador de recombinação, embora conseguisse produzir populações com boa qualidade em relação ao melhor indivíduo, teve dificuldades em estabelecer novos padrões genéticos de qualidade superior. Também fica visível, pelos gráficos, que os mecanismos de intensificação e diversificação do EA3 (descritos na página 18) tentaram cumprir seu papel de promover uma reinicialização da população quando esta foi considerada estagnada.

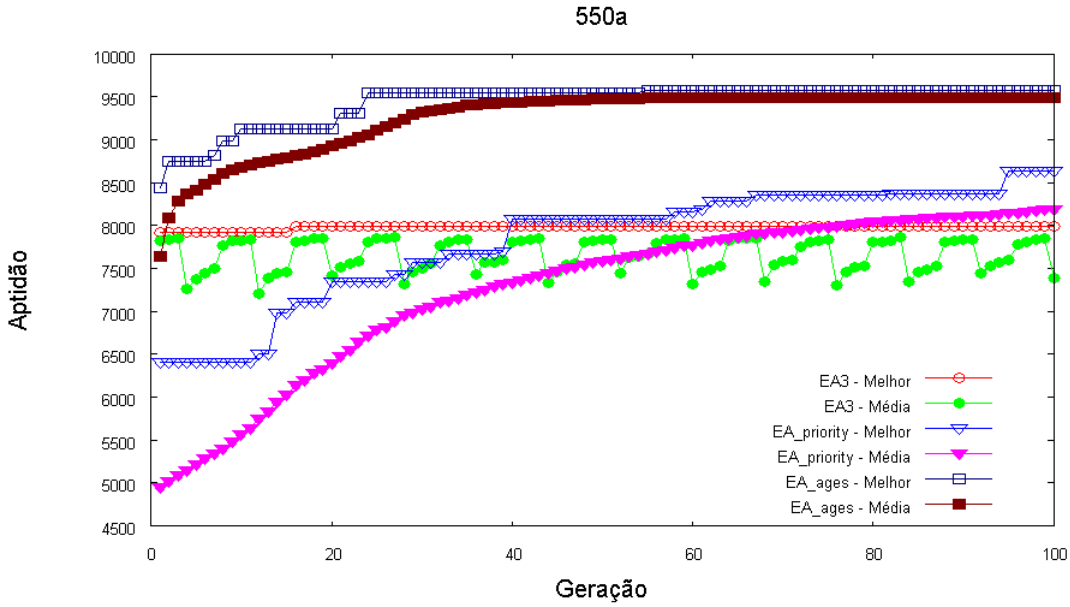
O operador de recombinação dos evolutivos EA_priority e EA_ages teve desempenho bem melhor, já que um número bem maior de evoluções foi conseguido. O EA_ages mostrou um desempenho um pouco melhor, inclusive em relação à média da população que foi sempre bem mais próxima da melhor solução do que a apresentada pelo EA_priority.

Fica bem claro que a chance de gerar indivíduos melhores do que os pais é bem maior usando a representação indireta e o operador de recombinação correspondente. Mesmo assim, após mais de 200 ou 300 gerações, a população começa a estagnar, ou seja, a não produzir indivíduos melhores. O algoritmo EA_ages [35] foi desenvolvido para tentar superar estas dificuldades através de uma série de novos operadores. A aplicação da LS_swap, que troca pares de prioridades, ajudou a aumentar a qualidade dos indivíduos que eram considerados novos *best*. Mas sem dúvida, o operador que permitiu adiar o estado de estagnação da população foi a reconstrução da população baseada no melhor indivíduo. Esta reconstrução, como já foi mencionado, é ativada quando a população é considerada estagnada, ou seja, quando não há indícios de que ela possa aprimorar mais a qualidade de seu melhor indivíduo, o que ocorre após 150 gerações sem melhorias.

Para ilustrar a capacidade deste esquema, a Tabela 6.9 apresenta o resultado do



(a)



(b)

Figura 6.1: Evolução da melhor solução e da média da população

melhor indivíduo (em termos percentuais), ao final de cada Era, em relação à primeira. Os valores são médias de 20 execuções. As instâncias com menos de 250 tarefas não foram analisadas, porque a solução ótima é encontrada ainda na primeira Era, logo, não há como melhorá-la mais.

	Eras								
Instância.	2^a	3^a	4^a	5^a	6^a	7^a	8^a	9^a	10^a
250a	0.21	0.25	0.25	0.31	0.31	0.31	0.31	0.31	0.31
300a	0.69	0.84	0.95	1.05	1.06	1.06	1.06	1.06	1.09
350a	0.00	0.10	0.11	0.17	0.17	0.17	0.17	0.18	0.18
400a	0.53	0.62	0.82	0.83	0.91	0.93	1.00	1.03	1.03
450a	1.10	1.41	1.63	1.74	1.77	1.83	1.85	1.85	1.86
500a	0.20	0.32	0.36	0.48	0.55	0.64	0.73	0.83	0.93
550a	0.95	1.64	1.65	1.68	1.73	1.74	1.87	1.89	1.90
600a	0.82	1.31	1.34	1.46	1.56	1.67	1.70	1.77	1.80
650a	1.28	1.70	2.36	2.52	2.57	2.65	2.68	2.74	2.74
700a	1.14	1.72	2.07	2.50	2.67	2.91	2.98	3.06	3.15
750a	0.85	1.35	1.48	1.77	1.97	2.20	2.24	2.36	2.37
800a	0.65	1.01	1.20	1.54	1.80	1.89	2.05	2.13	2.21
850a	1.30	1.61	1.96	2.05	2.12	2.18	2.22	2.25	2.36
900a	1.01	1.61	1.96	2.22	2.55	2.68	2.79	2.85	2.91
950a	0.80	1.14	1.39	1.64	1.78	1.90	2.10	2.24	2.27
1000a	0.56	0.87	1.10	1.26	1.41	1.53	1.58	1.66	1.69
Média	0.76	1.09	1.29	1.45	1.56	1.64	1.71	1.76	1.80

Tabela 6.9: Evolução das melhor solução ao longo das Eras

Embora nas primeiras eras o melhor indivíduo seja aprimorado até um pouco mais de 1%, é interessante ver que com o passar da eras, este aprimoramento chega a mais de 2.2% em várias instâncias, com média de 1.8%. Pode não ser um resultado brilhante, mas para populações que eram tidas como estagnadas e não tinham perspetivas de melhora, é um resultado a ser considerado.

Outra contribuição importante de [35] foi a versão paralela do EA_ages. Na verdade, foram propostas duas versões: uma que utiliza duas *threads* e outra que utiliza quatro. O *speedup* das duas versões paralelas pode ser visto na Tabela 6.10. Já a Tabela 6.11 apresenta os resultados de um teste que teve como critério de parada o tempo de 1 hora. Em cada marco de tempo (10, 100, 1000 ou 3600 segundos) é mostrado o valor da melhor solução encontrada por cada versão analisada.

O *speedup* das versões com 2 e 4 *threads* ficou um pouco abaixo do esperado, porque nem todos os operadores evolutivos foram paralelizados. Isto faz com que nem todo o

Instância	2 *threads*	4 *threads*
300a	1.453	2.131
400a	1.802	2.653
500a	1.851	2.273
600a	1.799	2.826
700a	1.673	2.176
800a	1.926	2.390
900a	1.855	2.073
1000a	1.788	2.302

Tabela 6.10: *Speedup* das versões *multi-thread* do evolutivo EA_ages

Instância	*threads*	10s	100s	1000s	3600s
400a	1	6519	6979	7025	7025
	2	6424	7000	**7100**	7167
	4	**6762**	**7009**	7094	**7174**
700a	1	25236	28124	29413	29796
	2	25916	28277	29661	**30108**
	4	**26214**	**28960**	**29674**	30070
1000a	1	46618	64885	67978	68765
	2	**59360**	64904	68332	69431
	4	58992	**65795**	**68710**	**69494**

Tabela 6.11: Evolução da melhor solução nas três versões do EA_ages

algoritmo possa ser considerado paralelo e acaba se refletindo nos valores mensurados. No entanto, a Tabela 6.11 traz resultados que mostram a capacidade das versões paralelas de produzir resultados melhores em menor espaço de tempo. Em cada um dos quatro marcos, uma das versões paralelas tinha sempre uma solução melhor que a versão sequencial. A versão com quatro *threads* teve um desempenho geral melhor, inclusive, do que a versão com apenas duas. Embora esperado, na prática esta situação nem sempre acontece.

Entre as versões de GRASP propostas (GRASP1 e GRASP2) o teste consistiu em analisar os resultados obtidos após 30 execuções para cada instância. Como a segunda versão apresenta métodos a mais do que primeira, no caso a recombinação de soluções armazenadas em um *pool*, o critério de parada estabelecido foi um tempo limite que varia de acordo com o tamanho da instância. A fórmula para calcular a quantidade de segundos dada a cada algoritmo é: $\frac{n^2}{1000}$, onde n é o número de tarefas da instância. A Tabela 6.12 mostra a média dos resultados obtidos, sendo a melhor em negrito. Também está presente a melhor solução encontrada por cada versão.

Em todas as instâncias testadas, a versão GRASP2 teve os melhores resultados e as

	Médias		Melhores Resultados	
Instância	GRASP1	GRASP2	GRASP1	GRASP2
50a	47.0	47.0	47	47
100a	304.0	304.0	304	304
150a	574.6	**575.1**	576	576
200a	633.9	**634.3**	636	636
250a	1015.7	**1023.2**	1030	1030
300a	1885.8	**1996.5**	1952	*2031*
350a	2284.6	**2428.3**	2384	*2482*
400a	6105.8	**6487.0**	6359	*6781*
450a	8119.9	**8579.1**	8487	*8974*
500a	11650.6	**12427.6**	12143	*12799*
550a	8517.4	**9231.1**	8912	*9673*
600a	7696.8	**8392.8**	8088	*8803*
650a	12058.8	**13221.5**	12714	*13725*
700a	24394.1	**26278.4**	25716	*27358*
750a	29980.4	**31505.0**	31019	*32390*
800a	31179.7	**33050.5**	32228	*34218*
850a	37662.8	**40013.0**	39145	*41496*
900a	28543.3	**30436.1**	29793	*31541*
950a	59262.6	**61359.0**	60278	*62187*
1000a	59848.0	**62806.7**	61752	*64500*

Tabela 6.12: Comparação dos resultados das versões GRASP

melhores médias. De fato, a utilização de várias boas soluções juntamente com o algoritmo de recombinação foi um dos fatores que permitiu que também os evolutivos obtivessem bons resultados. No GRASP2, cerca de 73% das soluções que, em algum momento da execução do algoritmo foram consideradas as melhores, são provenientes da recombinação.

A Tabela 6.13 apresenta os resultados de testes semelhantes, porém com as duas versões ILS propostas. Em negrito está a melhor média e, em itálico, a melhor solução produzida pelos algoritmos.

Diferentemente do esperado, o emprego do algoritmo de recombinação neste caso não melhorou a qualidade dos resultados obtidos pelo ILS2, em comparação com o ILS1. Apenas em algumas instâncias, esta melhoria pôde ser observada, como na 700a, por exemplo. A explicação para isto está no fato de que a recombinação foi aplicada a soluções que não tinham garantia alguma de serem boas soluções, já que as perturbações podem gerar soluções melhores, mas também podem piorar a qualidade de uma solução original. A Tabela 6.14 mostra a quantidade percentual de melhorias promovidas por cada perturbação e pela recombinação.

	Médias		Melhores Resultados	
Instância	ILS1	ILS2	ILS1	ILS2
50a	47.0	47.0	47	47
100a	304.0	304.0	304	304
150a	575.8	**576.0**	576	576
200a	**635.9**	635.3	636	636
250a	**1024.5**	1020.4	1030	1030
300a	**1934.7**	1903.8	*2011*	2006
350a	**2353.5**	2315.2	*2449*	2406
400a	**6185.8**	6078.5	*6534*	6376
450a	**8103.2**	8097.7	*8733*	8512
500a	11477.4	**11794.2**	11929	*12310*
550a	8553.8	**8675.7**	9122	*9271*
600a	**7906.3**	7797.4	*8411*	8178
650a	**12464.3**	12096.5	*13566*	12833
700a	24014.2	**24372.8**	25571	*26613*
750a	**29339.6**	28581.8	*30791*	29992
800a	**30851.8**	30631.2	*32509*	31899
850a	**36930.3**	36650.9	*39703*	39020
900a	**28789.2**	28090.1	*31038*	29289
950a	**57192.1**	56482.3	*59266*	59022
1000a	**59387.1**	58471.6	*62398*	61591

Tabela 6.13: Comparação dos resultados das versões ILS

Se comparado ao caso do GRASP2, onde cerca de 73% das soluções consideradas melhores foram produzidas pela recombinação, a recombinação no ILS2 teve um desempenho fraco com apenas 39%. Neste caso, onde as soluções a serem recombinadas podem apresentar configurações, isto é, conjuntos de prioridades, muito diferentes por causa das perturbações, a recombinação não consegue mostrar o mesmo potencial dos outros casos onde as soluções apresentam uma qualidade mínima assegurada. Para mostrar que a qualidade das soluções trabalhadas pelo ILS é realmente bastante inferior à dos algoritmos evolutivos, a Tabela 6.15 é apresentada, comparando os resultados da melhor de cada uma das três meta-heurísticas estudadas (Algoritmos Evolutivos - EA_ages; GRASP - GRASP2; ILS - ILS1). O critério de parada continua sendo um tempo limite calculado em função do número de tarefas de cada instância. Em negrito está a melhor média e, em itálico, a melhor solução.

Os resultados deixam claro que apenas em instâncias pequenas (até 300 tarefas) o ILS1 consegue manter o mesmo desempenho das demais meta-heurísticas. O desempenho do GRASP2, com recombinação de soluções, também é inferior ao do EA_ages, mas em

Instância	Pert1	Pert2	Pert3	Pert4	Recomb.
50a	0.00	0.00	0.00	0.00	0.00
100a	0.00	0.00	0.00	0.00	0.00
150a	0.06	0.16	0.10	0.00	**0.68**
200a	0.09	0.17	0.08	0.00	**0.66**
250a	0.07	0.18	**0.40**	0.00	0.36
300a	0.04	0.06	**0.52**	0.00	0.39
350a	0.01	0.09	**0.58**	0.00	0.32
400a	0.04	0.07	0.40	0.00	**0.48**
450a	0.10	0.05	0.37	0.01	**0.48**
500a	0.08	0.05	0.28	0.01	**0.59**
550a	0.07	0.11	0.24	0.02	**0.56**
600a	0.05	0.05	0.39	0.00	**0.50**
650a	0.07	0.12	0.35	0.02	**0.44**
700a	0.12	0.09	0.21	0.02	**0.56**
750a	0.27	**0.33**	0.22	0.07	0.11
800a	0.20	**0.27**	0.24	0.05	0.25
850a	0.28	**0.34**	0.14	0.06	0.18
900a	0.16	0.15	0.24	0.05	**0.40**
950a	0.25	**0.30**	0.25	0.06	0.13
1000a	0.32	**0.47**	0.12	0.09	0.00
Média	0.12	0.17	0.29	0.03	**0.39**

Tabela 6.14: Comparação da contribuição percentual (de 0.00 a 1.00) das perturbações e da recombinação para a geração dos soluções

menor grau. O resultado, de fato, já era esperado por alguns motivos: (i) os algoritmos evolutivos vêm sendo usados para resolver o PEPRRD há bastante tempo, o que permitiu que várias mecanismos fossem desenvolvidos para superar as dificuldades encontradas; (ii) o fato de que qualquer lista de prioridade pode gerar uma solução factível permite que os indivíduos possam ser manipulados livremente, o que é uma característica muito útil nos algoritmos evolutivos e genéticos, como pode ser visto também em [52]; (iii) a utilização do operador de recombinação praticamente requer que sejam utilizadas soluções com um mínimo de qualidade assegurada, caso contrário, as soluções geradas podem não ser melhores do que os pais.

A Figura 6.2 traz uma análise probabilística semelhante à encontrada em [53]. Foi estabelecido um alvo comum às três meta-heurísticas testadas, equivalente à média dos resultados do ILS1 em cada uma das três instâncias aleatoriamente escolhidas (450a, 650a e 900a). Foi anotado o tempo em que a meta-heurística levou para alcançar o alvo. A i-ésima execução mais rápida de cada algoritmo recebeu uma probabilidade $p_i = (i * \frac{100}{30})$,

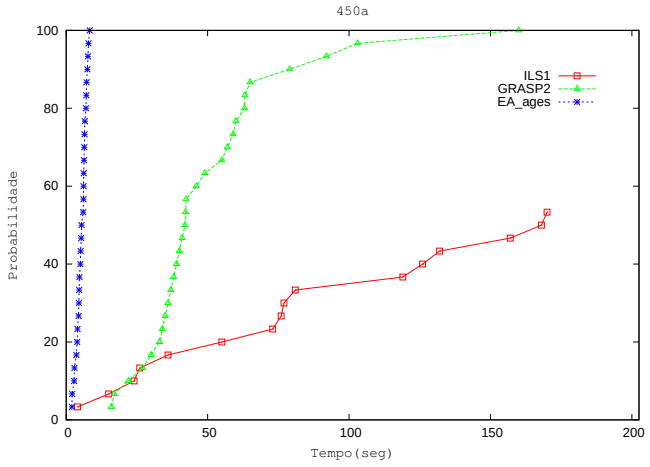
	Média			Melhores Resultados		
Instância	EA_ages	GRASP2	ILS1	EA_ages	GRASP2	ILS1
50a	47.0	47.0	47.0	47	47	47
100a	304.0	304.0	304.0	304	304	304
150a	**576.0**	575.1	575.8	576	576	576
200a	**636.0**	634.3	635.9	636	636	636
250a	1018.2	1023.2	**1024.5**	1023	*1030*	*1030*
300a	1991.8	**1996.5**	1934.7	2009	*2031*	2011
350a	**2519.1**	2428.3	2353.5	*2549*	2482	2449
400a	**6979.4**	6487.0	6185.8	*7053*	6781	6534
450a	**9283.6**	8579.1	8103.2	*9378*	8974	8733
500a	**13708.7**	12427.6	11477.4	*13784*	12799	11929
550a	**10016.1**	9231.1	8553.8	*10179*	9673	9122
600a	**9652.1**	8392.8	7906.3	*9807*	8803	8411
650a	**14886.7**	13221.5	12464.3	*15134*	13725	13566
700a	**28987.1**	26278.4	24014.2	*29249*	27358	25571
750a	**33684.4**	31505.0	29339.6	*34083*	32390	30791
800a	**36035.6**	33050.5	30851.8	*36382*	34218	32509
850a	**43889.0**	40013.0	36930.3	*44276*	41496	39703
900a	**35256.9**	30436.1	28789.2	*35763*	31541	31038
950a	**64189.0**	61359.0	57192.1	*64906*	62187	59266
1000a	**67684.3**	62806.7	59387.1	*68343*	64500	62398

Tabela 6.15: Comparação da melhor versão de cada Meta-heuristica estudada

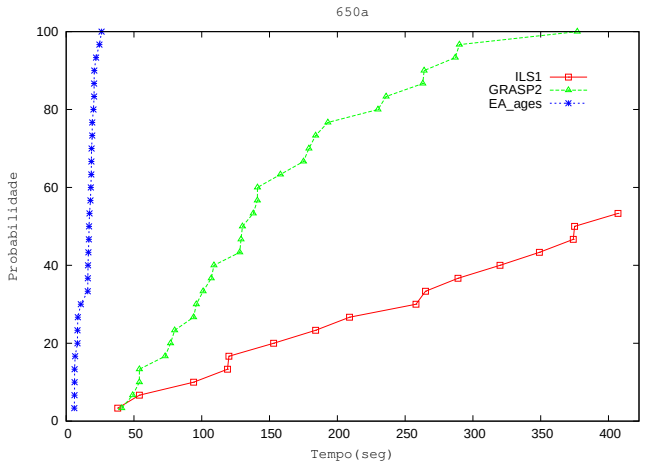
uma vez que foram realizadas 30 execuções. Os gráficos mostram os dados (t_i, p_i), onde t_i é o tempo que a i-ésima execução levou para atingir o alvo. Caso uma execução não atinja o alvo, ela não será mostrada no gráfico. O tempo limite para cada execução é o mesmo utilizado no experimento da Tabela 6.15.

A Figura 6.2 ilustra o fraco desempenho do ILS1 em atingir o alvo. Em aproximadamente metade dos casos, isto nem sequer foi possível. Quando o alvo foi atingido, o ILS1 precisou, em geral, de mais tempo do que as outras meta-heurísticas. O algoritmo GRASP2 conseguiu alcançar os alvos em todas as execuções, mas em alguns casos precisou de mais da metade do tempo dado. O bom desempenho do EA_ages deve ser destacado não só por ele ter conseguido atingir o alvo em todas as execuções, mas principalmente por ter feito isto em poucos segundos. Em praticamente todas as execuções, antes das outras meta-heurísticas atingirem o alvo pela primeira vez, o EA_ages já havia atingido o alvo em todas as 30 execuções.

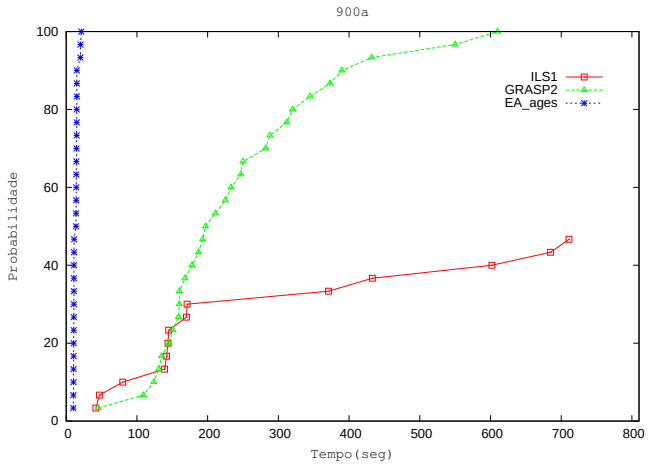
Vale reforçar que o método de construção da solução inicial, tanto no GRASP2 quanto no ILS1, é o mesmo empregado pelo EA_ages na construção da população inicial. Logo,



(a)



(b)



(c)

Figura 6.2: Análise probabilística da melhor versão de cada Meta-heurística estudada

não há favorecimento algum por parte do EA_ages. A única vantagem que ele possui é o fato de, já na primeira geração, ter um conjunto de indivíduos organizados por suas aptidões e uma política de escolha de bons candidatos dentre estes indivíduos. De fato, um dos pais será escolhido dentre os 20% melhores indivíduos, o que já garante um nível mínimo de qualidade para o operador de recombinação.

Também muito importante de ser mencionado é o fato de que o modelo de paralelização adotado pelo EA_ages paralelo não têm influência direta na qualidade das soluções geradas, já que os operadores têm apenas a sua carga de trabalho dividida entre os núcleos de processamento. Neste contexto, a versão paralela do EA_ages tem potencial para atingir os mesmos resultados da versão sequencial, porém consumido um pouco menos de tempo. Como o *speedup* das versões paralelas ficou abaixo do esperado, dificilmente as versões com 2 e 4 *threads* atingiriam o alvo em, respectivamente, metade ou um quarto do tempo mostrado na Figura 6.2.

6.4 Métodos Híbridos

O primeiro experimento com os métodos híbridos levou em conta o algoritmo EA_ages +CPLEX e a formulação F2, sem as restrições adicionais. Esta última escolha foi feita porque apresentou resultados tão bons quanto sua versão contendo as restrições. Além disso, para uma instância (800a) a utilização das restrições fez com que não fosse possível gerar uma primeira solução factível dentro do tempo anteriormente estipulado.

O teste atual consiste em executar o método híbrido e a formulação F2, em algumas instâncias, durante 25000 segundos (metade do tempo dado às formulações nos testes do início deste capítulo). Quatro atributos de execução foram analisados: o tempo total consumido, o valor da melhor solução encontrada, o tempo em que esta solução foi gerada e o limite dual ao final da execução.

Cabe, porém, uma explicação sobre alguns aspectos práticos que interferiram na realização do experimento. Para se implementar a troca de informações entre o método EA_ages e o CPLEX fez-se necessário lançar mão dos chamados *callbacks* do otimizador. *Callbacks* podem ser entendidos como possibilidades de incorporar e executar trechos de código de terceiros, em momentos pré-determinados. Exemplos de situações onde isto

pode ocorrer no CPLEX são: a descoberta de uma nova solução incumbente, a escolha do nó a ser explorado, a escolha da variável a ser fixada, entre outros.

O problema é que ao utilizar um *callback*, o CPLEX automatica e obrigatoriamente muda o algoritmo de otimização para o chamado *branch-and-bound* tradicional. O algoritmo padrão é chamado de *dynamic search*, mas não é sequer publicamente documentado, ao que nós sabemos. Existem poucas frases no *site* do fabricante [28] que dizem que este algoritmo traz "inovações" na escolha das variáveis a serem fixadas, nos nós a serem explorados e na geração de cortes, contudo, sem dizer como isto será feito. O algoritmo *dynamic search* pode ser considerado, portanto, uma caixa preta, o que torna mais difícil uma comparação justa com outros métodos, até porque não é possível utilizá-lo juntamente com códigos próprios, como já foi mencionado.

Nossa intenção inicial foi desprezar a utilização deste algoritmo e realizar as comparações apenas com o *branch-and-bound* tradicional, já que, afinal, este último seria o algoritmo utilizado pelo método híbrido - nós acreditamos que esta seria uma comparação mais justa. No entanto, como o *dynamic search* é o algoritmo padrão do CPLEX (pelo menos na sua versão 11.2) e como muitas pessoas podem estar utilizando-o inadvertidamente, decidimos realizar as comparações também com ele.

Uma última consideração sobre os parâmetros do CPLEX ainda precisa ser feita. Este programa permite que seja escolhido um entre cinco tipos de ênfase que será dada ao processo de otimização. Uma destas ênfases (número 4) procurar fazer com que novas soluções factíveis sejam encontradas mais rapidamente. Como mencionado nas Seções 5.1 e 5.1.1, será fornecido ao CPLEX uma lista de prioridades para a fixação das variáveis da formulação, com o intuito de fazer com que variáveis relacionadas às primeiras unidades de tempo sejam fixadas primeiro.

Se esta fixação acontecer, pode fazer com que muitas outras sejam automaticamente fixadas (devido às restrições do problema), além de fazer que os limites duais sejam menores e que a otimização termine mais cedo. A lista de prioridades, portanto, visa um processamento mais eficiente dos nós, embora se trate de artifício heurístico.

Se por um lado existe a preocupação em se reduzir os limites duais com a lista de prioridades, por outro deve haver algum mecanismo que tente melhorar os limites primais,

ou seja, o valor da melhor solução encontrada. Isto já é feito, em parte, pelo EA_ages quando é entregue ao CPLEX uma melhor solução gerada pelo evolutivo. Para potencializar a descoberta de novos limites primais, o CPLEX terá sua ênfase ajustada para o valor 4, fazendo com que mais processamento seja aplicado às soluções factíveis e delas sejam derivados os limites desejados.

Por tudo o que foi explicado, temos então três algoritmos que farão parte deste experimento: algoritmo híbrido EA_ages +CPLEX com todos os seus componentes mencionados ao longo deste texto e sua ênfase definida com o valor 4, o *dynamic search* com seus parâmetros e ênfase padrão (ênfase 0) e o *branch-and-bound* tradicional que terá sua ênfase trocada para o valor 4 (para torná-lo ainda mais semelhante ao algoritmo híbrido). A Tabela 6.16 e a Tabela 6.17 mostram os quatro atributos de execução que foram considerados nestes testes. O símbolo "-" indica que o tempo total consumido pelo algoritmo foi igual ao tempo limite de 25000 segundos (na Tabela 6.16) ou que o limite dual (na Tabela 6.17) é igual à melhor solução encontrada. Obviamente, este último fato só ocorrerá se a tal solução for uma solução ótima. Além das instâncias já testadas, um novo conjunto de instâncias será incluído no experimento, visando embasar melhor as considerações a serem feitas. Em negrito é destacado o menor tempo total e o melhor limite dual. Em itálico é destacada a melhor solução encontrada ou o menor tempo em que isto ocorreu.

Comparando os resultados das versões DS0 (*dynamic search* com ênfase 0 - padrão) e BB4 (*branch-and-bound* com ênfase 4) pode-se perceber, na maioria das instâncias, que a versão DS0 produziu soluções melhores ou consumiu menos tempo para alcançar o valor ótimo. Nas instâncias 350a3, 350a4, 350a7 e 400a2, a versão BB4 não conseguiu nem terminar a otimização antes do tempo limite. A aparente superioridade da versão DS0 é de difícil explicação, já que pouco se sabe sobre as técnicas utilizadas por esta versão.

Realizando a comparação do DS0 com o híbrido EA_ages +CPLEX (resultados médios de 10 execuções), pode-se notar que a versão híbrida consegue produzir soluções ainda melhores ou consumir ainda menos tempo, na maioria das instâncias. Das 10 instâncias onde não foi possível encontrar o valor ótimo (550a-1000a), em 7 delas a versão híbrida produziu resultados médios melhores e em outras 7 o limite dual foi melhor. Das 22 instâncias onde o tempo limite não foi estourado, em 17 delas a versão híbrida foi mais rápida, terminando mais cedo. Também é interessante notar que a melhor solução

	Tempo Total(seg)			Solução		
Inst.	DS0	BB4	Híbrido	DS0	BB4	Híbrido
300a	196.3	**178.9**	2460.4	2073	2073	2073.0
350a	211.3	408.6	**155.0**	2571	2571	2571.0
400a	467.8	1609.2	**163.7**	7271	7271	7271.0
450a	9843.2	7570.0	**1143.7**	9705	9705	9705.0
500a	6688.7	7228.1	**822.6**	14336	14336	14336.0
550a	-	-	-	*11081*	11054	11054.7
600a	-	-	-	10157	10155	*10159.2*
650a	-	-	-	*16332*	16308	16331.3
700a	-	-	**15061.2**	31820	31818	*31826.0*
750a	-	-	-	36216	36220	*36225.7*
800a	-	-	-	0	0	*38641.1*
850a	-	-	-	46835	0	*46837.2*
900a	-	-	-	38723	38650	*38787.9*
950a	-	-	-	67902	67854	*67909.0*
1000a	-	-	-	*71851*	71844	71838.6
300a2	**386.9**	435.4	1210.0	2399	2399	2399.0
300a3	358.6	18679.0	**90.6**	2340	2340	2340.0
300a4	1732.4	2410.9	**771.5**	2434	2434	2434.0
300a5	**486.6**	1358.3	486.9	4193	4193	4193.0
300a6	96.1	76.3	**38.3**	1257	1257	1257.0
300a7	191.0	**95.9**	112.7	1707	1707	1707.0
300a8	480.3	629.8	**105.9**	2009	2009	2009.0
300a9	**310.4**	986.7	446.5	2915	2915	2915.0
300a10	111.5	93.8	**46.6**	3233	3233	3233.0
350a2	3319.7	9459.7	**1925.3**	3945	3945	3945.0
350a3	7403.3	-	**1198.9**	3521	3521	3521.0
350a4	22771.9	-	**18731.5**	*4128*	4126	*4128.0*
350a5	-	-	**387.7**	3614	3614	3614.0
350a6	**770.9**	870.1	820.9	8328	8328	8328.0
350a7	4662.9	-	**1642.8**	3586	3586	3586.0
400a2	2895.8	-	**464.7**	11254	11254	11254.0
400a3	9745.5	14087.9	**3681.8**	5296	5296	5296.0
400a4	**1443.2**	5944.6	2724.0	10229	10229	10229.0

Tabela 6.16: Tempo gasto e melhor solução do híbrido EA_ages +CPLEX versus CPLEX

	Limite Dual			Tempo da Melhor Solução(seg)		
Inst.	DS0	BB4	Híbrido	DS0	BB4	Híbrido
300a	-	-	-	*155.3*	176.3	1266.8
350a	-	-	-	*70.1*	191.0	104.0
400a	-	-	-	440.6	1209.7	*120.4*
450a	-	-	-	9693.3	7544.5	*1113.9*
500a	-	-	-	896.8	851.2	*466.1*
550a	**11140.6163**	11152.8823	11162.1974	9221.6	22826.2	*6802.0*
600a	10205.1403	10239.7558	**10197.8842**	*4234.9*	18181.5	8845.4
650a	16363.3714	16446.5625	**16351.0578**	*5878.1*	24931.1	14068.8
700a	31843.8330	31854.0408	-	*6966.7*	19862.4	14541.9
750a	36301.9591	36316.9223	**36291.0778**	23388.6	20855.4	*19751.1*
800a	38736.3259	38757.7644	**38685.9676**	25000.0	25000.0	*12336.2*
850a	**46912.8588**	46968.5794	46920.4890	*21658.4*	25000.0	23233.0
900a	38837.0121	38845.0830	**38823.8940**	*14641.6*	24919.5	15187.5
950a	67931.0795	67943.6728	**67916.7546**	24982.8	17380.7	*14811.1*
1000a	**71914.0955**	71920.0715	71930.2535	*4326.4*	24372.2	18270.8
300a2	-	-	-	*224.2*	435.4	591.2
300a3	-	-	-	184.4	280.7	*19.7*
300a4	-	-	-	1000.1	163.8	*36.2*
300a5	-	-	-	*360.7*	991.7	477.2
300a6	-	-	-	74.3	71.7	*30.7*
300a7	-	-	-	60.8	43.4	*18.0*
300a8	-	-	-	105.3	403.9	*100.1*
300a9	-	-	-	310.3	*188.5*	232.0
300a10	-	-	-	96.8	70.8	*45.1*
350a2	-	-	-	662.7	1396.2	*262.5*
350a3	-	3524.9956	-	7347.9	*751.5*	1167.7
350a4	-	4147.4763	-	16520.1	21064.0	*1215.2*
350a5	3617.2861	3629.6628	-	180.0	2817.0	*32.4*
350a6	-	-	-	*576.3*	758.5	626.2
350a7	-	3594.4491	-	1265.4	3759.8	*1213.9*
400a2	-	11267.2867	-	1027.7	6210.9	*230.8*
400a3	-	-	-	8056.3	*439.3*	1850.5
400a4	-	10230.0228	-	*1289.5*	5785.2	2646.9

Tabela 6.17: Limite dual e tempo da melhor solução do híbrido EA_ages +CPLEX versus CPLEX

encontrada pelo método híbrido foi gerada em tempo menor em relação ao DS0 e ao BB4, na maioria das instâncias.

A instância 700a é um caso à parte. Nela, o método híbrido conseguiu encontrar o valor ótimo em aproximadamente 15000 segundos, o que é um ótimo resultado se considerarmos que, nos primeiros testes deste capítulo, utilizando várias formulações, isto não foi possível mesmo em 50000 segundos. A explicação para tantos bons resultados se deve às primeiras soluções que são geradas pelo evolutivo e aprimoradas por ele e pelo CPLEX. Estas soluções permitem que o limite primal seja rapidamente elevado (em relação a uma execução normal do evolutivo ou do CPLEX), reduzindo o número de nós a serem analisados e possibilitando ao CPLEX executar heurísticas de aprimoramento sobre uma solução de qualidade melhor. Também a prioridade para fixação de variáveis auxilia dentro deste contexto. Se não for utilizada, os tempos totais aumentam aproximadamente 14 vezes, em média.

Outro fato que não pode ser esquecido é a utilização dos núcleos de processamento. A versão CPLEX 11.2 paralela utiliza todos os 4 núcleos de processamento do equipamento utilizado, por meio de 4 *threads* que rodam simultaneamente. A versão híbrida EA_ages +CPLEX utiliza, além das 4 *threads*, mais um processo que é o algoritmo evolutivo. Portanto, nesta versão, pode-se considerar que existem 5 processos concorrendo por quatro núcleos de processamento, o que reduz o desempenho de cada um para 80% do seu potencial. Este fato torna ainda mais significativos os resultados mostrados nas tabelas anteriores.

Embora a versão DS0 tenha apresentado melhores resultado do que a BB4, na nossa opinião a comparação mais justa seria com esta última versão, pois como foi dito a utilização de *callbacks* força o CPLEX a utilizar o *branch-and-bound* tradicional. Se esta comparação for levada em consideração, o resultado do método híbrido prevalece ainda mais.

6.4.1 CPLEX como busca local

Para realizar os testes com este outro método híbrido, dois parâmetros precisaram ser cuidadosamente calibrados: o tempo máximo de execução das buscas locais feitas pelo

CPLEX e o valor de Z, ou seja, quantas unidades de tempo a mais ou a menos poderão ser analisadas para cada tempo de ativação das tarefas. Um valor grande para Z certamente possibilitará encontrar vizinhos melhores, mas o tempo computacional necessário para tanto também crescerá bastante.

Preferimos adotar valores pequenos (entre 2 e 4) para permitir que as buscas sejam mais breves e que o evolutivo EA_ages consiga chamá-las com mais frequência. O valor $Z = 2$ se mostrou menos eficiente que os outros dois, pois conseguia encontrar vizinhos melhores com muito menos frequência, principalmente se as soluções originais eram muito boas. Entre os valores $Z = 3$ e $Z = 4$ não foi possível constatar diferença significativa. Para estes valores, tempos da ordem de 100 segundos foram suficientes em praticamente todas as instâncias. Nem sempre era possível concluir a busca completa dentro deste prazo nas instância maiores, mas, frequentemente, pelo menos um vizinho melhor já era encontrado antes do prazo expirar.

A Tabela 6.18 mostra os resultados do experimento. O critério de parada foi o tempo limite de 2 horas para as instâncias onde a formulação F2 sozinha não encontra valor ótimo. Nas demais instâncias, o limite foi o tempo que a formulação levou para terminar a otimização. Os tempos estão presentes na segunda coluna.

Instância	Tempo Limite(s)	Média	Melhor	Pior	*gap*%
200a	6.4	636	636	636	0.00
300a	155.4	2038.2	2040	2037	0.15
400a	440.7	7217.2	7240	7201	0.54
500a	896.8	14319.7	14332	14287	0.31
600a	7200.0	10107.0	10122	10093	0.29
700a	7200.0	31819.6	31825	31803	0.07
800a	7200.0	38626.1	38635	38618	0.04
900a	7200.0	38766.6	38782	38741	0.11
1000a	7200.0	71855.8	71875	71832	0.06

Tabela 6.18: Resultados do algoritmo que usa a LS_cplex

Um dos resultados que chamou muito a atenção foi a robustez do algoritmo indicada pela pequena diferença entre a melhor e a pior execução (coluna "*gap*%" da tabela). São valores muito pequenos se comparados aos dos métodos heurísticos (EA_ages, GRASP2, ILS1) onde a maioria dos valores ficou acima de 2.0%. De fato, a reiterada execução da busca LS_cplex faz com que o algoritmo produza resultados muito próximos, já que boa parte do espaço de soluções está sendo analisado. Os resultados médios também são

muito bons, mostrando que a inclusão da LS_cplex conseguiu melhorar os resultados do evolutivo EA_ages.

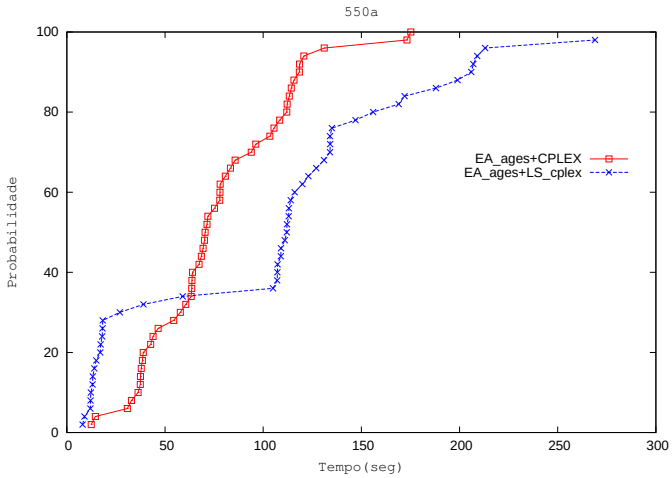
Um segundo experimento foi realizado a fim de verificar o desempenho dos dois métodos híbridos EA_ages +CPLEX e EA_ages +LS_cplex em um curto espaço de tempo. Foi estabelecido um limite de apenas 300 segundos para que os algoritmos possam atingir o alvo. Neste caso, o resultado médio do algoritmo EA_ages (Tabela 6.15, página 80) foi definido como alvo. É importante destacar que naquele experimento o tempo limite era calculado em função do número de tarefas. Três instâncias (550a, 700a e 850a) foram aleatoriamente escolhidas para fins de exibição dos resultados. Foram feitas 50 execuções em cada uma e a Figura 6.3 mostra a análise probabilística do experimento.

A Figura 6.3(a) mostra que os dois métodos híbridos conseguiram alcançar o alvo em todas as execuções, sendo o EA_ages +CPLEX um pouco mais rápido do que o EA_ages +LS_cplex. O desempenho deste está muito atrelado não à qualidade da solução, mas a sua vizinhança. Independente de a solução ser boa ou ruim, se sua vizinhança não contiver uma solução melhor, o tempo computacional do LS_cplex será desperdiçado em vão.

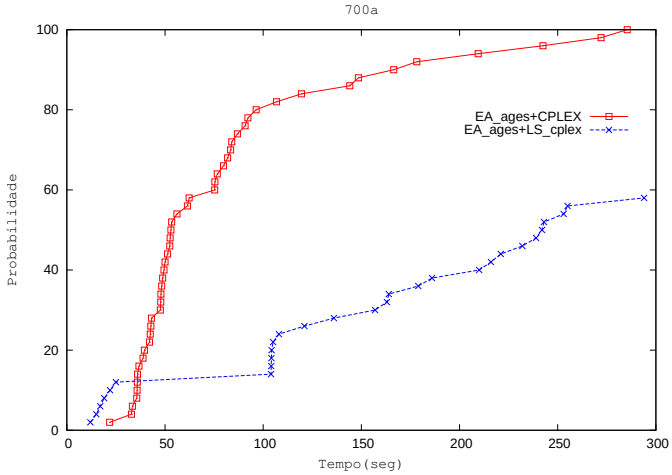
Para uma instância com maior número de nós (700a - Figura 6.3(b)), o EA_ages +LS_cplex começa a perder desempenho, atingindo o alvo em pouco mais de metade das execuções, enquanto o EA_ages +CPLEX tem um declínio de performance menos expressivo. Também é possível perceber um delineamento dos instantes mais prováveis nos quais o EA_ages +LS_cplex pode atingir o alvo.

O primeiro deles é logo no começo da execução, durante a primeira chamada ao LS_cplex. Se o melhor indivíduo da população inicial tiver como vizinho alguém com aptidão melhor ou igual ao alvo, o LS_cplex ou o encontra rapidamente (entre 10 e 15 segundos) ou o encontra ao final de sua primeira execução (logo após os primeiros 100 segundos).

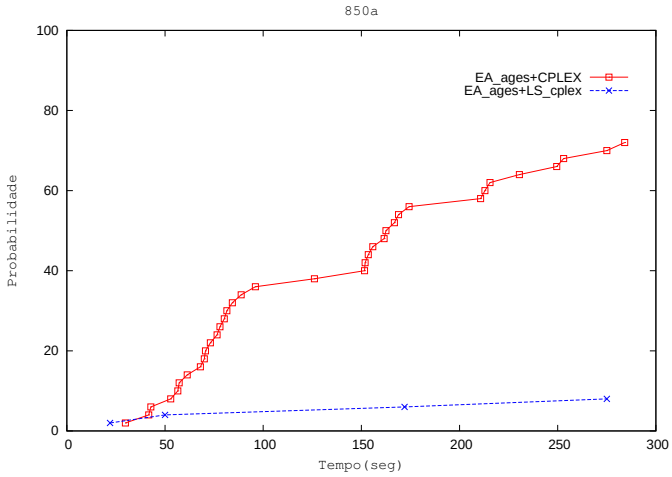
O segundo momento é logo após a segunda execução da busca local, em torno ou um pouco depois de 200 segundos. Como entre uma chamada e outra o evolutivo é retomado e precisa encontrar outro melhor indivíduo para proceder a outra chamada ao LS_cplex, nem sempre é possível terminar uma terceira execução da busca. Se a vizinhança dos



(a)



(b)



(c)

Figura 6.3: Análise probabilística das versões EA_ages +CPLEX e EA_ages +LS_cplex

primeiros indivíduos não for muito promissora, com poucas execuções da busca local realmente não será possível atingir o alvo com a frequência desejada.

A Figura 6.3(c) mostra o resultado para outra instância com um alvo difícil. O limite de 300 segundos é muito menor do que o tempo dado ao EA_ages no experimento que definiu o valor do alvo (naquele caso, 722.5 segundos). Neste contexto, fazer os algoritmos atingirem o resultado médio do EA_ages em menos da metade do tempo pode ser considerado um objetivo difícil.

O desempenho do EA_ages +LS_cplex foi muito ruim exatamente porque ele fica dependente da vizinhança da solução a ele passada. Como o experimento foi projetado para testar a capacidade dos algoritmos em prover boas soluções rapidamente, fica claro que, para isto ocorrer, é necessária uma quantidade maior de tempo. As buscas locais até foram capazes de encontrar vizinhos melhores, mas estes eram apenas ligeiramente superiores às soluções originais, o que acabou não sendo suficiente para que o EA_ages +LS_cplex tivessem performance semelhante ao EA_ages +CPLEX.

Para encerrar os experimentos com os métodos híbridos que utilizam o EA_ages, resolvemos repetir o experimento cujos resultados foram mostrados na Tabela 6.18. Agora, no entanto, serão testados, além do EA_ages +LS_cplex, o EA_ages paralelo com 4 *threads*, o EA_ages +CPLEX e o EA3, algoritmo evolutivo proposto em [27] e melhor heurística conhecida para o PEPRRD antes do começo do presente estudo. O limite de tempo será o mesmo do experimento original (página 88). Os resultados médios de 30 execuções serão apresentados na Tabela 6.19.

Instância	Tempo(seg)	EA3	Paralelo	+LS_cplex	+CPLEX	Melhoria(%)
200a	6.4	604.1	**636.0**	**636.0**	**636.0**	5.3
300a	155.4	1757.8	2002.6	2038.2	**2072.5**	17.9
400a	440.7	5550.2	7062.1	7217.2	**7271.0**	31.0
500a	896.8	11345.2	13933.1	14319.7	**14335.9**	26.4
600a	7200.0	7049.3	9922.9	10107.0	**10157.6**	44.1
700a	7200.0	20982.2	30408.2	31819.6	**31823.9**	51.7
800a	7200.0	30491.9	37324.3	38626.1	**38637.0**	26.7
900a	7200.0	25233.3	37045.5	38766.6	**38768.6**	53.6
1000a	7200.0	60080.5	69602.2	**71855.8**	71843.0	19.6

Tabela 6.19: Resultados dos algoritmos baseado no EA_ages em comparação com o EA3

A última coluna da Tabela 6.19, mostra o quanto a solução média do EA3 foi superada

em relação ao resultado médio do melhor entre os algoritmos propostos. Só na instância 1000a, o EA_ages +CPLEX não obteve a melhor média, o que não impediu que os resultados gerados por ele fossem até 53.6% melhores que os resultados do EA3.

Portanto, analisando-se os resultados do vários testes realizados, pode-se dizer que o algoritmo EA_ages +CPLEX apresenta os melhores resultados dentre os métodos não-determinísticos, tanto em testes com limite de tempo muito pequeno quanto em testes onde o algoritmo pode ser mais longamente executado.

6.4.2 Intervalos de tamanho fixo

Os experimentos com os métodos híbridos, onde o horizonte de planejamento é particionado em intervalos, consistem em executar cada um deles até que todo o escalonamento tenha sido realizado. Embora, a divisão do horizonte de planejamento abrevie bastante o tempo necessário para concluir a otimização em cada intervalo, não é possível prever um tempo máximo para que isto ocorra. Desta forma, será definido um limite de 3 horas (10800 segundos) para que a otimização de cada parte seja concluída. Se o limite for atingido, a solução incumbente será considerada como o resultado deste intervalo e servirá de base para outras eventuais etapas. É importante dizer que estes algoritmos são determinísticos, uma vez que não há variáveis aleatórias envolvidas, e que foi utilizada a formulação F2 com as restrições adicionais. Esta escolha foi feita porque a versão sem as restrições demorava até 30 vezes mais, em média, para terminar as etapas do escalonamento.

O primeiro esquema testado realiza a divisão em intervalos de tamanho K. A Tabela 6.20 (página 97) mostra os resultados finais para diversos valores de K (de 1 a 10). A última coluna apresenta a média dos diversos resultados.

Duas subdivisões com valores de K semelhantes apresentam resultados também semelhantes. Era esperado que um valor de K maior produzisse resultados melhores por abranger uma parte maior do escalonamento. Isto não se confirmou porque a visão de apenas parte do escalonamento pode influenciar na percepção dos algoritmos sobre a importância de uma tarefa para o escalonamento como um todo. Exemplo: uma tarefa pode ser considerada muito ruim se olharmos só para ela e não considerarmos a possibilidade

de ativação de suas sucessoras; porém, se existir tempo suficiente para que também as sucessoras sejam ativadas, a tarefa inicialmente ruim pode ser considerada muito importante. De forma contrária, uma tarefa inicialmente considerada boa, pode ser preterida por outras que venham ser consideradas melhores.

Obviamente que uma diferença muito grande nos valores de K tende a produzir resultados igualmente diferentes. Outra constatação é que o tempo consumido também perde seu caráter crescente conforme aumenta o número de tarefas da instância ou dos intervalos de tempo. Na verdade, cada subdivisão de cada instância vai corresponder uma subproblema novo de dificuldade muito complicada de ser prevista. A Tabela 6.21 (página 98) mostra os tempos totais (em segundos).

6.4.3 Intervalos de tamanho variável

Neste experimento, a divisão do horizonte de planejamento se dará por meio do número de tarefas que podem começar a ser ativadas em cada intervalo. Com isso, poderemos ter algumas subdivisões contemplando poucas unidades de tempo e outras com muitas unidades, desde que o número de tarefas relacionadas seja o mais semelhante possível. A Tabela 6.22 (página 99) mostra o resultado final para escalonamentos particionados de 2 a 10 vezes de acordo com este critério.

Semelhantemente à divisão em intervalos de tempo fixo, os resultados desta segunda estratégia não apresentam um comportamento bem definido do resultado em relação à quantidade de intervalos. Em teoria, quanto menor esta quantidade, maiores serão os intervalos e melhores tendem a ser as soluções. No entanto, estipular um mapeamento exato entre as Tabelas 6.20 e 6.22 é muito complicado porque mesmo em uma instância onde um valor de K corresponda a v intervalos, estes podem agrupar tarefas completamente diferentes. Por exemplo: a instância 400a tem o tamanho do horizonte $H = 20$. A primeira divisão com $K = 10$ produzirá dois escalonamentos: $H_1 = 1..10$ e $H_2 = 11..20$. Já realizando a divisão pelo segundo critério com $v = 2$, poderemos ter dois escalonamentos $H_1 = 1..10$ e $H_2 = 11..20$, ou $H_1 = 1..9$ e $H_2 = 10..20$, ou $H_1 = 1..11$ e $H_2 = 12..20$, ou $H_1 = 1..8$ e $H_2 = 9..20$ ou outro qualquer dependendo da quantidade de tarefas que estão inseridas em cada intervalo. Logo, não é possível fazer uma comparação caso a caso, mas se tomarmos os resultados de um e de outro critério, juntamente com os tempos compu-

tacionais mostrados nas Tabelas 6.21 e 6.23, poderemos verificar que o segundo critério é um pouco mais eficiente.

Tomemos por comparação o caso onde $v = 2$. Esta é subdivisão mais árdua, em teoria, pois temos apenas dois intervalos. Mais difícil do que isto só se considerássemos o horizonte inteiro de uma única vez. As instâncias com mais de 400 tarefas, terão mais de 20 unidades de tempo, já que $H = \sqrt{n}$ nas instâncias testadas. Se $K = 10$, teremos para o primeiro critério pelo menos três subdivisões, sendo a última geralmente menor do que as demais. O tempo computacional gasto, para estas instâncias, pelo primeiro critério foi de 29184.3 segundos; pelo segundo, apenas 25442.0. Os resultados do segundo critério são 4.5% melhores do que os do primeiros critério, sendo que em apenas uma instância (600a) o primeiro critério teve resultado melhor. Se levarmos em conta também a instância 400a, que no caso seria subdividida em duas partes pelos dois critérios, o segundo critério teria desempenho ainda ligeiramente melhor. Assim, o segundo critério com apenas dois intervalos consegue se mostrar mais eficiente do que o primeiro critério com três ou mais intervalos.

A subdivisão por intervalos variáveis tem como principal vantagem, nas instâncias testadas, o fato de que os primeiros intervalos contemplam um número maior de tarefas do que na divisão por intervalos de tamanho fixo. No "início" do grafo, poucas tarefas estão livres para serem ativadas. Assim, para conseguir agrupar o número de tarefas desejado, é preciso incorporar mais unidades de tempo a esses intervalos. Com mais unidades de tempo sendo consideradas inicialmente, melhor o escalonamento parcial naqueles intervalos. Como os demais escalonamentos são baseados nos resultados dos escalonamentos anteriores, melhor tende a ser o resultado final.

Tudo isto, que em teoria faz sentido, provou ser bom também na parte prática. A questão do menor consumo de tempo pode ser explicada de forma semelhante: os escalonamentos iniciais não costumam demorar muito; se forem bem feitos, tendem a oferecer limites primais melhores para os escalonamentos seguintes, que tenderão a terminar mais rapidamente.

6.4.4 Múltiplos intervalos

Como mencionado, esta divisão utiliza o mesmo critério da anterior, porém ao terminar de tratar todo o horizonte de planejamento, os intervalos são alongados ou encurtados e um novo escalonamento é realizado. Este critério, portanto, é uma extensão do anterior, já que se trata de repetidas execuções que abordam a "vizinhança" dos intervalos. De fato, o número execuções é igual a $2v - 1$, onde v é a quantidade desejada de intervalos, pois o último deles termina obrigatoriamente em $t = H$, mas há uma primeira execução idêntica à execução do segundo critério. A Tabela 6.24 (página 101) mostra o resultado para diversos valores de v.

Como esperado, os resultados desta estratégia de particionamento são um pouco melhores do que a anterior - cerca de 2.8%. O tempo computacional, no entanto, é muito maior, não só porque são executadas várias iterações, mas também porque ao se alterar o tamanho dos intervalos, é possível deixá-los mais difíceis de serem resolvidos. A Tabela 6.25 (página 102) mostra os tempos de execução. Pelos valores apresentados, pode-se dizer que este método tem uma relação custo-benefício pior do que o anterior, já que demora muito mais, mesmo quando há grande número de intervalos. Portanto, das três estratégias de particionamento, a segunda - com intervalos de tamanho variável - apresenta os melhores resultados em um espaço de tempo não muito longo.

Um último experimento foi realizado comparando o melhor método híbrido (EA_ages +CPLEX) com o melhor particionamento (em intervalos de tamanho variável). Como os particionamentos são métodos determinísticos, o critério de parada utilizado foi o tempo computacional gasto pelo melhor particionamento (melhor valor de v) obtido em cada instância separadamente. O método híbrido EA_ages +CPLEX foi executado 10 vezes com este tempo limite e a média dos resultados bem como os demais parâmetros do experimento são apresentados na Tabela 6.26 (página 103).

Os resultados mostram que em boa parte das instâncias o método híbrido não conseguiu produzir os mesmos resultados do particionamento, em especial nas instâncias onde o tempo era muito curto. Isto ocorre por dois motivos: (i) as primeiras populações do evolutivo, geralmente, têm aptidão inferior às primeiras soluções geradas pela formulação (com o horizonte completo ou particionado); (ii) a parte exata do EA_ages +CPLEX

demora até vários segundos realizando pré-processamento, principalmente no nó raiz, o que impede que a (primeira) solução passada pelo evolutivo seja analisada e aprimorada pelo CPLEX. Se levarmos em consideração que estas primeiras etapas do método exato são realizadas de forma sequencial, podemos entender que existe também no EA_ages +CPLEX uma situação de início lento, até que evolutivo e CPLEX consigam atingir todo o seu desempenho. A Figura 6.3 (página 90) já mostrava que os métodos híbridos perdiam um pouco do seu desempenho com um limite de tempo curto, conforme as instâncias aumentam de tamanho. A instância 1000a é um ótimo exemplo: em 4 das 10 vezes em que foi executado, a parte exata do EA_ages +CPLEX não teve tempo suficiente para aprimorar a solução passada pelo evolutivo, o que justifica seu fraco desempenho quando comparado ao particionamento.

Duas coisas, porém, precisam ser ditas em defesa do EA_ages +CPLEX. A primeira é quanto ao particionamento utilizado para fins de comparação: o valor de v precisa ser indicado como um parâmetro de execução, ou seja, o utilizador do algoritmo precisa saber *a priori* qual é o valor que produz o melhor escalonamento. Na maioria das instâncias testadas, este valor foi igual a 2, mas isto depende da instância e pode ser necessário tentar dois ou três valores até se obter o melhor resultado. Por outro lado, o método híbrido não precisa de parâmetro externo algum para que possa iniciar sua execução, o que poderia fazer com que ele tivesse um desempenho melhor caso a tentativa do valor de v não seja o melhor possível. O segundo argumento em favor do método híbrido é que a motivação do problema trata da execução de etapas de um projeto que serão ativadas por semanas, meses ou anos. Logo, não há uma necessidade urgente em se produzir um escalonamento em poucos segundos. Em uma situação real, é melhor produzir, em minutos ou poucas horas, um escalonamento de meses ou anos que seja melhor do que gastar dois ou três minutos e ter um escalonamento não tão bom. A menos que haja uma aplicação onde o escalonamento seja necessários em pouco tempo, o algoritmo existente para o PEPRRD com maior potencial parece ser o híbrido EA_ages +CPLEX.

	Tamanho do Intervalo(K)										
Instância	K = 1	2	3	4	5	6	7	8	9	10	Média
50a	47	47	47	47	47	47	47	-	-	-	47.0
100a	304	303	304	303	**304**	**304**	**304**	**304**	**304**	**304**	303.8
150a	575	575	575	**576**	**576**	575	575	**576**	**576**	**576**	575.5
200a	627	632	628	632	**636**	632	635	632	635	**636**	632.5
250a	986	1014	1014	1022	1026	**1030**	**1030**	1025	1021	1026	1019.4
300a	1777	1898	1927	1965	1946	2020	2041	**2043**	2025	1950	1959.2
350a	2271	2325	2423	2449	2421	2432	2487	**2571**	2568	2545	2449.2
400a	5891	5618	6143	6641	6169	**6741**	6732	6656	6460	6158	6320.9
450a	7847	7874	7817	9024	8512	9050	8819	**9505**	9036	8606	8609.0
500a	12251	12689	13239	13006	13433	13397	13408	**13941**	13846	13510	13272.0
550a	9188	9895	9653	9976	9915	10053	10112	10435	**10929**	10733	10088.9
600a	8178	8890	9554	9424	9321	9809	9526	9417	9864	**10096**	9407.9
650a	12023	13511	14042	14855	14959	**15539**	13851	15149	15294	16028	14525.1
700a	22891	25454	28769	28398	28599	28772	29882	**30587**	29919	28824	28209.5
750a	28529	28345	31633	30950	32643	34478	32678	33684	**34548**	32793	32028.1
800a	32384	32889	35127	35614	34118	35683	35300	37007	**37363**	37145	35263.0
850a	35572	38031	38975	41682	42127	41512	40006	**45312**	44687	42239	41014.3
900a	30083	31981	33635	35070	33594	36046	34442	34619	36730	**37529**	34372.9
950a	58762	61539	61931	62393	62599	61707	64512	**65419**	65203	62877	62694.2
1000a	60412	64744	65840	65207	67699	67885	66024	67867	69338	**69717**	66473.3

Tabela 6.20: Resultados para divisão do horizonte de planejamento em intervalos de tamanho fixo

	Tamanho do Intervalo(K)										
Instância	K = 1	2	3	4	5	6	7	8	9	10	Média
50a	0.0	0.0	0.0	0.0	0.0	0.0	0.0	-	-	-	0.0
100a	0.0	0.0	0.0	0.0	0.0	0.1	0.1	0.1	0.1	0.1	0.1
150a	0.0	0.0	0.0	0.1	0.0	0.1	0.1	0.1	0.2	0.2	0.1
200a	0.0	0.0	0.2	0.2	0.3	0.3	0.3	0.4	0.7	0.9	0.3
250a	1.4	0.4	0.3	0.5	0.5	1.1	0.7	0.6	0.6	0.5	0.7
300a	1.7	0.9	0.6	0.7	1.0	1.4	2.6	0.9	1.4	1.8	1.3
350a	1.9	1.1	1.1	1.4	1.4	2.0	4.2	5.1	4.3	1.0	2.4
400a	3.0	1.4	0.9	0.9	2.4	1.8	2.0	3.8	14.1	43.1	7.3
450a	4.2	2.3	1.5	1.8	2.3	6.2	5.8	3.4	4.1	8.1	4.0
500a	4.4	2.5	2.9	5.6	2.6	12.1	9.0	10.5	41.5	2043.7	213.5
550a	5.9	2.8	2.7	2.7	2.7	5.0	35.6	28.3	55.6	270.7	41.2
600a	7.0	3.8	2.9	3.4	11.3	12.7	7.5	514.3	870.5	50.0	148.3
650a	9.1	4.4	4.4	4.5	11.4	16.2	18.2	400.8	201.8	400.6	107.1
700a	11.6	5.3	5.8	5.2	8.5	11.7	31.3	23.0	19.6	22.3	14.4
750a	11.5	7.1	6.0	4.9	5.6	11.7	45.7	13.6	65.6	10910.2	1108.2
800a	15.1	7.5	7.4	16.0	9.2	203.7	164.1	1346.3	2081.9	555.1	440.6
850a	14.9	8.2	7.5	7.9	12.8	28.9	31.4	52.5	644.2	10878.7	1168.7
900a	17.0	9.3	7.7	7.8	22.4	59.9	10805.9	10803.9	7788.6	1729.8	3125.2
950a	20.1	11.5	9.5	17.6	12.7	540.6	38.3	45.0	28.0	572.5	129.6
1000a	23.5	13.0	9.3	12.3	9.7	1158.3	102.1	694.1	22.2	1742.6	378.7

Tabela 6.21: Tempos computacionais para divisão do horizonte de planejamento em intervalos de tamanho fixo

	Quantidade de Intervalos(v)									
Instância	v = 10	9	8	7	6	5	4	3	2	Média
50	47	47	47	47	47	47	47	47	47	47.0
100	**304**	**304**	**304**	**304**	**304**	**304**	303	**304**	303	303.8
150	575	575	575	**576**	**576**	575	575	575	**576**	575.3
200	627	628	628	628	628	628	629	631	**635**	629.1
250	1002	1002	1014	1014	1009	1021	1021	1021	**1025**	1014.3
300	1784	1833	1869	1839	1885	1841	2026	2030	**2042**	1905.4
350	2271	2270	2345	2354	2227	2195	2467	2534	**2567**	2358.9
400	5374	5999	5570	5503	5503	5647	**6641**	6612	6460	5923.2
450	7700	7657	7741	7533	7768	8401	8578	8529	**9505**	8156.9
500	12994	13113	13376	13288	13230	13501	13846	13938	**14225**	13501.2
550	9588	9816	9573	9632	9631	10176	10200	10732	**11054**	10044.7
600	9183	9110	9230	9441	9414	9814	9830	**10090**	10035	9571.9
650	13687	13204	15160	15174	15468	16067	16069	16136	**16198**	15240.3
700	25106	25237	25135	25291	27415	29311	27445	28563	**31329**	27203.6
750	29787	30681	27791	29607	28594	30197	31675	32778	**34336**	30605.1
800	32686	33542	34555	35287	36362	35601	37508	37089	**38511**	35682.3
850	36883	38497	38563	38583	39556	39336	41641	40813	**44693**	39840.6
900	32447	32004	31705	34266	34345	34431	36475	37295	**38684**	34628.0
950	62019	60907	60270	61485	61558	62312	62763	65281	**67529**	62680.4
1000	63840	63796	64821	62294	64253	65332	65779	68297	**70519**	65436.8

Tabela 6.22: Resultados para divisão do horizonte de planejamento em intervalos de tamanho variável

	Quantidade de Intervalos(v)									
Instância	v = 10	9	8	7	6	5	4	3	2	Média
50	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
100	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.2	0.0
150	0.1	0.0	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0
200	0.0	0.0	0.0	0.0	0.1	0.4	0.2	0.2	0.5	0.2
250	0.4	0.4	0.4	0.4	0.5	0.4	0.4	0.4	1.0	0.5
300	1.0	1.0	0.8	0.8	0.6	0.6	1.0	1.5	5.7	1.4
350	1.0	0.9	0.8	0.7	0.8	0.7	0.6	2.4	4.2	1.3
400	1.2	1.0	1.2	1.1	1.0	0.8	0.9	1.3	16.9	2.8
450	2.1	1.9	1.7	1.5	1.3	1.1	1.3	1.2	5.6	2.0
500	3.2	2.8	2.5	3.8	2.2	3.2	3.3	11.0	342.5	41.6
550	3.7	3.3	2.7	2.3	2.1	3.4	5.5	10.5	566.5	66.7
600	3.5	3.2	5.2	3.5	3.9	8.8	20.3	172.4	9910.8	1125.7
650	5.0	4.7	7.1	11.0	44.2	18.0	65.1	148.7	661.3	107.2
700	4.4	4.2	3.7	3.3	3.3	3.2	3.4	7.1	164.7	21.9
750	5.2	4.8	4.0	4.4	3.4	3.7	3.6	12.9	112.6	17.2
800	6.0	5.6	5.5	5.3	18.2	5.6	29.6	87.4	6616.0	753.2
850	6.5	5.8	5.3	4.9	4.5	5.6	34.4	8.2	1335.0	156.7
900	8.6	6.4	6.3	5.9	7.0	5.4	28.1	81.5	4072.4	469.1
950	9.3	8.5	6.7	7.2	7.3	9.2	8.4	9.4	1166.5	136.9
1000	8.5	8.3	7.7	6.9	7.1	6.9	7.8	31.8	488.1	63.7

Tabela 6.23: Tempos computacionais para divisão do horizonte de planejamento em intervalos de tamanho variável

	Quantidade de Intervalos(v)									
Instância	v = 10	9	8	7	6	5	4	3	2	Média
50	47	47	47	47	47	47	47	47	47	47.0
100	304	304	304	304	304	304	304	304	304	304.0
150	576	576	576	576	576	576	576	576	576	576.0
200	**636**	632	632	632	632	632	635	**636**	635	633.6
250	1014	1014	1026	1026	1021	1026	1026	1026	**1030**	1023.2
300	1830	1890	1901	1929	2003	1993	2026	2053	**2061**	1965.1
350	2359	2270	2431	2426	2351	2365	2540	2559	**2571**	2430.2
400	6311	6349	6156	6349	6349	6510	6716	**6741**	6656	6459.7
450	8205	8014	8094	8488	7886	8461	**9115**	8997	9565	8536.1
500	13441	13273	13490	13501	13369	13792	14014	14127	**14239**	13694.0
550	9750	9974	9948	9948	9963	10605	10681	10954	**11070**	10321.4
600	9429	9429	9342	9616	9702	9969	9921	10123	**10167**	9744.2
650	14339	14751	15531	15535	16067	16096	16180	16232	**16272**	15667.0
700	27321	28272	27297	28844	28830	29310	28409	29810	**31567**	28851.1
750	30871	31086	31948	31609	31286	33639	31849	34621	**35555**	32496.0
800	35877	35432	35878	36140	37456	35663	37692	38096	**38579**	36757.0
850	38698	38661	39529	39515	40774	39920	41841	42219	**46093**	40805.6
900	33144	33678	35034	34982	35953	36404	37537	38287	**38793**	35979.1
950	62178	62141	61636	62200	62281	64586	65188	66520	**67840**	63841.1
1000	64284	65532	65295	66010	66933	67589	67551	68301	**71419**	66990.4

Tabela 6.24: Resultados para múltiplos intervalos de tamanho variável

	Quantidade de Intervalos(v)									
Instância	v = 10	9	8	7	6	5	4	3	2	Média
50	0.0	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
100	2.0	1.6	1.4	0.3	0.0	0.0	0.0	0.1	0.1	0.6
150	5.3	5.0	3.5	0.2	0.1	0.1	0.1	0.6	0.2	1.7
200	8.4	7.0	6.5	0.7	1.4	1.6	1.5	1.0	1.1	3.2
250	16.4	15.8	11.9	8.9	6.8	4.6	3.1	2.5	2.9	8.1
300	21.0	17.7	13.5	10.8	7.3	5.3	7.6	8.9	12.2	11.6
350	24.5	20.7	16.5	13.7	10.0	6.9	6.7	10.6	11.7	13.5
400	32.3	27.2	22.0	13.7	9.1	12.2	8.9	8.4	98.5	25.8
450	43.8	34.6	28.8	20.0	15.1	13.7	11.9	6.9	114.7	32.2
500	76.3	53.3	46.2	36.4	24.0	33.9	27.6	358.0	1340.6	221.8
550	66.8	54.2	42.8	29.7	24.4	41.8	70.2	130.1	989.0	161.0
600	73.2	59.4	68.5	58.5	51.1	110.0	205.9	495.3	21043.5	2462.8
650	92.9	85.3	155.7	4271.4	1964.1	2347.3	11696.8	12804.5	1864.3	3920.3
700	84.5	75.3	59.1	51.0	44.0	40.9	264.7	71.7	1069.3	195.6
750	106.1	87.2	73.0	79.2	49.6	111.9	62.3	88.9	4277.6	548.4
800	151.8	117.6	97.2	88.7	195.0	88.0	174.5	5548.7	10840.0	1922.4
850	117.0	95.7	81.4	66.7	60.5	64.6	574.5	165.3	11577.0	1422.5
900	133.9	111.6	108.9	96.2	104.9	84.5	278.6	1485.6	13762.2	1796.3
950	166.2	191.8	110.3	120.0	100.1	161.3	149.3	236.5	13657.4	1654.8
1000	156.8	132.2	115.2	108.4	97.1	106.0	91.1	1283.5	4404.4	721.6

Tabela 6.25: Tempos computacionais para múltiplos intervalos de tamanho variável

Instância	Melhor v	Tempo(seg)	Solução Partic.	Média
250a	2	1.0	**1025**	998.8
300a	2	5.7	2042	**2055.6**
350a	2	4.2	**2567**	2514.4
400a	4	0.9	**6641**	5324.9
450a	2	5.6	**9505**	8669.3
500a	2	342.5	14225	**14328.8**
550a	2	566.5	**11054**	11052.9
600a	3	172.4	**10090**	10082.6
650a	2	661.3	16198	**16243.7**
700a	2	164.7	**31329**	27562.3
750a	2	112.6	**34336**	27369.9
800a	2	6616.0	38511	**38636.9**
850a	2	1335.0	44693	**46788.4**
900a	2	4072.4	38684	**38771.4**
950a	2	1166.5	67529	**67903.5**
1000a	2	488.1	**70519**	62067.9

Tabela 6.26: Comparação do híbrido EA_ages +CPLEX e do particionamento com intervalos de tamanho variável

Capítulo 7

Conclusões

Este trabalho abordou o Problema de Escalonamento de Projetos com Restrições de Recursos Dinâmicos (PEPRRD), através de duas formulações matemáticas, de restrições adicionais que podem ser inseridas nelas, métodos heurísticos e métodos híbridos.

O PEPRRD se diferencia dos demais problemas de escalonamento de projetos, porque as tarefas não só consomem recursos quando são ativadas, mas também são capazes de gerá-los após suas ativações. O problema encontra aplicações práticas em ambientes comerciais ou industriais nos planos de expansão das empresas.

As restrições adicionais, inicialmente projetadas para a formulação original F1, auxiliaram bastante na geração de limites primais e duais mais próximos, em especial as restrições de somas de variáveis, que foram capazes de reduzir o valor das relaxações lineares em mais de 25%, na média. Foi proposta uma nova formulação (F2) que já incorpora os princípios destas restrições através de um outro significado das variáveis binárias. As restrições adicionais adaptadas para F2 também ajudaram a reduzir um pouco o valor das relaxações lineares. Foi mostrado que é possível passar de uma formulação para outra por meio de equações que relacionam as variáveis das duas formulações. Porém, os resultados obtidos por meio da formulação F2 são bem melhores, já que esta contém equações mais simples de serem manipuladas.

Os métodos heurísticos propostos são versões de três meta-heurísticas bastante conhecidas: Algoritmos Evolutivos, *Greedy Randomized Adaptive Search Procedure* (GRASP) e *Iterated Local Search* (ILS). Um novo esquema de representação das soluções do PEPRRD foi proposto. Este esquema é composto por uma lista de prioridades para cada solução. O algoritmo de escalonamento ativa as tarefas que estiverem disponíveis respeitando a

prioridade estabelecida. Desta forma, qualquer lista de prioridade pode dar origem a um escalonamento viável.

O primeiro evolutivo proposto, EA_priority, foi capaz de gerar resultados já bem superiores ao algoritmo EA3, melhor algoritmo heurístico conhecido para o PEPRRD até então. A quantidade de atualizações na melhor solução também aumentou consideravelmente. A versão mais complexa, EA_ages, apresentou características muito interessantes como o critério de parada com gerações extras, a busca local LS_swap, mas principalmente a estratégia de reconstrução de população quando esta for considerada estagnada. Todas estas características se mostraram eficientes. Duas versões paralelas foram propostas para utilizar melhor a capacidade computacional dos processadores com múltiplos núcleos, já bastante comuns atualmente. As versões paralelas conseguiram obter resultados melhores que a versão sequencial em tempo um pouco menor.

A versão mais complexa do GRASP, o GRASP2, obteve resultados melhores do que sua versão básica, GRASP1. Isto foi conseguido com a utilização de uma memória responsável por armazenar soluções de boa qualidade que tenham sido geradas ao longo da execução do algoritmo. De tempos em tempos estas soluções eram recombinadas para verificar se era possível encontrar uma solução ainda melhor.

O algoritmo de recombinação, que foi responsável pela melhora de desempenho na versão GRASP, não conseguiu repetir a boa performance nos algoritmos ILS. As perturbações existentes no ILS fizeram com que as soluções ficassem tão diferentes a ponto de não serem capazes de melhorar com a recombinação. A versão mais simples, ILS1, contendo apenas as perturbações, obteve resultados melhores. É necessário investir mais tempo na confecção de novas perturbações que contribuam mais favoravelmente com o algoritmo de recombinação para que os resultados das versões ILS sejam aprimorados. Experimentos comparando as três melhores versões meta-heurísticas (EA_ages, GRASP2 e ILS1) mostraram que o evolutivo teve desempenho superior ao atingir os alvos estabelecidos. De fato, os evolutivos já vinham sendo usados há bastante tempo, o que propiciou que eles estejam mais "amadurecidos" do que as demais meta-heurísticas.

Os métodos híbridos, que mesclam elementos heurísticos e exatos, foram propostos com a intenção de aproveitar o melhor das duas abordagens. O primeiro método, EA_ages +CPLEX, é composto pelo evolutivo EA_ages e pelo otimizador CPLEX que rodam em

paralelo. Quando uma nova melhor solução é gerada, é passada ao outro algoritmo para que ele também tenha a chance de aprimorá-la. Além disso, o CPLEX fornece ao evolutivo um perfil de ativação. Este perfil corresponde ao número de tarefas que é ativado a cada unidade de tempo, na melhor solução encontrada pelo CPLEX. O perfil ajuda a suprir uma deficiência do escalonamento por prioridades que é ativação irrestrita das tarefas com maior prioridade. Por sua vez, é fornecido ao CPLEX, no começo de sua execução, uma lista com prioridades para a ordem de fixação das variáveis que tiverem valores fracionários em cada nó da árvore do *branch-and-bound*.

O segundo método híbrido também faz uso do otimizador CPLEX, porém como uma busca local na vizinhança de uma solução gerada pelo EA_ages. Uma solução S' é considerada vizinha de S, se para cada tarefa de S' a diferença dos tempos de ativação nas duas soluções for menor ou igual a um número inteiro positivo Z. O CPLEX realiza a busca local dentro de um tempo pré-determinado e devolve ao evolutivo o melhor vizinho encontrado. Este é inserido na população corrente e o evolutivo prossegue. A média dos resultados deste método, chamado de EA_ages +LS_cplex, foram quase tão boas quanto as do EA_ages +CPLEX, com a vantagem de apresentar uma robustez muito maior, pois a diferença entre a melhor e a pior soluções geradas era muito pequena.

Contudo, em um experimento que forneceu um curto limite de tempo para a execução dos algoritmos, o EA_ages +LS_cplex teve um desempenho muito fraco quando a instância tinha grande número de tarefas, devido ao tempo consumido nas buscas locais sem encontrar vizinhos suficientemente melhores. Pode-se dizer que o EA_ages +CPLEX apresentou o melhor desempenho entre os métodos não-determinísticos ainda mais se levarmos em consideração que os cinco processos que compõem o algoritmo (4 *threads* do CPLEX mais um do evolutivo) concorrem por apenas 4 unidades de processamento, reduzindo, assim, seu potencial para 80% do total.

Os últimos métodos híbridos propostos se diferenciam dos demais por serem considerados determinísticos, ou seja, por produzirem sempre o mesmo resultado para a mesma instância de entrada. Outra diferença é que eles utilizam apenas o otimizador CPLEX, mas trabalhando com partições (intervalos) do horizonte de planejamento. Quando um intervalo é executado, as tarefas ativadas até o final do intervalo são fixadas e o intervalo seguinte é executado até que seja completada a análise de todo o horizonte de planeja-

mento.

Para realizar as divisões do horizonte, foram propostas três estratégias. A primeira trabalha com intervalos de tamanho fixo, ou seja, todos os intervalos (à exceção do último) têm obrigatoriamente a mesma quantidade de unidades de tempo. A segunda estratégia observa a quantidade de tarefas que podem iniciar sua ativação em cada unidade de tempo. De acordo com o número de intervalos que o usuário deseja, é calculada uma quantidade média de tarefas em cada intervalo. As consecutivas unidades de tempo serão agrupadas de forma a conter aproximadamente a quantidade média, independentemente do tamanho que venha a ter este intervalo.

A terceira estratégia é uma extensão da segunda. Um primeiro particionamento é definido e executado conforme a estratégia anterior. Logo após, o primeiro intervalo é encurtado e procede-se a uma nova execução. Em seguida, o intervalo é alongado em relação ao tamanho original e outra execução acontece. Passa-se, então, para o segundo intervalo realizando os mesmos dois procedimentos até que o penúltimo intervalo seja considerado. Trata-se, portanto, de repetidas execuções da segunda estratégia, com pequenas diferenças no tamanho dos intervalos.

Os resultados mostraram que a segunda estratégia apresenta melhores resultados médios do que a primeira e em menor tempo. De fato, o desempenho da execução do CPLEX em cada intervalo vai depender muito mais da quantidade de tarefas (e consequentemente de variáveis) a serem analisadas do que de unidades de tempo.

Os resultados médios da terceira estratégia são ainda um pouco melhores do que os da segunda, porém os tempos computacionais crescem significativamente. Fazendo uma comparação da relação custo-benefício da segunda e terceira estratégia, escolhemos a segunda como a mais vantajosa estratégia de particionamento do horizonte de tempo.

Os algoritmos híbridos, em especial o EA_ages +CPLEX, se mostraram mais eficientes do que as versões puramente heurísticas dos Algoritmos Evolutivos. Os métodos com particionamento do horizonte de planejamento são bastante rápidos, mas dependem de boas escolhas para os parâmetros que definem os intervalos de tempo. O algoritmo EA_ages +LS_cplex apresentou boa robustez, oferendo soluções de qualidade sempre muito boa e sem grandes variações, mas para tanto precisam de quantidade de tempo

razoável, da ordem de dezenas de minutos. O EA_ages +CPLEX apresentou resultados também muito bons, com a vantagem de não requer tanto tempo para produzí-los. Por isso, este algoritmo pode ser considerada a melhor opção para se resolver o PEPRRD de forma exata, se houver tempo suficiente, ou de forma heurística.

7.1 Trabalhos futuros

Embora muitos algoritmos tenham sido propostos e estudados neste trabalho, sempre existe a possibilidade de se utilizar uma outra abordagem para os problemas de otimização combinatória. Como várias versões de algoritmo meta-heurísticos se mostram incapazes de superar os resultados do evolutivo EA_ages, pode-se pensar em utilizá-lo mais intensamente em versões paralelas que utilizem diversas máquinas conectadas por uma rede local. Cada máquina (com seus núcleos de processamento) poderia simular uma população com seus mecanismos próprios de reprodução e aprimoramento das soluções com no modelo de ilhas [54, 55].

Existe a possibilidade de se utilizar mais de uma solução para formar a vizinhança que será analisada pela busca local LS_cplex, como em [56, 57]. Com duas ou três soluções distintas, é possível ampliar a vizinhança em torno de soluções consideradas boas. Isto pode melhorar o desempenho do EA_ages +LS_cplex, já que a busca em um espaço maior pode encontrar vizinhos melhores, embora consuma mais tempo computacional.

Outra maneira de utilizar a LS_cplex é juntamente com o particionamento do horizonte de planejamento. A cada intervalo executado, é possível aplicar a busca local antes de passar para o intervalo seguinte. Com isso, espera-se que uma solução parcial de qualidade superior seja melhor aproveitada na próxima etapa. Ainda pensando na abordagem por particionamento, uma forma de torná-la não-determinística é solicitar que o EA_ages desenvolva um escalonamento para cada intervalo. Desta forma, pode ser dado um limite de tempo e o melhor indivíduo gerado até este prazo é considerado como solução parcial, que servirá de base para ativações em intervalos subsequentes.

Na abordagem por formulações matemáticas pode ser útil desenvolver um esquema que permita alterar a prioridade das variáveis a serem fixadas dinamicamente, ou seja, em tempo de execução. Com o decorrer do algoritmo *branch-and-bound*, as variáveis

sofreriam alterações nas suas prioridades de acordo com a solução incumbente. Tarefas que forem consideradas desnecessárias poderiam ter a prioridade diminuída para intensificar as operações na fixação de variáveis que possam gerar resultados melhores, ou ter a prioridade aumentada e seus valores forçados a zero, pois isso implicaria em zerar diversas variáveis de tarefas sucessoras. Trabalhar com dados provenientes da Relaxação Linear é outra forma de fixar partes de uma solução.

Ainda no âmbito das formulações, o Apêndice A introduz novas generalizações para o PEPRRD. Uma delas apresenta um elemento completamente distinto dos demais vistos em escalonamento de projetos e outras incorporam elementos presentes em modelos mais comuns.

APÊNDICE A

Generalizações para o PEPRRD

Uma das diferenças entre o PEPRRD e outros modelos de escalonamento de projetos é que já no final do instante de tempo onde a tarefa é ativada, os lucros começam a ser gerados e continuam até o final do horizonte de planejamento. No entanto, a maioria dos modelos pressupõem que a tarefa tenha uma duração, ou seja, um intervalo de tempo de algumas unidades até que ela seja considerada executada completamente [58, 59]. Pensando nisso, construímos três modelagens matemáticas derivadas daquela presente no Capítulo 3.

A.1 Modelagem com tempo de construção

Nesta modelagem, após se pagar o custo de uma tarefa i, é preciso esperar d_i unidades de tempo até que a tarefa esteja pronta para produzir recursos. A idéia é tornar mais próximo do real o mecanismo de construção de filiais que baseia e motiva o PEPRRD. Na grande maioria dos empreendimentos comerciais e industriais, a instalação de uma filial demanda tempo para que a edificação seja construída ou mesmo reformada para atender às necessidades do negócio em questão. Enquanto esta etapa não é finalizada, não é possível que o negócio entre em funcionamento e, consequentemente, não há geração de dividendos.

A variável binária x_{it} indica se a tarefa i está desativada (valor zero) ou se já teve sua construção iniciada (valor um) até o tempo t. A variável z_i^t indica se a tarefa i está produzindo recursos (valor um) ou não (valor zero), no instante t. As variáveis inteiras Q_t e P_t indicam, respectivamente, a quantidade de recursos disponíveis e o lucro acumulado no instante t.

$$Max \quad Q_H + P_H \tag{A.1}$$

Sujeito à

$$x_{it} \leq x_{it+1} \quad \forall i = 1, .., n \quad \forall t = 1, ..., H-1 \tag{A.2}$$

$$z_i^t \leq z_i^{t+1} \quad \forall i = 1, .., n \quad \forall t = 1, ..., H-1 \tag{A.3}$$

$$z_i^t \leq z_j^{t-1} \quad \forall i = 1, .., n \quad \forall t = 1, ..., H \quad \forall j \in Pred(i) \tag{A.4}$$

$$x_i^t = z_i^{t+d_i} \quad \forall i = 1, .., n \quad \forall t = 1, ..., H-d_i \tag{A.5}$$

$$z_i^t = 0 \quad \forall i = 1, .., n \quad \forall t = 1, ..., d_i \tag{A.6}$$

$$Q_t = Q_{t-1} + P_{t-1} - \sum_{i=1}^{n} c_i(x_{it} - x_{it-1}) \quad \forall t = 1, ..., H \tag{A.7}$$

$$P_t = \sum_{i=1}^{n} p_i z_i^t \quad \forall t = 0, ..., H \tag{A.8}$$

$$x_{i0} = 0 \quad \forall i = 1, ..., n \tag{A.9}$$

$$x_{it}, z_i^t \in \{0, 1\} \quad \forall i = 1, ..., n \quad \forall t = 1, ..., H \tag{A.10}$$

$$Q_t, P_t \in N \quad \forall t = 0, ..., H \tag{A.11}$$

O objetivo do problema continua maximizar os recursos ao final do horizonte de planejamento. As restrições (A.2) e (A.3) indicam que as variáveis binárias mantêm seus estados relativos ao valor um até o final do horizonte de planejamento, a partir do momento que a tarefa i começa a ser construída ou começa a produzir recursos, respectivamente. A precedência entre as tarefas é dada pela variável z_i^t (A.4), já que é exatamente ela que define o término da etapa de construção da tarefa e o início do período de atividade da mesma. As restrições (A.5) definem que a tarefa pode começar a produzir recursos d_i instantes de tempo após o início de sua construção. Durante os primeiros d_i instantes de tempo, não é possível que a tarefa i produza lucros (A.6). Estas restrições são necessárias para que o somatório dos lucros acumulados (A.8) permaneca correto. As restrições (A.7) determinam como a quantidade de recursos varia ao longo do tempo. As restrições (A.9) determinam que todas as tarefas iniciam desativadas. As restrições (A.10) e (A.11) definem os domínios das variáveis.

A Tabela A.1 traz um exemplo de variáveis binárias que atendem à formulação. No caso, a tarefa começou a ser construída no tempo 3, tem tempo de construção de duas

Tempo	1	2	3	4	5	6	7
x	0	0	1	1	1	1	1
z	0	0	0	0	1	1	1

Tabela A.1: Exemplo de valores binários para a formulação com tempo de construção

unidades e começa a produzir lucro, portanto, no instante de tempo 5. Vale reforçar que o custo é pago no instante de tempo 3, mas os recursos só começam a ser produzidos no instante 5.

A.2 Formulação com produção limitada

Outra maneira de interpretar a duração da tarefa proveniente de outros modelos de escalonamento de projetos, é limitar a produção de recursos após a ativação da tarefa. Em vez de permitir que a tarefa produza recursos indefinidamente até o final do horizonte de planejamento, definimos para cada tarefa i uma duração d_i dentro da qual a tarefa pode produzir lucros. Após este prazo, a tarefa deixa de produzir recursos até o final do horizonte de planejamento. A justificativa para este conceito encontra-se nas situações onde a utilização de uma filial se dá de forma temporária. Por exemplo, um *stand* ou loja para um evento específico, durante um período de férias ou próximo a uma data comemorativa. Passada a época em questão não se deseja mais utilizar esta filial. Uma forma simples de modelar isto é torná-la ociosa ou improdutiva. Neste modelo, não é pressuposto um tempo de construção para as tarefas.

É importante destacar que se o modelo contivesse apenas uma tarefa, não haveria diferença entre o final do horizonte de planejamento e o final da duração da tarefa, pois após este instante nenhuma alteração seria possível nas quantidades de lucro acumulado e recurso disponível. Ao adotar uma duração específica para cada tarefa, permitimos que em um mesmo problema de escalonamento sejam tratadas diversas tarefas com durações potencialmente distintas, embora haja um único instante máximo (final do horizonte de planejamento - H) para que este escalonamento seja realizado.

A variável binária x_{it} indica se a tarefa está ativada ou não (valor um ou zero, respectivamente). A variável binária w_i^t indica se a tarefa i estourou o prazo de produção ou não (valor um ou zero, respectivamente). A idéia é fazer com que as variáveis x_{it} não

precisem voltar a valer zero após o período de atividade da tarefa. Para tanto, uma outra variável deve anular o valor igual a um após o determinado período. As demais variáveis têm o mesmo significado das formulações anteriores.

$$Max \quad Q_H + P_H \tag{A.12}$$

Sujeito à

$$x_{it} \leq x_{it+1} \quad \forall i = 1, .., n \quad \forall t = 1, ..., H-1 \tag{A.13}$$

$$x_{it} \leq x_{jt-1} \quad \forall i = 1, .., n \quad \forall t = 2, ..., H \quad \forall j \in Pred(i) \tag{A.14}$$

$$w_i^t \leq w_i^{t+1} \quad \forall i = 1, .., n \quad \forall t = 1, ..., H-1 \tag{A.15}$$

$$x_i^t = w_i^{t+d_i} \quad \forall i = 1, .., n \quad \forall t = 1, ..., H-d_i \tag{A.16}$$

$$Q_t = Q_{t-1} + P_{t-1} - \sum_{i=1}^{n} c_i(x_{it} - x_{it-1}) \quad \forall t = 1, ..., H \tag{A.17}$$

$$P_t = \sum_{i=1}^{n} p_i(x_{it} - w_i^t) \quad \forall t = 0, ..., H \tag{A.18}$$

$$x_{i0} = 0 \quad \forall i = 1, ..., n \tag{A.19}$$

$$x_{it}, w_i^t \in \{0, 1\} \quad \forall i = 1, ..., n \quad \forall t = 1, ..., H \tag{A.20}$$

$$Q_t, P_t \in N \quad \forall t = 0, ..., H \tag{A.21}$$

As variáveis x_{it} e w_i^t têm o mesmo comportamento não decrescente ao longo do horizonte de planejamento (linhas A.13 e A.15). A precedência volta ser garantida pelas variáveis x_{it} (A.14). O impedimento de se contabilizar a produção de recursos é feito com por meio das restrições (A.16) e (A.18). Os recursos disponíveis a cada instante de tempo são dados pelas restrições (A.17). Inicialmente todas as tarefas estão desativadas (A.19) e o domínio das variáveis é dado pelas restrições (A.20) e (A.21). Não é necessário forçar o valor zero para as variáveis w_i^t antes dos primeiros d_i instantes porque na solução ótima estas variáveis serão automaticamente fixadas neste valor. Isto pode ser visto pelas restrições (A.18): se o valor das variáveis w_i^t for igual a um, nos primeiros d_i instantes, haverá uma perda de lucros acumulados. A Tabela A.2 mostra um exemplo de valores para as variáveis. A tarefa em questão começa a produzir recursos no instante 2 e deixa de produzi-los no instante 5, após 3 unidades de tempo de atividade.

Tempo	1	2	3	4	5	6	7
x	0	1	1	1	1	1	1
w	0	0	0	0	1	1	1

Tabela A.2: Exemplo de valores binários para a formulação com produção limitada

A.3 Aquisição de recursos extras

Outra situação muito comum no meio comercial é a aquisição de capital (recursos) por meios que vão além do foco principal do negócio. Uma das formas mais comuns de se conseguir este capital extra pode ser feito através do levantamento de empréstimos. Um montante de recursos é obtido de forma única e pago em parcelas nas unidades de tempo subsequentes. Na nossa proposta de formulação o montante L adquirido será pago em Y parcelas com valores R_y (possivelmente distintos). A variável binária l_t indica se o empréstimo será tomado ou não no tempo t. A variável binária r_t^y indica se a y-ésima parcela do empréstimo será paga na tempo t. As parcelas sempre serão pagas nos instante de tempos subsequentes à tomada do empréstimo e devem ser todas pagas antes do final do horizonte de planejamento. Por isso, o empréstimo só poderá ser tomado até o instante $H - Y$.

$$Max \quad Q_H + P_H \tag{A.22}$$

Sujeito à

$$x_{it} \leq x_{it+1} \quad \forall i = 1, .., n \quad \forall t = 1, ..., H-1 \tag{A.23}$$

$$x_{it} \leq x_{jt-1} \quad \forall i = 1, .., n \quad \forall t = 2, ..., H \quad \forall j \in Pred(i) \tag{A.24}$$

$$Q_t = Q_{t-1} + P_{t-1} - \sum_{i=1}^{n} c_i(x_{it} - x_{it-1}) + L.l_t - \sum_{y=1}^{Y} r_t^y R_y \quad \forall t \in H \tag{A.25}$$

$$P_t = \sum_{i=1}^{n} p_i x_{it} \quad \forall t = 0, ..., H \tag{A.26}$$

$$\sum_{t=1}^{H-Y} l_t \leq 1 \tag{A.27}$$

$$\sum_{t=H-Y+1}^{H} l_t = 0 \tag{A.28}$$

$$r_t^{t+y} = l_t \quad \forall t = 1, ..., H \quad \forall \{y | 1 \leq y \leq Y, t+y \leq H\} \tag{A.29}$$

$$x_{i0} = 0 \quad \forall i = 1, ..., n \tag{A.30}$$

$$x_{it}, r_t^y, l_t \in \{0, 1\} \quad \forall i = 1, ..., n \quad \forall t = 1, ..., H \quad \forall y = 1, ..., Y \tag{A.31}$$

$$Q_t, P_t, R_y, L \in N \quad \forall t = 0, ..., H \quad \forall y = 1, ..., Y \tag{A.32}$$

As restrições (A.23), (A.24), (A.26) e (A.30) têm o mesmo significado das formulações anteriores. As restrições (A.25) têm, além dos componentes tradicionais, um termo extra que indica que o empréstimo deve ser contabilizado no tempo t por meio do produto $L.l_t$. Somente em um único instante de tempo do horizonte este termo será levado em conta, já que o empréstimo deve ser único como indicado na restrição (A.27). O outro termo presente refere-se ao pagamento de um parcela R_y entre as Y parcelas possíveis. As restrições (A.28) impedem que o empréstimo seja obtido depois do tempo limite $H - Y$, já que ele não será completamente quitado. A relação entre ad variáveis que indicam a obtenção do empréstimo e as variáveis que controlam seu pagamento ocorre por meio das restrições (A.29). As demais restrições definem o domínio das variáveis. A Tabela A.3 mostra um exemplo de um empréstimo adquirido no instante 3 e pago em 4 parcelas nos tempos subsequentes. Vale notar que não é obrigatório forçar o valor igual a zero nas variáveis binárias r_t^y que se referem a tempos onde não haverá pagamento de parcelas,

Tempo	l_t	r_t^1	r_t^2	r_t^3	r_t^4
1	0	0	0	0	0
2	0	0	0	0	0
3	1	0	0	0	0
4	0	1	0	0	0
5	0	0	1	0	0
6	0	0	0	1	0
7	0	0	0	0	1
8	0	0	0	0	0

Tabela A.3: Exemplo de valores binários para a formulação com aquisição de recursos extras

como as variáveis r_1^3 e r_6^1, por exemplo. É óbvio que a terceira parcela nunca poderá ser paga no primeiro instante de tempo, o mesmo para a primeira parcela sendo paga em $t = 6$, já que neste último caso, o empréstimo não será totalmente pago. Na solução ótima, estas variáveis terão valor igual a zero porque elas indicam perda de recursos por causa do pagamento da parcela em questão, parcela esta que seria paga desnecessariamente nestes casos.

Referências

[1] ALVAREZ-VALDES, R. et al. A scatter search algorithm for project scheduling under partially renewable resources. *Journal of Heuristics*, v. 12, p. 95–113, 2006.

[2] BöTTCHER, J. et al. Project scheduling under partially renewable resource constraints. *Management Science*, v. 45, n. 2, p. 543–559, 1999.

[3] BOULY, H. et al. Solving RCPSP with resources production possibility by tasks. In: *Méthodologies et Heuristiques pour l'Optimisation des Systèmes Industriels (MHOSI 2005)*. Hammamet (Tunísia): [s.n.], 2005.

[4] BOULEIMEN, K.; LECOCQ, H. A new efficient simulated annealing algorithm for the resource-constrained project scheduling problem and its multiple mode version. *European Journal of Operational Research*, v. 149, p. 268–281, 2003.

[5] DAMAKA, N. et al. Differential evolution for solving multi-mode resource-constrained project scheduling problems. *Computers & Operations Research*, v. 36, p. 2653–2659, 2009.

[6] MLADENOVIĆ, N. et al. The p-median problem: A survey of metaheuristic approaches. *European J Operational Research*, v. 179, n. 3, p. 927–939, 2007.

[7] ROSING, K. E.; HODGSON, M. J. Heuristic concentration for the p-median: an example demonstrating how and why it works. *Computers & Operations Research*, v. 29, n. 10, p. 1317–1330, 2002.

[8] LIM, G.; REESE, J.; HOLDER, A. Fast and robust techniques for the euclidean p-median problem with uniform weights. *Computers & Industrial Engineering*, v. 57, n. 3, p. 896–905, 2009.

[9] BLAZEWICZ, J.; LENSTRA, J.; KAN, A. Scheduling projects to resource constraints: Classification and complexity. *Discrete Applied Mathematics*, v. 5, p. 11–24, 1983.

[10] DEMEULEMEESTER, E.; HERROELEN, W. A branch-and-bound procedure for multiple resource-constrained project scheduling problem. *Management Science*, v. 38, p. 1803–1818, 1992.

[11] MINGOZI, A. et al. An exact algorithm for project scheduling with resource constraints based on a new mathematical formulation. *Management Science*, v. 44, p. 714–729, 1998.

[12] PATTERSON, J. et al.

[13] BOCTOR, F. F. Some efficient multi-heuristic procedures for resourceconstrained project scheduling. *European Journal of Operational Research*, v. 49, p. 3–13, 1990.

[14] KOLISCH, R. Efficient priority rules for the resource-constrained project scheduling problem. *Journal of Operations Management*, v. 14, p. 179–192, 1996.

[15] VALLS, V.; BALLESTíN, F.; QUINTANILLA, S. A hybrid genetic algorithm for the resource-constrained project scheduling problem. *European Journal of Operational Research*, v. 185, p. 495–508, 2008.

[16] NONOBE, K.; IBARAKI, T. Formulation and tabu search algorithm for the resource constrained project scheduling problem. 2002.

[17] ALCARAZ, J.; MAROTO, C. A robust genetic algorithm for resource allocation in project scheduling. *Annals of Operations Research*, v. 102, p. 83–109, 2001.

[18] ZHU, G.; BARD, J.; TU, G. A branch-and-cut procedure for the multimode resource-constrained project-scheduling problem. *Journal on Computing*, v. 18, n. 3, p. 377–390, 2006.

[19] GONÇALVES, J.; MENDES, J.; RESENDE, M. A random key based genetic algorithm for the resource constrained project scheduling problems. *International Journal of Production Research*, v. 36, p. 92–109, 2009.

[20] SNYDER, L. V.; DASKIN, M. S. A random-key genetic algorithm for the generalized traveling salesman problem. *European Journal of Operational Research*, v. 174, n. 1, p. 38–53, 2006.

[21] SAMANLIOGLU, F.; JR., W. G. F.; KURZ, M. E. A memetic random-key genetic algorithm for a symmetric multi-objective traveling salesman problem. *Computers & Industrial Engineering*, v. 55, n. 2, p. 439–449, 2009.

[22] SILVA, A. R. V.; OCHI, L. S. A dynamic resource constrained task scheduling problem. In: *Proc. of Latin-Ibero-American Congress on Operations Research (CLAIO)*. Montevideo (Uruguai): [s.n.], 2006.

[23] FEO, T.; RESENDE, M. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, v. 6, p. 109–133, 1995.

[24] SILVA, A. R. V.; OCHI, L. S. Effective grasp for the dynamic resource-constrained task scheduling problem. In: *Proc. of International Network Optimization Conference (INOC)*. Spa (Bélgica): [s.n.], 2007.

[25] FESTA, P.; RESENDE, M. GRASP: An annotated bibliography. In: RIBEIRO, C.; HANSEN, P. (Ed.). *Essays and Surveys on Metaheuristics*. [S.l.]: Kluwer Academic Publishers, 2002. p. 325–367.

[26] AIEX, R. et al. GRASP with path relinking for three-index assignment. *INFORMS Journal on Computing*, v. 17, n. 2, p. 224–247, 2005.

[27] SILVA, A. R. V.; OCHI, L. S. A hybrid evolutionary algorithm for the dynamic resource constrained task scheduling problem. In: *Proc. of the International Workshop on Nature Inspired Distributed Computing (NIDISC'07)*. LongBeach (EUA): [s.n.], 2007.

[28] CPLEX software web page `http://www.ilog.com/products/cplex`.

[29] LIU, S.; WANG, C. Resource-constrained construction project scheduling model for profit maximization considering cash flow. *Automation in Construction*, v. 17, p. 966–974, 2008.

[30] MARTELLO, S.; TOTH, P. *Knapsack Problems: Algorithms and Computer Implementations*. [S.l.]: John Wiley & Sons.

[31] PISINGER, D. Where are the hard knapsack problems? *Computers & Operations Research*, v. 32, n. 9, p. 2271–2284, 2005.

[32] SILVA, C. G. da; CLíMACO, J.; FIGUEIRA, J. R. Core problems in bi-criteria 0,1-knapsack problems. *Computers & Operations Research*, v. 35, n. 7, p. 2292–2306, 2008.

[33] HOLLAND, J. H. *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI: [s.n.], 1975.

[34] SILVA, A. R. V.; OCHI, L. S.; SANTOS, H. G. New effective algorithm for dynamic resource constrained project scheduling problem. In: *Proc. of International Conference on Engineering Optimization (ENGOPT)*. Rio de Janeiro (Brasil): [s.n.], 2008.

[35] SILVA, A. R. V.; OCHI, L. S. New sequential and parallel algorithm for dynamic resource constrained project scheduling problem. In: *Proc. of the International Workshop on Nature Inspired Distributed Computing (NIDISC'09)*. Roma (Itália): [s.n.], 2009.

[36] CARICATO, P. et al. Parallel tabu search for a pickup and delivery problem under track contention. *Parallel Computing*, v. 29, n. 5, p. 631–639, 2003.

[37] JAMES, T.; REGO, C.; GLOVER, F. A cooperative parallel tabu search algorithm for the quadratic assignment problem. *European J. of Operational Research*, v. 195, n. 3, p. 810–826, 2009.

[38] RESENDE, M.; RIBEIRO, C. Grasp with path-relinking: Recent advances and applications. In: IBARAKI, T.; NONOBE, K.; YAGIURA, M. (Ed.). *Metaheuristics: Progress as Real Problem Solvers*. [S.l.]: Springer, 2005. p. 29–63.

[39] RIBEIRO, C. C.; ROSSETI, I. Efficient parallel cooperative implementations of grasp heuristics. *Parallel Computing*, v. 33, n. 1, p. 21–35, 2007.

[40] ARMENTANO, V. A.; FILHO, M. F. de F. Minimizing total tardiness in parallel machine scheduling with setup times: An adaptive memory-based grasp approach. *European J. of Operational Research*, v. 183, n. 1, p. 100–114, 2007.

[41] LAURENT, B.; HAO, J.-K. Iterated local search for the multiple depot vehicle scheduling problem. *Computers & Industrial Engineering*, v. 57, p. 277–286, 2009.

[42] CORDEAU, J.; LAPORTE, G.; PASIN, F. An iterated local search heuristic for the logistics network design problem with single assignment. *Int. J. Production Economics*, v. 113, p. 626–640, 2008.

[43] VANSTEENWEGEN, P. et al. Iterated local search for the team orienteering problem with time windows. *Computers & Operations Research*, v. 36, p. 3281–3290, 2009.

[44] LIN, S.-W. et al. Applying hybrid meta-heuristics for capacitated vehicle routing problem. *Expert Systems with Applications*, v. 36, n. 2, p. 1505–1512, 2009.

[45] POORZAHEDY, H.; ROUHANI, O. M. Hybrid meta-heuristic algorithms for solving network design problem. *European J, of Operational Research*, v. 182, n. 2, p. 578–596, 2007.

[46] HOPPER, E.; TURTON, B. C. H. An empirical investigation of meta-heuristic and heuristic algorithms for a 2d packing problem. *European J, of Operational Research*, v. 128, n. 1, p. 34–57, 2001.

[47] BERTACCO, L.; FISCHETTI, M.; LODI, A. A feasibility pump heuristic for general mixed-integer problems. *Discrete Optimization*, n. 1.

[48] FISCHETTI, M.; GLOVER, F.; LODI, A. The feasibility pump. *Mathematical Programming*, v. 104, n. 1, p. 91–104, 2005.

[49] FISCHETTI, M.; LODI, A. Local branching. *Mathematical Programming*, v. 98, p. 23–47, 2003.

[50] FISCHETTI, M.; LODI, A. Repairing mip infeasibility through local branching. *Computers & Operations Research*, v. 35, n. 5, p. 1436–1445, 2008.

[51] LABIC Project web page `http://www.ic.uff.br/~labic`.

[52] BURIOL, L. et al. A hybrid genetic algorithm for the weight setting problem in ospf/is-is routing. *Networks*, v. 46, n. 1, p. 36–56, 2005.

[53] AIEX, R.; RESENDE, M.; RIBEIRO, C. Probability distribution of solution time in grasp: An experimental investigation. *Journal of Heuristics*, v. 8, p. 343–373, 2002.

[54] ALBA, E.; TOMASSINI, M. Parallelism and evolutionary algorithms. *IEEE Trans. Evolutionary Comput.*, v. 5, p. 443–462, 2002.

[55] GUSTAFSON, S.; BURKE, E. K. The speciating island model:an alternative parallel evolutionary algorithm. *J. Parallel Distrib. Comput.*, v. 66, p. 1025–1036, 2006.

[56] PIGATTI, A.; ARAGÃO, M. P. d.; UCHOA, E. Stabilized branch-and-cut-and-price for the generalized assigment problem. In: *Proc. of the 2nd Brazilian Symposium on Graphs, Algorithms and Combinatorics*. Angra dos Reis: [s.n.], 2005.

[57] SANTOS, H. G. *Formulações e Algoritmos para o Problema de Programação de Horários em Escolas*. Tese (Doutorado) — Universidade Federal Fluminense, 2007.

[58] MIKA, M.; WALIGÓRA, G.; WEZGLARZ, J. Simulated annealing and tabu search for multi-mode resource-constrained project scheduling with positive discounted cash flows and different payment models. *European Journal of Operational Research*, v. 164, n. 3, p. 639–668, 2005.

[59] KE, H.; LIUA, B. Project scheduling problemnext term with stochastic activity duration times. *Applied Mathematics and Computation*, v. 168, n. 1, p. 342–353, 2005.