Curso de Ciência da Computação
Universidade Estadual de Mato Grosso do Sul

Aplicação dos métodos GRASP e BRKGA para um problema de otimização de portfólios

Bruno Yoichi Tanaka

Prof. Doutor Cleber Valgas Gomes Mira (Orientador)

Dourados - MS
2018

Aplicação dos métodos GRASP e BRKGA para um problema de otimização de portfólios

Bruno Yoichi Tanaka

Este exemplar corresponde à redação final da monografia da disciplina Projeto Final de Curso devidamente corrigida e defendida por Bruno Yoichi Tanaka e aprovada pela Banca Examinadora, como parte dos requisitos para a obtenção do título de Bacharel em Ciência da Computação.

Dourados, 21 de novembro de 2018

Prof. Doutor Cleber Valgas Gomes Mira (Orientador)

Curso de Ciência da Computação
Universidade Estadual de Mato Grosso do Sul

Aplicação dos métodos GRASP e BRKGA para um problema de otimização de portfólios

Bruno Yoichi Tanaka

Novembro de 2018

Banca Examinadora:

Prof. Dr. Cleber Valgas Gomes Mira (Orientador)
Área de Computação - UEMS

Prof. Dr. Fabrício Sérgio de Paula
Área de Computação - UEMS

Prof. Dr. Evandro Cesar Bracht
Área de Computação - UEMS

AGRADECIMENTOS

Agradeço aos meus pais que me proporcionaram a oportunidade de ingressar no ensino superior, familiares e amigos que sempre me apoiaram.

Agradeço ao meu Orientador Professor Doutor Cleber Valgas Gomes Mira que me ofereceu essa oportunidade, esta que com certeza me proporcionou um diferencial acadêmico.

Agradeço muito pelo o que fez por mim, principalmente por não ter me deixado desistir e incentivando para determinar este trabalho e sempre dizendo que eu conseguiria.

Muito obrigado!

RESUMO

Neste trabalho é apresentado uma versão específica do problema de otimização de portfólios de projetos baseado em um problema real de uma companhia de geração de energia elétrica brasileira (MIRA et al., 2015). Esse tipo de problema é conhecido como problema de agendamento de portfólio de projetos, *Project Portfolio Selection* (PPS) que consiste em agendar os projetos da melhor forma seguindo as restrições estabelecidas. O objetivo desse trabalho é desenvolver e implementar os métodos GRASP (*Greedy Randomized Adaptive Search Procedure*) e BRKGA (*Biased Random-Key Genetic Algorithm*) para encontrar soluções razoáveis para o PPS, comparar os resultados encontrados em experimentos e identificar qual método apresenta resultados melhores. Os experimentos foram executados em uma mesma máquina e com os mesmos arquivos de entrada gerados aleatoriamente, mas com base em parâmetros de um cenário real. Os resultados sugerem que o GRASP consegue encontrar melhores soluções e em menor tempo de execução do que o BRKGA.

Palavras-chave: PPS, Portfólio de projeto, Agendamento de projetos, BRKGA, GRASP.

ABRSTRACT

In this work, a specific version of the problem of optimization of project portfolios based on a real problem of a Brazilian electric power generation company (MIRA et al., 2015) is presented. This type of problem is known as project portfolio scheduling problem, or Project Portfolio Selection (PPS) problem, which consists of optimizing projects according to established constraints. The objective of this work is to develop and implement a GRASP (Greedy Randomized AdaptiveSearch Procedure) and a BRKGA (Biased Random-Key Genetic Algorithm) methods to find reasonable solutions for PPS, to compare the results found in experiments, and to identify which method presents better results. The experiments were run on the same machine and with the same input files generated randomly, but based on parameters of a real scenario. The results suggest that GRASP can find better solutions and less execution time than BRKGA.

Keywords: PPS, Project portfolio, Project scheduling, BRKGA, GRASP.

SUMÁRIO

LISTA DE ILUSTRAÇÕES

LISTA DE ALGORITMOS

LISTA DE ABREVIATURAS E SIGLAS

PPS	*Project Portfolio Selection*
BRKGA	*Biased Random-Key Genetic Algorithm*
GRASP	*Greedy Randomized Adaptive Search Procedure*
PH	*Planning Horizon*
OPEX	*Operational Expenditures*
CAPEX	*Capital Expenditures*
HRSP	*High-risk Sorted Pairs*
SP	*Sorted Pairs*

1 INTRODUÇÃO

O problema estudado neste trabalho é o agendamento de portfólio projetos do inglês *Project Portfolio Selection* (PPS). Esse problema consiste em dado um conjunto de projetos com certas propriedades (custos, riscos, etc.), encontrar o melhor agendamento possível dos projetos sem violar certas restrições.

Segundo Montes (MONTES, 2017), um portfólio de projetos é o estabelecimento dos projetos de uma empresa de forma que melhore o gerenciamento dos recursos compartilhados e aumente o desempenho dos resultados.

Esse é um problema clássico de otimização encontrado na área de finanças, pesquisa e desenvolvimento, desenvolvimento de software, entre outros (MIRA et al., 2015). Ele é um problema NP-difícil (DOERNER et al., 2004), isso significa que não se conhece um algoritmo que resolve esse problema em tempo polinomial.

No trabalho de Albuquerque é feito um estudo do problema na área financeira (ALBUQUERQUE, 2009). Nesse trabalho o problema está relacionado ao mercado de ações, onde o investidor procura aplicar seu dinheiro de forma a obter um valor próximo do esperado.

Em outro trabalho, Ribeiro (RIBEIRO; ALVES, 2017) discute a distribuição de bolsas em instituições de ensino. O problema consiste em controlar a melhor distribuição de bolsas de incentivo a pesquisas científicas de forma a evitar gastos e maximizar a distribuição.

Em um trabalho anterior foi apresentado uma versão do problema para gestão de risco e a solução utilizada foi baseada em uma meta-heurística GRASP (*Greedy Randomized Adaptive Search Procedure*) (MIRA et al., 2015). Antes de ser apresentada essa solução, o agendamento era feito manualmente por especialistas. Foi decidido o uso desse método devido a sua fácil compreensão e rápida implementação, e não precisar atender a certas exigências de modelos matemáticos de otimização; como, por exemplo, restrições lineares.

O problema PPS estudado neste trabalho é baseado em Mira e colaboradores (MIRA et al., 2015). Da mesma forma que foi utilizado o GRASP para resolver esse problema PPS, também foi utilizado um outro método meta-heurístico, o BRKGA (*Biased Random-Key Genetic Algorithm*). Ambos os métodos GRASP e BRKGA foram implementados e testados.

1.1 Métodos Heurísticos e Meta-heurísticas

Os métodos heurísticos são técnicas utilizadas para resolver problemas mais rapidamente que outros métodos clássicos, como força bruta, programação linear, programação dinâmica, *branch and bound*, entre outros, ou encontrar uma solução aproximada quando outros métodos clássicos falham (GENDREAU; POTVIN, 2010).

As meta-heurísticas são métodos utilizados para coordenar a execução de heurísticas simples, com a finalidade de encontrar melhores soluções do que as soluções encontradas utilizando apenas as heurísticas simples (GENDREAU; POTVIN, 2010). Alguns exemplos mais conhecidos de meta-heurísticas são: GRASP, *Tabu Search*, *Hill Climbing*, *Simulated Annealing*, etc.

O GRASP (FESTA; RESENDE, 2002) é um método heurístico guloso e *multi-start*, ou seja, possui vários recomeços (MIRA et al., 2015). Ele é composto por:

- Uma fase de construção que constrói uma solução válida inicial de maneira gulosa.
- Uma busca local que procura por soluções melhores a partir da solução inicial obtida da fase de construção. A busca é feita pela vizinhança da solução inicial.

Esses passos são repetidos até que não se encontre uma solução melhor ou um número de vezes definido seja atingido (p.e. mil vezes).

O BRKGA (MENDES; GONÇALVES; RESENDE, 2009) é um algoritmo genético que a partir de uma população inicial faz uma seleção dos melhores indivíduos para serem adicionados a uma nova população. Cada nova população gerada é composta por indivíduos selecionados da população anterior, novos indivíduos gerados aleatoriamente (mutantes) e indivíduos gerados a partir de um cruzamento enviesado. Cada indivíduo possui um valor de *fitness* que representa a sua qualidade como solução. A geração de novas populações é feita até que um critério de parada determinado seja satisfeito.

Os métodos GRASP e BRKGA foram utilizados para resolver diversos problemas de otimização.

Na tese de doutorado (MELO, 2014) foi utilizado o GRASP para resolver uma generalização do problema de programação de tarefas. O método foi utilizado com intuito de minimizar os atrasos das tarefas.

Na tese de mestrado (MEGALE, 2015) é utilizado o BRKGA para resolver o problema de clusterização de módulo de software. Clusterização de módulo de software é o agrupamento dos arquivos de um software com o objetivo de facilitar o entendimento do software.

Já na tese de doutorado (MAINIERI, 2014) foi utilizado o BRKGA em um ambiente *flowshop híbrido*. O *Flowshop híbrido* é um ambiente de trabalho sequencial em estágios, em que cada estágio deve ser executado apenas uma vez e por uma máquina. O método foi utilizado para minimizar o atraso de todos os trabalhos.

1.2 Objetivo do projeto

O objetivo geral do projeto é desenvolver/implementar os métodos GRASP e BRKGA. A partir dos resultados gerados compará-los e determinar qual resolve o PPS apresentado de forma mais eficiente.

1.3 Organização do texto

No Capítulo 2 é explicada detalhadamente a versão do problema estudado e na suas seções as características das instância de entrada.

No Capítulo 3 é apresentado como foi feito o estudo e o desenvolvimento dos métodos utilizados para resolver o problema.

O Capítulo 4 discute os parâmetros utilizados nas heurísticas, a geração das instâncias de entrada do problema, como foram feitos os experimentos, os resultados obtidos e a análise desses resultados.

O último Capítulo 5 sumariza brevemente o problema estudado, a geração dos experimentos, a análise dos resultados, e as novas abordagens para futuros trabalhos.

2 REVISÃO DA LITERATURA

Neste capítulo será explicado com mais detalhes o problema estudado e os algoritmos que serão utilizados para resolvê-lo. Na Seção 2.1 é apresentado com detalhes as características do problema, o formato de entrada para implementação, restrições e função objetivo. A Seção 2.2 discute o método GRASP e como modelar o GRASP para resolver o problema PPS. A Seção 2.3 apresenta o algoritmo BRKGA e discute como modelar uma solução de PPS em um indivíduo da população.

2.1 Otimização de portfólio de projetos

Vamos apresentar uma versão específica do problema de otimização de portfólios baseado em um problema real de uma companhia de geração de energia elétrica brasileira (MIRA et al., 2015). As informações discutidas abaixo são referentes a essa companhia.

Os projetos devem alcançar vários objetivos como:

- Cumprir com a regulamentação do governo.
- Minimizar risco de exposição.
- Aumentar o retorno financeiro.
- Reduzir o custo.

Existem dois tipos de projetos:

1. Projetos de gerenciamento de risco: são projetos responsáveis por controlar riscos que possam surgir.
2. Projetos de gerenciamento de não-risco: são projetos que não ajudam a controlar riscos. São projetos responsáveis pelos suprimentos de escritório, administração, leis, regulamentação, etc. Esses projetos devem seguir estritamente seu agendamento pré-definido.

Os projetos devem ser agendados em um mês no período de tempo chamado de *Planning Horizon* (PH), ou seja, o horizonte de planejamento. Os projetos de gerenciamento de não-risco são obrigatórios, o que significa que devem iniciar no mês pré-definido no PH. Já os projetos de gerenciamento de risco são não obrigatórios, ou seja, seu agendamento pode ser alterado conforme o necessário.

Devido a complexidade dos processos e da operação da empresa, vários eventos indesejáveis podem surgir durante a geração de energia, por exemplo falha de equipamentos,

violação de regulamentos, acidentes, etc. E para evitar/controlar esses eventos indesejáveis é preciso seguir os seguintes passos:

1. Eleger *pontos de atenção* que servirão para identificar possíveis eventos indesejáveis que possam ocorrer. Pontos de atenção também descrevem o tipo de medida que o especialista deve utilizar para evitar o evento.

2. Atribuir um *risco* a cada ponto de atenção. O risco para cada ponto de atenção é baseado em um critério. O valor é um número real proporcional ao impacto que os eventos indesejáveis possam causar. Os pontos de atenção são classificados em dois tipos segundo o nível de risco: crítico e não-crítico. O nível crítico engloba os pontos de atenção intoleráveis, enquanto o nível não-crítico trata dos pontos de atenção com risco tolerável ou moderado.

3. Criar projetos que controlam pontos de atenção para evitar eventos indesejáveis. Os pontos de atenção são controlados por um grupo de projetos que contribuem igualmente para o controle do seu risco. O total de risco controlado por um projeto é a soma das porções de risco que ele contribui para controlar em cada ponto de atenção. Por exemplo, considere um ponto de atenção com risco 100 que é controlado pelos projetos 1, 2, 3 e 4. Então o risco controlado por cada projeto é 25. Caso o projeto 4 controle outro ponto de atenção de risco 30, então o total de risco controlado pelo projeto 4 é 55.

Os pontos de atenção intoleráveis devem ser controlados completamente em um determinado prazo no PH. Apenas os pontos de atenção intoleráveis possuem um prazo limite. Por exemplo, se um ponto de atenção intolerável possui um prazo limite de 10 meses, isso significa que ele deve ser controlado em até 10 meses do PH.

A geração de energia é organizada por localização. Cada localização deve gerar uma quantidade miníma para cumprir com a regulamentação do governo. A quantidade de energia gerada por localização é feita pela soma das energias geradas pelas usinas elétricas.

Cada usina elétrica encontra-se em uma localização específica e, além disso, estão associada a certas divisões administrativas. Essas associações possuem influência na operação das usinas elétricas e refletem nas restrições envolvendo manutenções de unidade geradoras.

Cada usina elétrica possui um número de unidades geradoras. Essas unidades geradoras podem eventualmente parar por várias razões, como clima, falta de mão-de-obra, falhas da unidade, manutenção, etc. Os projetos de manutenção que provocam paradas de unidades geradoras especificam um mês inicial de parada e duração de parada. A execução de projetos de manutenção podem ser afetada pelas restrições de parada. Por exemplo, se uma unidade geradora é interrompida para a manutenção, outra unidade geradora na mesma usina elétrica deve continuar a sua execução para atender as restrições. Apenas projetos de manutenção são afetadas pelas restrições de parada.

2.1.1 Formato da Instância de Entrada

O formato da entrada disponibilizada pela companhia geradora de energia é composto por: projetos, pontos de atenção, horizonte de planejamento, recursos disponíveis ao longo do horizonte de planejamento e informações sobre as usinas hidroelétricas (MIRA et al., 2015).

O PH é de 60 meses e há mais 60 meses de *horizonte de execução*. Os cinco primeiros anos são para o PH, enquanto o resto é para os projetos que terminarem depois do PH. Duplicar o tempo de PH é suficiente já que nenhum projeto executa em mais que 5 anos.

2.1.1.1 Recursos Disponíveis

A empresa disponibiliza recursos financeiros para a execução dos projetos. Por questões de fiscalização, administrativas e governamentais, os recursos são classificados em dois tipos: gastos operacionais (OPEX) e gastos capitais (CAPEX). Os custos dos projetos sempre serão apenas de uma dessas duas categorias.

2.1.1.2 Projetos

Os projetos são definidos seguindo os parâmetros a seguir:

1. Identificador do projeto: número inteiro único.

2. Área: Identifica a unidade administrativa ou a usina hidroelétrica associada ao projeto. As unidades administrativas incluem o COG e o IT, já as usinas hidroelétricas incluem o AGV, BAB, BAR, CAC, EUC, IBI, LMO, MOG, NAV, PRO, SJS e SJQ.

3. Identificador da unidade geradora: para os projetos de manutenção, informa a unidade geradora que será afetada pela execução do projeto na sua usina hidroelétrica correspondente.

4. Duração da manutenção: para os projetos de manutenção, indica o tempo de manutenção do projeto. Para os projetos de curta duração é usado o (S), para longa duração o (L) e para os que não são de manutenção (N).

5. Mês de início da parada: para os projetos de manutenção, indica qual mês o projeto, contando desde o início do primeiro mês de execução, irá parar a unidade geradora. Por exemplo, se um projeto iniciou no mês 10 e o mês de início de parada for 10, então o projeto irá desativar a unidade geradora no mês 20.

6. Duração da parada: para os projetos de manutenção, informa quanto tempo, em meses, que a unidade geradora ficará desativada devido ao projeto.

7. Mês inicial: é o mês pré-definido que cada projeto será iniciado.

8. Tempo de espera: é o número de meses, contando do início do PH, após o qual o projeto pode ser agendado para ser iniciado.

9. Tipo de projeto: os projetos podem ser de gerenciamento de risco ou de não-risco. O tipo de projeto é usado para determinar se um projeto é obrigatório ou não. Projetos de gerenciamento de não-risco são obrigatórios, e portanto devem ser iniciados no mês pré-definido.

10. Recursos: determina o tipo do recurso (CAPEX ou OPEX) que o projeto consome.

11. Custos: informa a quantidade de recurso que o projeto irá consumir em cada mês em que será executado. A *duração do projeto* é determina pela quantidade de meses que o projeto será executado.

2.1.1.3 Pontos de Atenção

Os pontos de atenção são definidos seguindo os parâmetros a seguir:

1. Identificador: número inteiro único.

2. Valor de risco: um valor real indicando o risco do ponto de atenção.

3. Grupo de projetos: lista de identificadores de projetos que controlam o ponto de atenção.

4. Categoria do risco: classifica o ponto de atenção em crítico ou não-crítico.

5. Prazo limite: o último mês que o ponto de atenção deve ser controlado. Esse parâmetro é presente em apenas pontos de atenção intoleráveis.

2.1.1.4 Usinas Hidroelétricas e Unidades Geradoras

O sistema de geração de energia do nosso problema inclui 12 usinas hidroelétricas. Cada usina possui um número de unidades geradoras e devem seguir uma regulamentação específica para sua operação de acordo com sua localização geográfica e divisão administrativa (MIRA et al., 2015).

Os parâmetros de cada usina hidroelétrica são os seguintes:

1. Identificador de usina: um código de três letras único.

2. Unidades geradoras: quantidade de unidades geradoras por usina.

3. Divisão: a divisão responsável pelo gerenciamento da usina.

4. Localização geográfica: localização geográfica da usina.

A Tabela 1 mostra a hierarquia administrativa, informações sobre as divisões, áreas e unidades geradoras, localização geográfica e o número de unidades geradoras por usina.

Por exemplo, a usina BAB está localizada na região BBB e está associada a divisão BAR e possui 4 unidades geradoras.

Divisão	Área/Usina	Localização	Unidades Geradoras
COG	COG	-	-
IT	IT	-	-
AGV	AGV	AV	6
PRO	NAV	NAPI	3
	PRO	NAPI	3
BAR	BAB	BBB	4
	BAR	BBB	3
	IBI	NAPI	3
LMO	CAC	RP	2
	EUC	RP	4
	LMO	RP	2
	MOG	RMG	2
	SJS	RJM	2
	SJQ	RJM	1

Tabela 1 – A tabela mostra a hierarquia administrativa, informações sobre as divisões, áreas e unidades geradoras, localização geográfica e o número de unidades geradoras por usina.

2.1.2 Restrições

Nesta seção são listadas todas as restrições para esse problema de agendamento de portfólios.

Seja I um conjunto de projetos de entrada, W o conjunto de pontos de atenção e T o tamanho do horizonte de planejamento, em meses. Para cada $p \in I$, sua duração será representado por d_p.

O portfólio é o conjunto de pares (p, m) onde p é o projeto e o m é mês de início do projeto p, isto é, $P \subseteq I \times [T]$. Vamos identificar $I_P \subseteq I$ como os projetos que fazem parte do portfólio.

Um portfólio P é válido se satisfaz as seguintes restrições:

Agendamento consistente: Um projeto não pode ser agendado para iniciar em dois meses diferentes.

$$\text{Se } (p, m_1) \in P \text{ e } (p, m_2) \in P \text{ então } m_1 = m_2, \text{ para todo } p \in I_P. \tag{2.1}$$

Isso significa que um portfólio válido P é de fato uma função $P : I_P \mapsto [T]$. Usaremos m_p para indicar o mês de início do projeto e $e_p = m_p + d_p - 1$ indica o último mês de execução do projeto p, quando $p \in I_P$.

Tempo de espera: Um projeto p não pode iniciar antes que o seu tempo de espera l_p tenha passado. Então,

$$m_p > l_p, \text{ para todo } p \in I_p. \tag{2.2}$$

Projetos obrigatórios: Um projeto obrigatório p deve iniciar no mês definido na entrada.

$$(p, p_m) \in P, \text{ para todo projeto obrigatório } p \in I_P. \tag{2.3}$$

Pontos de atenção intoleráveis: Um ponto de atenção de alto risco é classificado como intolerável e portanto possui um prazo limite. Isso significa que todos os projetos que controlam um ponto de atenção intolerável devem ser completados antes do seu prazo limite. Seja $G_w \subseteq I$ o grupo de projetos que controlam um ponto de atenção w, onde W são os pontos de atenção de I. Então devemos ter

$$G_w \subseteq I_P, \text{para todo ponto de atenção intolerável } w \in W. \tag{2.4}$$

Essa restrição satisfeita, define

$$f_w = \max_{p \in G_w}(m_p + d_p + 1). \tag{2.5}$$

Exigimos ainda que

$$f_w \leq D_w, \text{ para todo ponto de atenção intolerável } w \in W, \tag{2.6}$$

onde D_w é o prazo limite associado ao ponto de atenção intolerável.

Limite de recursos: Cada projeto possui um gasto mensal do início até o último mês de execução. Todo o recurso gasto durante a execução do projeto é de apenas um tipo. O total de gastos de todos os projetos em execução por ano não deve ultrapassar os recursos disponíveis do ano e do mesmo tipo.

O conjunto da classificação dos recursos é $Q = \{CAPEX, OPEX\}$. Para qualquer projeto $p \in I$, seja $r_{p,m}$ o recurso gasto pelo projeto p durante o m-ésimo mês de execução ($1 \leq m \leq d_p$). Para o portfólio P, nós coletamos em $P_{t,q}$ o conjunto de todos os projetos da classe $q \in Q$ que estão ativos no mês $t \in [2T]$, isso é $P_{t,q} = \{p \in I_P | t \in [m_p, e_p]\}$. Então

$$c_{P,t,s} = \sum_{p \in P_{t,q}} r_{p,t-m_p+1}, \qquad t \in [2T], q \in Q, P \text{ um portfólio}, \tag{2.7}$$

onde $c_{P,t,s}$ é o consumo dos recursos no mês t de todos os projetos ativos da classe q que já estão agendado no portfólio P. Podemos calcular o consumo dos recursos da classe q no ano y para o portfólio P, denotado por $C_{P,y,q}$, da seguinte maneira:

$$C_{P,y,q} = \sum_{t \in M_y} c_{P,t,q}, \qquad y \in [1, T/12], q \in Q, \tag{2.8}$$

$$M_y = [12y - 11, 12y], P \text{ um portfólio}.$$

Se $S_{y,q}$ é a quantidade de recursos total da classe q disponível para o ano y, temos a restrição

$$C_{P,y,q} \leq S_{y,q} \qquad \text{para todo } y \in [1, T/12], q \in Q, P \text{ um portfólio}. \tag{2.9}$$

As próximas restrições são relacionadas aos projetos de manutenção. As unidades geradoras apenas são interrompidas se um projeto de manutenção em execução afetar elas. Um projeto de manutenção desativa uma unidade por um período de tempo. Esse tempo é definido no projeto, no mês de início de parada e a duração de parada.

Restrição de parada por localização geográfica: As restrições envolvendo paradas de unidades geradoras levam em conta considerações de engenharia. Seja $g(u, m)$ o número de unidades geradoras paradas na planta u e no mês m em um dado portfólio, enquanto $g_L(u, m)$ denota o número de unidades geradoras paradas na planta u e no mês m devido a um projeto de manutenção longo. Referindo-se a figura 1, as restrições são:

1. Na localização RP, em um dado mês qualquer, se uma usina possuir duas ou mais unidades geradoras paradas, as outras usinas não podem possuir nenhuma unidade geradora desativada.

$$(g(u_1, m) \geq 2) \rightarrow (g(u_2, m) = 0), \tag{2.10}$$

   para $m \in [2T]$ e $u_1, u_2 \in \{CAC, EUC, LMO\}$, sendo $u_1 \neq u_2$.

2. Na localização BBB, temos:

   - Em um dado mês, se a usina possuir duas ou mais unidades geradoras paradas, as outras usinas não podem possuir nenhuma usina desativada. Portanto,

$$(g(u_1, m) \geq 2) \rightarrow (g(u_2, m) = 0), \tag{2.11}$$

para $m \in [2T]$ e $u_1, u_2 \in \{BAB, BAR\}$, sendo $u_1 \neq u_2$.

- Em um dado mês, no máximo duas unidades geradoras de BAB podem ser desativadas ao mesmo tempo, apenas se nenhum outra estiver desativada na usina BAR. Portanto,

$$(g(BAB, m) = 2) \rightarrow (g(BAR, m) = 0), \quad \text{para } m \in [2T]. \tag{2.12}$$

- Em um dado mês, no máximo duas unidades geradoras podem ser desativas ao mesmo tempo na usina BAR. Portanto,

$$g(BAR, m) \leq 2, \qquad \text{para } m \in [2T]. \tag{2.13}$$

3. Na localização NAPI, em um dado mês qualquer, no máximo duas unidades geradoras podem ser desativas ao mesmo tempo. Portanto,

$$g(NAV, m) + g(PRO, m) + g(IBI, m) \leq 2, \qquad \text{para } m \in [2T]. \tag{2.14}$$

4. Em um dado mês qualquer, na localização AV, no máximo duas unidades geradoras podem ser desativas ao mesmo tempo devido a projetos de manutenção longo. Portanto,

$$g_L(AGV, m) \leq 2, \qquad \text{para } m \in [2T]. \tag{2.15}$$

Restrições de parada na localização AGV: Em um dado mês qualquer, se duas ou mais unidades geradoras da usina AGV estiverem desativada, então nenhuma unidade geradora das usinas BAB, NAV, ou PRO podem ser interruptas para manutenção, isso é

$$(g(AGV, m) \geq 2) \rightarrow (g(BAB, m) = g(NAV, m) = g(PRO, m) = 0), \quad m \in [2T]. \tag{2.16}$$

Restrição de parada por divisão: Em um dado mês qualquer, na mesma divisão não pode possuir mais de três unidades geradoras desativadas. Seja D o conjunto de divisões, conforme a tabela 1. Para cada divisão $d \in D$, seja U_d o conjunto de usinas hidroelétricas em d. Devemos ter

$$\sum_{u \in U_d} g(u, m) \leq 3, \qquad \text{para } m \in [2T], d \in D. \tag{2.17}$$

Por exemplo, se nenhuma outra restrição for violada, então é permitido desativar uma unidade de cada usina BAR, BAB, e IBI, na divisão BAR.

2.1.3 Função Objetivo

A função objetivo utilizada neste trabalho foi desenvolvida pensando na redução de risco de pontos de atenção não controlados do portfólio (MIRA et al., 2015).

Relembrando que W é o conjunto de pontos de atenção de uma entrada I para o problema. Seja R_w o risco do ponto de atenção $w \in W$. Para o portfólio P seja $W_P \subseteq W$ o conjunto de pontos de atenção definitivamente controlados em P. Então, a função que indica o montante de risco não controlado ao longo do tempo é definida como:

$$R(P) = 2T \sum_{w \in W} R_w - \sum_{w \in W_P} R_w(2T - f_w), \tag{2.18}$$

onde f_w, dado na Equação 2.5, é o primeiro mês que o ponto de atenção foi finalmente controlado.

Para dada uma entrada I do problema, seja V_I o conjunto de todos os portfólios válidos obtidos de I. O objetivo, então, é construir um portfólio P^* tal que $R(P^*)$ é o mínimo entre $\{R(P)|P \in V_I\}$. Na próxima seção é explicado o método heurístico utilizado para resolver o problema.

A Figura 1, reproduzida do trabalho (MIRA et al., 2015), mostra o gráfico da função objetivo.

A parte cinza representa o risco controlado. Cada um dos retângulos representa o risco ao longo do tempo de cada ponto de atenção, por exemplo o primeiro retângulo mais abaixo no gráfico representa o risco não controlado pelo tempo (em meses) do ponto de atenção W_1. Ao tentarmos minimizar $R(P)$, desejamos encontrar um portfólio que reduza a área em cinza no gráfico, ou seja, que controle os riscos dos pontos de atenção o mais cedo possível no horizonte de planejamento.

2.2 GRASP

O GRASP é uma meta-heurística com um procedimento de busca local com vários recomeços, em que cada iteração é composta por uma fase de construção e uma busca local (FESTA; RESENDE, 2002).

No trabalho de (BINATO; OLIVEIRA; ARAÚJO, 2001) o GRASP foi aplicado no problema do planejamento da expansão de transmissão. Apesar de outros métodos apresentarem bons resultados, o GRASP também apresentou bons resultados e em alguns casos até melhores do que outras técnicas utilizadas.

Em outro exemplo, o GRASP conseguiu encontrar bons resultados para resolver o problema do corte bidimensional guilhotinado (ALVAREZ-VALDES; PARREÑO; TAMARIT, 2005).

Em outro trabalho essa meta-heurística foi aplicada para resolver o problema de comunicação cooperativa em redes ad hoc (COMMANDER et al., 2006), onde conseguiu encontrar bons resultados.

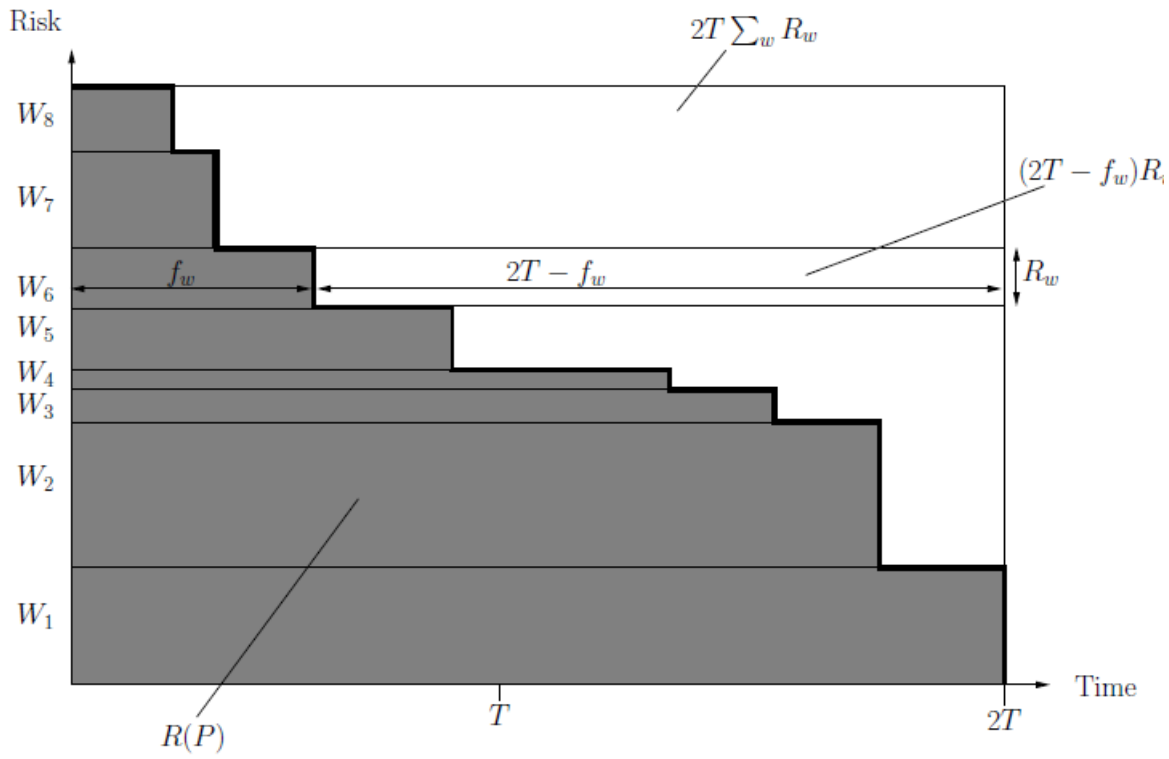

Figura 1 – Gráfico da função objetivo $R(P)$.

O GRASP (Algoritmo 1) espera como entrada os dados apresentados e discutidos na Seção 2.1.1. O algoritmo repete as fases de construção e busca local R vezes e retorna o melhor portfólio encontrado durante as repetições.

2.2.1 Fase de construção

A fase de construção é responsável por gerar um conjunto de soluções válidas. Uma solução de PPS é composta por uma lista de elementos, onde cada elemento é o agendamento de um projeto. Cada solução é construída escolhendo-se cada elemento de uma lista de candidatos restritos ordenadas por uma função gulosa que mede o benefício daquele elemento (FEO; RESENDE, 1995). Após a escolha de um elemento, a lista de candidatos deve ser refeita e reordenada a partir da função gulosa, o que caracteriza o método adaptativo. A característica de aleatoriedade do método é devido a escolha randômica entre os k melhores elementos da lista ordenada. Por exemplo, se a lista de candidatos possui 10 elementos e $k = 5$, então a escolha do candidato é feita entre os 5 primeiros melhores elementos.

A fase de construção é repetida até que o *pool* de soluções válidas C seja preenchida.

A seguir apresentamos a descrição do procedimento da fase de construção (Algoritmo 2).

Na linha 1 é criado outro conjunto para armazenar e ordenar de forma decrescente os elementos candidatos gerados a partir da função gulosa de benefício. As linhas 2 a 10 são

Algoritmo 1: GRASP

```
1  Inicialize um portfólio vazio como o atual BEST_SOLUTION
2  para R iterações faça
3  |  // Fase de construção
4  |  Inicialize um pool de soluções vazio Π.
5  |  enquanto |Π| < C faça
6  |  |  O algoritmo de construção retorna uma solução P para o problema PPS.
7  |  |  se P satisfaz a restrição de ponto de atenção intolerável então
8  |  |  |  Inclui P no pool de soluções Π.
9  |  |  fim
10 |  fim
11 |  // Fase de busca local
12 |  para cada solução P entre as β melhores soluções em Π faça
13 |  |  P = Busca local(P).
14 |  |  se P é melhor que BEST_SOLUTION então
15 |  |  |  BEST_SOLUTION = P.
16 |  |  fim
17 |  fim
18 fim
19 retorna BEST_SOLUTION.
```

Algoritmo 2: Algoritmo de construção

```
1  criar e ordenar lista de candidatos a partir da função gulosa;
2  enquanto houver elementos candidatos faça
3  |  escolher aleatoriamente entre os k melhores candidatos;
4  |  se P + candidato for válido então
5  |  |  P = P+ candidato;
6  |  senão
7  |  |  descartar o candidato;
8  |  fim
9  |  recriar a lista de candidatos;
10 fim
11 retorna solução P;
```

responsáveis por criar um portfólio e serão executadas enquanto houver elementos na lista de candidatos que possam ser inseridos neste portfólio.

Na linha 3 é feita uma escolha aleatória entre os k melhores elementos da lista dos candidatos. Entre as linhas 4 a 8, é feita uma validação da solução. Se ao adicionar o candidato escolhido para a solução P, esta não violar alguma restrição, então o candidato é mantido na solução (linha 5). Porém, se ao adicioná-lo a nova solução parcial violar alguma restrição, então o candidato é descartado (linha 7). Na linha 11 é retornada a solução P.

2.2.2 Fase de busca local

Nesta fase do algoritmo, a partir de uma solução inicial, proveniente do conjunto de soluções gerado na fase anterior, é feita uma busca por soluções melhores na sua vizinhança. A vizinhança de uma solução é determinada de acordo com o problema. Dependendo da definição de vizinhança o número de soluções analisadas pela busca local pode ser "grande"ou "pequena".

No problema PPS a vizinhança é definida como: dada uma solução P são consideradas soluções vizinhas de P aquelas obtidas ao se modificar o mês de início m de cada projeto em uma faixa de valores entre $m - \Delta$ e $m + \Delta$, onde Δ é um parâmetro da heurística. Desejamos encontrar uma solução melhor do que a corrente em sua vizinhança, caso ela exista.

A busca pela melhor solução na vizinhança pode ser feita de duas maneiras: *best-fit* ou *first-fit*. A primeira maneira consiste em percorrer toda a vizinhança e determinar qual é a melhor solução baseada na função objetivo. Já a segunda maneira consiste em percorrer a vizinhança até que encontremos uma solução melhor do que a atual.

Essa fase é repetida até um critério de parada estabelecido ser satisfeito. Por exemplo, o procedimento pode encerrar quando não for possível encontrar soluções melhores ou ser repetido um certo número de vezes.

Algoritmo 3: Busca local

```
   Entrada: solução P inicial;
1  enquanto critério de parada não for cumprido faça
2  |  pegue a lista de vizinhos V da solução P;
3  |  percorra a vizinhança V;
4  |  se a solução Q vizinha em V é melhor do que P então
5  |  |  P ← Q;
6  |  fim
7  fim
8  retorna P;
```

A seguir apresentamos a descrição do procedimento de busca local (Algoritmo 3).

O algoritmo recebe como parâmetros uma solução P. A partir dela é feita a busca pela vizinhança V de P. As linhas entre 1 e 7 são executadas até que um critério de parada seja satisfeito.

Na linha 2 é criado um conjunto V que armazena as soluções vizinhas de P e na linha 3 elas são percorridas. Na linha 4 é feita uma verificação, se a solução Q vizinha em V for melhor do que a solução P, então P recebe Q (linha 5). Na linha 8 a solução P é retornada.

Algoritmo 4: Procedimento de retorno de uma vizinhança de uma solução

```
   Entrada: Uma solução inicial P
1  Inicialize uma lista de soluções V vazia
2  para todo par (p, m) de P faça
3  |  para todo mês t no intervalo de [m − Δ, m + Δ] faça
4  |  |  Q = (P − {(p, m)}) ∪ {(p, t)}
5  |  |  se Q for válido então
6  |  |  |  V = V ∪ Q
7  |  |  fim
8  |  fim
9  fim
10 retorna lista de vizinhos V
```

O procedimento do Algoritmo 4 recebe como parâmetros uma solução P. Na linha 1 é criado e inicializado um conjunto para armazenar as soluções vizinhas de P que serão retornadas. Entre as linhas 2 e 9 são executadas para todos os pares (p, m), onde m é o mês de agendamento do projeto p na solução P. Entre as linhas 3 e 8 a execução ocorre até que o intervalo $[m - \Delta, m + \Delta]$ seja cumprido. Na linha 4, a variável Q recebe a solução inicial P tal que o par (p, m) é alterado para o par (p, t). Nas linhas 5 e 7 é feita uma validação para verificar se a solução vizinha Q não viola nenhuma restrição, ou seja, ela é válida. Caso seja válida, Q adicionada para o conjunto de soluções vizinhas V. E no final, na linha 10, é retornada a lista de vizinhos V encontrada pelo busca local.

2.2.3 GRASP para o problema PPS

Na versão do GRASP utilizada para resolver o PPS apresentado são utilizadas 2 listas de elementos candidatos na fase de construção. Essas duas listas contém os elementos que serão escolhidos gulosamente e classificam os projetos em duas categorias. Observe que essa versão do GRASP é não-adaptativa, isso significa que as listas de candidatos não são recalculadas após um elemento candidato for escolhido. A escolha dessa versão não-adaptativa foi devido as limitações encontradas na versão adaptativa para esse problema no trabalho de Mira e colaboradores (MIRA et al., 2015).

Seja I e W, respectivamente, os conjuntos de projetos e pontos de atenção com o PH T. Para avaliar o quão vantajoso é escolher um par $(p, m) \in I \times [T]$, é calculado o

benefício $b_{p,m}$ do par (p, m). Seja R_p a soma das porções de riscos dos pontos de atenção que o projeto p controla, calculado sobre todos os pontos de atenção W, a função $r_{p,t}$ é o quanto de recurso foi consumido por um projeto p durante a seu t-ésimo mês de execução e, recordando, d_p indica a duração de um projeto p. Nós temos

$$b_{p,m} = R_p((2T - m - d_p + 1)/\sum_{t=1}^{d_p} r_{p,t}), \tag{2.19}$$

então temos que $b_{p,m}$ representa o quanto de risco um projeto p ajuda a controlar, ponderado o quão longo o risco foi resolvido por unidade de custo.

Na fase de construção, inicialmente os projetos obrigatórios são agendados e então removidos de I. Em seguida, os pares $(p, m) \in I \times [T]$ tem seu benefício calculado e as duas listas de candidatos são construídas e ordenadas por benefício de forma decrescente. A lista *high-risk sorted pairs* (HRSP) são os pares (p, m) onde o projeto p contribui para controlar um ponto de atenção de risco alto. A lista *sorted pairs* (SP) são os pares restantes, ou seja, são projetos que não controlam pontos de atenção de risco alto. A lista de candidatos é divida em dois para dar maior prioridade a lista de candidatos HRSP sobre SP, já que seus projetos ajudam a controlar pontos de atenção de risco alto.

Na fase de construção, a construção de um portfólio é feita escolhendo iterativamente um par (p, m) das lista HRSP e SP de candidatos com probabilidade, respectivamente, η e $\eta - 1$. Um vez escolhida a lista, o par (p, m) é escolhido aleatoriamente entre os k primeiros pares da lista. Então o par (p, m) é incluído no portfólio caso a sua inclusão não violar nenhuma das seguintes restrições:

- "agendamento consistente", Eq. (2.1);
- "tempo de espera", Eq. (2.2);
- "limite de recursos", Eq. (2.7);
- "restrições de parada", Eqs. (2.10-17).

Observe que, a cada novo par (p, m) incluído, o recurso disponível a cada ano diminui e a iteração é repetida até que nenhum par (p, m) possa ser incluído no portfólio. No final do algoritmo de construção o portfólio gerado é retornado.

2.3 BRKGA

O BRKGA (*Biased Random-Key Genetic Algorithm*) é um algoritmo genético (MENDES; GONÇALVES; RESENDE, 2009) que segue o princípio darwinista de escolher sempre os melhores indivíduos de um população corrente para gerar uma nova população (MENDES; GONÇALVES; RESENDE, 2009). Uma população é representada por um conjunto de

vetores, onde cada vetor corresponde a um indivíduo. De maneira diferente a outros algoritmos genéticos ele realiza uma escolha enviesada de indivíduos elite para realizar os cruzamentos.

O BRKGA foi aplicado no problema de formação de células de fabricação e seus resultados foram na média melhores do que os outros métodos utilizados (GONÇALVES; RESENDE, 2004).

Em outro trabalho o BRKGA foi comparado com outros dois algoritmos genéticos no problema de empacotamento ortogonal bidimensional. O BRKGA conseguiu encontrar em média melhores resultados entre os algoritmos comparados (GONÇALVES, 2007).

Em outro exemplo, o BRKGA foi aplicado e comparado com outros algoritmos no problema de Job-Shop. Neste problema o BRKGA conseguiu resultados bons nos testes (GONÇALVES; MENDES; RESENDE, 2005).

O algoritmo começa com uma população p com cada indivíduo contendo n genes, cada gene possui um valor de chave escolhida aleatoriamente entre [0,1). A partir dessa população são geradas novas populações. Na i-ésima geração a população é particionada em dois conjuntos p_e e $p - p_e$. O conjunto p_e, de tamanho $p_e < p/2$, são os indivíduos elites, ou seja, os melhores indivíduos segundo uma função de *fitness*. O conjunto $p - p_e$ são os indivíduos restantes não-elite.

Todos os elites são copiados para a próxima população da *i+1*-ésima geração. A população da *i+1*-ésima geração é composta pelos vetores elite copiados, p_m vetores mutantes que são indivíduos escolhidos aleatoriamente da i-ésima população com chaves geradas aleatoriamente e $p - p_e - p_m$ são vetores gerados a partir do cruzamento dos vetores da população da i-ésima geração.

Cada indivíduo mutante é gerado a partir de um indivíduo (elite ou não-elite) escolhido aleatoriamente da população anterior. Após escolhido o indivíduo é feito uma "mutação" nos seus genes, ou seja, os valores das chaves são alterados no intervalo de [0,1). Os vetores mutantes são utilizados para evitar que a população convirja para um ótimo local não global.

Os vetores gerados (filho C) são o resultado do cruzamento de um pai A *elite* e um pai B *não-elite*, ambos escolhidos aleatoriamente. Esse cruzamento é feito escolhendo-se aleatoriamente entre as chaves dos pais que serão herdadas pelo filho. Essa escolha é enviesada, pois a chance de escolher uma chave do pai elite é maior.

A seguir apresentamos a descrição do algoritmo BRKGA (Algoritmo 5).

Na linha 1 é inicializada uma variável f^* para armazenar o valor de *fitness* da melhor solução com infinito. As linhas de 2 a 22 são executadas até que a condição de parada da linha 2 seja satisfeita. A condição de parada pode por ser por exemplo um número de vezes que será executado a geração de populações. Na linha 3 é criada uma população inicial que é construída por meio da geração de chaves aleatórias. As linhas entre 4 e 21 são executadas até que o critério de reinicialização seja cumprido. O critério de reinicialização

Algoritmo 5: BRKGA

```
1  inicializar o fitness da melhor solução com infinito: f* ← ∞;
2  enquanto critério de parada não for satisfeito faça
3      Gere uma população P com vetores de n chaves aleatórias;
4      enquanto critério de reinicialização não for satisfeito faça
5          particione a população em elite e não-elite;
6          copie os indivíduos elite para a próxima população: P+ ← Pe;
7          gere indivíduos mutantes Pm e adicione para a próxima população:
           P+ ← P+ ∪ Pm;
8          para todo i de 1 até P − Pe − Pm faça
9              escolha um pai a elite aleatoriamente;
10             escolha um pai b não-elite aleatoriamente;
11             c ← gerarFilho(a, b);
12             adicionar o filho c para a próxima geração: P+ ← P+ ∪ {c};
13         fim
14         atualizar a população: P ← P+;
15         encontre a melhor solução de P;
16         χ+ ← melhor solução de P;
17         se f(χ+) < f* então
18             χ* ← χ+;
19             f* ← f(χ+);
20         fim
21     fim
22 fim
23 retorna χ*;
```

verifica se atingimos uma população que contém indivíduos ótimos e pode ser, por exemplo, um número máximo de gerações que serão criadas. Na linha 5 é feita a partição dos indivíduos elite e não-elite, na linha 6 os elites são copiados para a próxima população $P^+ \leftarrow P_e$. Na linha 7 são gerados os mutantes e adicionados para a próxima população $P^+ \leftarrow P^+ \cup P_m$.

Entre as linhas 8 e 13 são gerados os indivíduos filhos. Essas linhas são executadas até preencher o restante da população $P - P_e - P_m$. Na linha 9 é escolhido o pai A aleatoriamente entre os indivíduos elite e na linha 10 o pai B escolhido aleatoriamente entre os indivíduos não-elite. Na linha 11 é gerado o indivíduo filho dos pais A e B e na linha 12 ele adicionado para a próxima população $P^+ \leftarrow P^+ \cup \{c\}$.

Na linha 14 a população atual é atualizada com a próxima população $P \leftarrow P^+$. Na linha 15 a população P é percorrida para encontrar o melhor indivíduo baseado na função $fitness$ e na linha 16 uma variável recebe esse indivíduo. Entre as linhas 17 e 20 é feita uma validação se o valor do $fitness$ encontrado é o maior, caso seja verdadeiro a variável que armazena a melhor solução é atualizada e o valor do melhor $fitness$ também. E por fim, na linha 23 é retornado a solução χ^*.

O procedimento *gerarFilho* (Algoritmo 6) recebe como parâmetros o pai A elite e o

Algoritmo 6: gerarFilho

```
   Entrada: Pai A elite, Pai B não-elite
1  para todo j ← 1 até n faça
2  |  Jogue uma moeda viciada com probabilidade de escolher o pai A > 0,5 cara
3  |  se der cara então
4  |  |  C[j] ← A[j];
5  |  senão
6  |  |  C[j] ← B[j];
7  |  fim
8  fim
9  retorna C;
```

pai B não-elite. As linhas de 1 a 8 são executadas n vezes, onde n é a quantidade de genes de um indivíduo. Na linha 2 é utilizado um método de escolha aleatória enviesado para garantir que a escolha do pai elite seja maior. Isso mostra o porquê o BRKGA é um algoritmo enviesado. Nas linhas 3 e 7 é feita uma validação, se for cara o filho recebe uma chave do pai A elite, mas se for coroa recebe do pai B não-elite. Por fim, na linha 9, é retornado o filho C.

2.3.1 BRKGA para o problema PPS

Para utilizar o BRKGA para a versão do PPS apresentado foram feitas algumas adaptações. Um indivíduo é uma solução/portfólio representada por um vetor onde os índices são os *ids* de projetos e os valores (chaves aleatórias) de cada elemento do vetor são os meses de início. Uma população é um conjunto de portfólios de projetos.

A função *fitness* é definida como a função objetivo do problema, ela calculará o quanto o risco foi controlado pela solução retornada pelo algoritmo BRKGA.

3 METODOLOGIA

Foram feitas diversas pesquisas em artigos e livros para escrita dos conceitos descritos no Capítulo 2, reuniões semanais para tomadas de decisões de pesquisa e implementação e esclarecimento de dúvidas, além de testes em laboratório.

As implementações foram baseadas nos trabalhos de (MIRA et al., 2015) e (MENDES; GONÇALVES; RESENDE, 2009). Os formatos dos dados são baseados no trabalho de (MIRA et al., 2015).

3.1 Detalhes de Implementação

Para a implementação poderia ser utilizada tanto a linguagem de programação *C* ou *C++*, por ambas serem muito adequadas para implementação de algoritmos de otimização. Foi escolhida a linguagem *C*, porque possuímos maior conhecimento dessa linguagem.

Para auxiliar na implementação dos algoritmos foi usada a *IDE Code::Blocks* que é um ambiente de desenvolvimento de software para *C/C++*. Essa disponibiliza diversas ferramentas que facilitam o desenvolvimento. Por exemplo, ela facilita a depuração do algoritmo, tornando as correções de erros mais fáceis. Também foi utilizado o editor de texto de programação *Sublime Text*.

Os algoritmos necessitam a leitura de um arquivo de entrada contendo todas as informações de projetos, pontos de atenção e recursos disponíveis. Esses arquivos são escritos em formato *JSON*. Ele é um formato simples e muito utilizado.

Para tornar o uso de *JSON* mais fácil e simples foi utilizada uma biblioteca em linguagem *C* para leitura de arquivos *JSON* chamada *cJSON*. Ela possui funções e métodos que facilitam gravar e ler arquivos em formato *JSON*.

Para gerar os valores aleatórios necessários para ambos os algoritmos, foi utilizada uma função do *C* chamada *rand()*. Esta função retorna um valor entre 0 e *RAND_MAX*. Por exemplo, se quisermos obter um valor entre 0 e 9, usamos *rand*()%10. Para que a função *rand()* não retorne sempre os mesmos valores a cada execução é necessário utilizar a função *srand()* para gerar uma nova semente. Esta função garante que a cada nova execução da função *rand()* novos valores possam ser escolhidos.

Foi decido utilizar a estrutura de dados *Hash*, ao invés de *Arrays*, devido a facilidade de manipulação e o ganho de processamento em algumas partes do algoritmo, como, por exemplo, adicionar e deletar elementos na *Hash*. Todos os dados necessários para a execução do algoritmo são armazenados em *Hash*.

Para facilitar o uso de *Hash*, foi utilizada uma biblioteca em linguagem *C* chamada *UtHash*. Ela possui funções e métodos que facilitam a manipulação das estruturas de dados. Por exemplo, *HASH_ITER* é um método da biblioteca que percorre toda a *Hash*.

Durante o desenvolvimento foram tomadas algumas decisões para facilitar a implementação dos algoritmos. A seguir discutimos algumas das tomadas de decisões:

- Surgiram oportunidades de utilizar algoritmos já implementados em outras linguagens como *Go* e *C++*, contudo devido a complexidade dos códigos e falta de conhecimento dessas linguagens, foi decidido que nossos algoritmos seriam implementados do zero usando a linguagem *C*.

- A ideia inicial para a execução dos algoritmos era utilizar os mesmos arquivos de entrada utilizados no trabalho de (MIRA et al., 2015), contudo não tivemos acesso à eles, então foi decidido que as entradas seriam geradas por um gerador de entrada *JSON*.

- Durante a construção/geração de uma solução, todos os projetos são armazenados, porém aqueles que não fazem parte da solução, ou seja, não atenderam as restrições, são agendados para o mês 61.

- Foi criado uma *Hash* que mapeia projetos para pontos de atenção. Isso facilita a partir de um projeto saber quantos e quais pontos de atenção que ele controla.

- Para cada solução o valor da função objetivo é calculada e armazenada. Isso agiliza e facilita a ordenação das soluções e a busca da melhor solução.

- No GRASP foi decidido utilizar *Hash* para armazenar os valores de benefícios, devido a facilidade de deletar valores da *Hash*.

- No BRKGA foi decidido que na função de geração de mutantes devem ser alterados aleatoriamente os meses apenas dos projetos que fazem parte da solução, enquanto aqueles agendados no mês 61 não são alterados.

- Decidimos utilizar o *InsertionSort* como algoritmo de ordenação, devido ao seu fácil entendimento e implementação.

4 EXPERIMENTOS

Neste capítulo são descritas as formas como os experimentos foram realizados e a escolha dos parâmetros para cada algoritmo. Além disso, sumarizamos os resultados obtidos e realizamos uma análise dos mesmos.

4.1 Gerador de Instâncias de Entrada

As instâncias de entrada são geradas a partir de um algoritmo gerador *JSON*. O nome do arquivo representa a categoria do tipo de entrada. As entradas são categorizadas a partir da quantidade de projetos gerados e um identificador, este porque para a mesma quantidade de projetos podem ser geradas instâncias diferentes. Por exemplo, se o gerador produzir 3 instâncias com 100 projetos, essas entradas serão chamadas: *entrada-100p-01.json, entrada-100p-02.json, entrada-100p-03.json*.

Os recursos gerados para as classes *CAPEX* e *OPEX* de uma entrada são calculados a partir do produto de um valor entre 1.000.000 e 1.990.000 e um fator entre 1 e 5 de acordo com as respectivas categorias de entrada por quantidade de projetos.

As quantidades de projetos são consideradas nos experimentos são 30, 50, 100, 150 e 200. Para cada quantidade de projetos foram geradas várias instâncias e os algoritmos GRASP e BRKGA foram executadas para cada uma delas. Da quantidade total de projetos 20% são classificados em obrigatórios e 80% em não-obrigatórios.

A duração dos projetos é gerada aleatoriamente entre os valores de 5 a 15 meses. A partir da duração e da categoria do recurso utilizado são atribuídos os custos mensais aos projetos. O custo mensal de um projeto é de 1% a 2% do menor valor de recursos anuais da categoria (CAPEX ou OPEX).

Os projetos foram associados aos pontos de atenção a partir de um vetor que armazena os identificadores (*id's*) dos projetos não-obrigatórios. A partir desse vetor é feita uma escolha aleatória para associar o projeto ao ponto de atenção, evitando que sejam escolhidos *id's* iguais.

A quantidade de pontos de atenção é de 30% da quantidade de projetos em uma instância. Por exemplo, se o número de projetos for 200 em uma instância, serão gerados 60 pontos de atenção na mesma. Um ponto de atenção é controlado por um número de projetos igual a um valor entre 1% a 5% do total de projetos na instância de entrada.

Da quantidade total dos pontos de atenção 5% são classificados em intoleráveis e 95% em toleráveis. O deadline foi definido como um valor escolhido aleatoriamente entre 40 a 60 meses.

O grau de risco que cada ponto de atenção é escolhido aleatoriamente entre os valores de 1 a 100.

4.2 Parâmetros de Heurísticas

Os valores dos parâmetros utilizados nos experimentos do GRASP foram baseados nos valores utilizados no trabalho de (MIRA et al., 2015). Os parâmetros η, k e Δ foram os mesmos valores utilizados no trabalho citado, porém os valores de R, Repetições da busca local, C e β foram alterados para se adequarem melhor aos nossos experimentos.

Os valores dos parâmetros utilizados nos experimentos do BRKGA foram baseados nos valores utilizados no trabalho de (MENDES; GONÇALVES; RESENDE, 2009). Nos parâmetros p_e e p_m foram utilizados os valores máximos recomendados nesse trabalho. Já os demais parâmetros foram alterados para valores mais adequados aos nossos experimentos.

Os parâmetros utilizados para os testes do GRASP são apresentados na tabela 2.

Parâmetros	**Valor**	**Descrição**
R	2	número vezes que o GRASP será reinicializado
Repetições da busca local	5	número de vezes que a busca local será repetida
η	$0,7$	probabilidade de escolher a lista HRSP
k	5	k melhores candidatos
C	20	tamanho do *pool* de solução
β	2	número das melhores soluções encontradas no *pool* e que são usadas na fase de busca
Δ	8	$2 * \Delta$ tamanho da vizinhança

Tabela 2 – Parâmetros do GRASP

Os parâmetros utilizados para os testes do BRKGA são apresentados na tabela 3.

Parâmetros	**Valor**	**Descrição**
Critério de parada do BRKGA	10	número de vezes que serão geradas populações iniciais para serem evoluídas
ρ	100	Critério de reinicialização: número de gerações que serão produzidas a partir de uma população inicial
p	50	tamanho da população
p_e	$p * 25\%$	tamanho da população de elites
p_m	$p * 20\%$	tamanho da população de mutantes
p_a	70%	probabilidade de escolher um pai elite em *crossover*

Tabela 3 – Parâmetros do BRKGA

4.3 Execução dos experimentos e resultados obtidos

Os experimentos foram realizados em uma máquina *desktop* disponível no laboratório de pesquisa de *hardware* do Bloco G da UEMS - Dourados. Devido ao tempo necessário para a validação dos algoritmos e a resolução de erros encontrados durante os testes de validação, não foi possível testar muitos valores de parâmetros para os algoritmos.

Na Tabela 4 são apresentadas as configurações da máquina utilizada para realizar os testes dos algoritmos.

Sistema Operacional	**Windows 7 64Bits**
Memória Principal (RAM)	**4 Gb**
CPU	**3,40 GHz**
Processador	**Intel(R) Core(TM) i3-3240 4 Núcleos**

Tabela 4 – Configurações da máquina utilizada

Implementamos um programa responsável por executar os algoritmos BRKGA e o GRASP, passando como parâmetro os nomes dos arquivos de entrada. Para cada entrada os algoritmos serão executados 10 vezes, gerando 10 arquivos de saída. Por exemplo, para o arquivo de entrada *entrada-100p-01.json* foram geradas 10 arquivos de saídas para cada método heurístico.

Os nomes dos arquivos de saídas dos algoritmos são constituídos pela quantidade projetos, índices e um valor de *timestamp*. Por exemplo, para a entrada *entrada-100p-01.json* serão geradas as saídas com os nomes: 100p-01-saida-1538245230.json, 100p-02-saida-1538365230.json, 100p-03-saida-183824623.json, . . . , 100p-10-saida-184824723.json, totalizando 10 arquivos de saída para a mesma entrada de um mesmo algoritmo.

O conteúdo dos arquivos de saídas é um documento *JSON* composto pelo nome do arquivo de entrada responsável pela sua geração, número total de projetos, número total de pontos de atenção, tempo de execução em segundos, *timestamp*, a solução, o valor da função objetivo, valor do limitante inferior da função objetivo e a razão da função objetivo pelo valor do limitante da função objetivo. O valor do limitante inferior da função objetivo é calculado a partir de uma solução onde todos os seus projetos (menos os obrigatórios) foram agendados um dias após o seu tempo de espera, ignorando as violações de restrições.

Na Tabela 5 são mostrados os resultados obtidos pela execução dos experimentos para cada categoria de entrada para o método GRASP.

Na Tabela 6 são mostrados os resultados obtidos pela execução dos experimentos para cada categoria de entrada para o método BRKGA.

4.4 Análise de Resultados

Nesta seção discutiremos os resultados obtidos na Seção 4.3.

Entradas	**Min FO**	**Max FO**	**Média FO**	**Razão LFO**	**Média TE (s)**
30	5908	10210	7883	1,68	6,24
50	9214	15232	12797	1,63	29,19
100	27148	36636	31281	1,61	263,44
150	41136	76837	54094	1,92	807,51
200	59817	116553	86486	2,14	1824,07

Tabela 5 – Resultados obtidos para o método GRASP. A primeira coluna identifica a categoria de entrada pelo número de projetos. A coluna *Min FO* representa o menor valor da função objetivo, a coluna *Max FO* contém o maior valor da função objetivo, a coluna *Média FO* contém a média dos valores da função objetivo na categoria de entrada, a coluna *Razão LFO* apresenta a razão do valor da média de função objetivo pelo valor do limitante inferior da função objetivo, a última coluna *Média TE* representa o tempo médio de execução em segundos.

Entradas	**Min FO**	**Max FO**	**Média FO**	**Razão LFO**	**Média TE (s)**
30	5908	11207	8164	1,74	71,06
50	9995	18561	14399	1,82	218,27
100	35058	67168	49577	2,55	1613,53
150	80925	121482	100555	3,56	2733,87
200	149963	211644	177834	4,43	4717,03

Tabela 6 – Resultados obtidos para o método BRKGA. A primeira coluna identifica a categoria de entrada pelo número de projetos. A coluna *Min FO* representa o menor valor da função objetivo, a coluna *Max FO* contém o maior valor da função objetivo, a coluna *Média FO* contém a média dos valores da função objetivo na categoria de entrada, a coluna *Razão LFO* apresenta a razão do valor da média de função objetivo pelo valor do limitante inferior da função objetivo, a última coluna *Média TE* representa o tempo médio de execução em segundos.

Em ambos os métodos, conforme a quantidade de projetos aumenta, o tempo médio de execução também aumenta. O GRASP conseguiu encontrar soluções melhores e mais rapidamente que o BRKGA, no entanto isso provavelmente é devido aos baixos valores de parâmetros utilizados. Porque a partir de uma análise do pseudocódigo do método GRASP, estimamos que a fase de construção do GRASP para construir a lista de candidatos leva um tempo da ordem de $O((Tn)^2 + Tn^2|W|C)$, onde n é o número de projetos, $/W/$ é o total de pontos de atenção e C o tamanho do conjunto (*pool*) de soluções. A fase de busca local leva um tempo de execução da ordem de $O(\beta n^{2\Delta+2})$ e é o responsável principal pelo elevado tempo de execução do GRASP. Por outro lado, o algoritmo do BRKGA leva um tempo de execução da ordem de $O(\rho n^2 T|W|(np)^2)$ devido ao tempo gasto com a ordenação da população por meio do algoritmo *InsertionSort*. Observe que o tempo de execução apenas da fase de construção do GRASP é de uma ordem similar ao do tempo de execução do BRKGA. No entanto, queremos destacar que os tempos de execução apresentados para

o GRASP e BRKGA são estimativas e não demonstrações baseadas na análise do código.

Como a versão do PPS apresentado se trata de um problema de minimização do risco descontrolado, o valor da função objetivo expressa o montante de risco que uma solução não controla. Portanto, quanto menor o valor da função objetivo encontrada, melhor é a solução encontrada. A partir dos resultados obtidos é possível observar que o GRASP tanto no valor mínimo, máximo e na média da função objetivo encontrou valores melhores do que o BRKGA.

A partir tabela 5, podemos observar que o GRASP obteve os valores de 5908; 10210; 7883; $1,68$ e $6,24$ respectivamente para as colunas Min FO, Max FO, Média FO, Razão LFO e Média TE para a categoria de instâncias de entrada de 30 projetos. O menor valor de FO encontrada pelo GRASP foi 5908. É importante notar que o valor da razão LFO foi consideravelmente baixa, além disso o tempo de execução do GRASP nessa categoria foi de $6,24$. Por outro lado, o BRKGA apresentou os seguintes valores para esta categoria: 5908; 11207; 8164; $1,74$ e $71,06$. O valor da razão LFO também foi consideravelmente baixa. Os melhores valores de solução encontrados para ambos os algoritmos são os mesmos. No entanto o tempo de execução para o BRKGA é em torno de 11 vezes mais lento do que o GRASP.

Para a categoria de 50 projetos os valores do GRASP foram: 9214; 15232; 12797; $1,63$ e $2,19$. Note que o valor da razão LFO diminuiu em relação a categoria 30, e, por isso, suspeitamos que o algoritmo possa ter encontrado uma solução ótima. Agora para o BRKGA os valores para esta categoria foram: 9995; 18561; 14399; $1,82$ e $218,27$. Todos os valores encontrados para esta categoria foram superiores e alguns possuem até mesmo o dobro do valor encontrado na categoria 30. O valor da razão LFO do GRASP foi melhor do que a do BRKGA, bem como tempo de execução.

Os valores do GRASP encontrados para a categoria de 100 projetos de Min FO, Max FO, Média FO, Razão LFO e Média TE são: 27148; 36636; 31281; $1,61$ e $263,44$. O valor da razão LFO encontrado foi o menor em relação as categorias anteriores, porém o tempo de execução foi maior. No BRKGA os valores encontrados são: 35058; 67168; 49577; $2,55$ e $1613,53$. Os valores de Min FO, Max FO e Média FO para esta categoria foram mais ou menos 3,5 vezes maior e o tempo de execução foi 7 vezes maior que a categoria 50. O GRASP conseguiu encontrar o menor valor da razão LFO. Essa instância foi relativamente fácil para ser resolvido pelo GRASP, enquanto para o BRKGA foi relativamente difícil, como pode ser deduzido dos tempos de execução de ambos os algoritmos.

Foram encontrados os valores de 41136; 76837; 54094; $1,92$ e $807,51$ para a categoria de 150 projetos para o GRASP. Para esta categoria o valor da razão LFO piorou muito em relação as demais categorias analisadas até o momento e o tempo de execução também aumentou consideravelmente. Para o BRKGA os valores encontrados para esta categoria foram: 80925; 121482; 100555; $3,56$ e $2733,87$. Os valores encontrados de Min FO, Max FO e Média FO para essa categoria categoria foram aproximadamente 2 vezes maior que

ao do GRASP para a mesma e o tempo de execução 3 vezes maior. Para o BRKGA essa instância de entrada não foi favorável.

Por último, o GRASP encontrou os seguintes valores para categoria de 200 projetos: 59817; 116553; 86486; 2, 14 e 1824, 07. Nessa categoria o GRASP apresentou o pior valor de razão LFO e de tempo execução dos testes executados. O BRKGA obteve os seguintes valores 149963; 211644; 177834; 4, 43 e 471703. É possível notar que a medida que a quantidade projetos aumenta, o valor da razão de LFO e o tempo de execução aumentam significativamente.

Com base nos experimentos, observamos que o GRASP obtém melhores soluções e muito mais rapidamente do que o BRKGA, mesmo com valores de parâmetros de heurística "menores" do que o BRKGA, ou seja, com menos iterações da heurística. Portanto os resultados sugerem que o GRASP consegue encontrar melhores soluções e em menor tempo de execução do que o BRKGA.

5 CONCLUSÃO

Neste trabalho foi apresentando uma versão específica do PPS baseado em um problema real de uma companhia geradora de energia elétrica e os métodos que utilizamos para resolvê-lo foram o GRASP e o BRKGA.

O objetivo do trabalho foi de desenvolver/implementar ambos os métodos e comparar seus resultados para determinar qual foi o melhor para resolver o problema PPS.

As implementações foram feitas em linguagem de programação *C* e seguindo os trabalhos de (MIRA et al., 2015) para o GRASP e o de Mendes (MENDES; GONÇALVES; RESENDE, 2009) para o BRKGA.

As entradas para os algoritmos foram geradas aleatoriamente em formato *JSON*. Um algoritmo foi responsável por executar os métodos GRASP e BRKGA passando como parâmetros os arquivos de entrada. A cada execução do GRASP e do BRKGA é gerado um arquivo de saída no formato *JSON* contendo as informações estatísticas necessárias para preencher as tabelas apresentadas na Seção 4.3.

Com base nos resultados dos experimentos, apresentados na Seção 4.3, foi feito uma análise no tempo de execução e no valor da função objetivo.

Os resultados sugerem que o GRASP obtém melhores soluções e muito mais rapidamente do que o BRKGA, mesmo com valores de parâmetros de heurística "menores" do que o BRKGA, ou seja, com menos iterações da heurística. Portanto os resultados sugerem que o GRASP consegue encontrar melhores soluções e em menor tempo de execução do que o BRKGA.

Para trabalhos futuros seria interessante utilizar outra função objetivo ou melhorar a utilizada neste trabalho, pois foi identificada uma situação na qual os projetos de um grupo de projetos que resolvem um ponto de atenção poderiam ser ponderados de maneira melhor. Tentar utilizar outras estruturas de dados, por exemplo *arrays*, para melhorar o desempenho dos algoritmos. O tempo de execução dos métodos pode ser melhorado ao se implementar outras técnicas de ordenação. Além disso, seria interessante utilizar outros métodos heurísticos e comparar os seus resultados.

REFERÊNCIAS

ALBUQUERQUE, G. U. V. d. *Um estudo do problema de escolha de portfólio ótimo*. Tese (Doutorado) — Universidade de São Paulo, 2009.

ALVAREZ-VALDES, R.; PARREÑO, F.; TAMARIT, J. M. A grasp algorithm for constrained two-dimensional non-guillotine cutting problems. *Journal of the Operational Research Society*, Taylor & Francis, v. 56, n. 4, p. 414–425, 2005.

BINATO, S.; OLIVEIRA, G. C. D.; ARAÚJO, J. L. D. A greedy randomized adaptive search procedure for transmission expansion planning. *IEEE Transactions on Power Systems*, IEEE, v. 16, n. 2, p. 247–253, 2001.

COMMANDER, C. et al. A greedy randomized algorithm for the cooperative communication problem on ad hoc networks. In: CITESEER. *8th INFORMS Telecommunications Conference*. [S.l.], 2006.

DOERNER, K. et al. Pareto ant colony optimization: A metaheuristic approach to multiobjective portfolio selection. *Annals of operations research*, Springer, v. 131, n. 1-4, p. 79–99, 2004.

FEO, T. A.; RESENDE, M. G. Greedy randomized adaptive search procedures. *Journal of global optimization*, Springer, v. 6, n. 2, p. 109–133, 1995.

FESTA, P.; RESENDE, M. G. Grasp: An annotated bibliography. In: *Essays and surveys in metaheuristics*. [S.l.]: Springer, 2002. p. 325–367.

GENDREAU, M.; POTVIN, J.-Y. *Handbook of Metaheuristics*. 2nd. ed. [S.l.]: Springer Publishing Company, Incorporated, 2010. ISBN 1441916636, 9781441916631.

GONÇALVES, J. F. A hybrid genetic algorithm-heuristic for a two-dimensional orthogonal packing problem. *European Journal of Operational Research*, Elsevier, v. 183, n. 3, p. 1212–1229, 2007.

GONÇALVES, J. F.; MENDES, J. J. de M.; RESENDE, M. G. A hybrid genetic algorithm for the job shop scheduling problem. *European journal of operational research*, Elsevier, v. 167, n. 1, p. 77–95, 2005.

GONÇALVES, J. F.; RESENDE, M. G. An evolutionary algorithm for manufacturing cell formation. *Computers & industrial engineering*, Elsevier, v. 47, n. 2-3, p. 247–273, 2004.

MAINIERI, G. B. *Meta-heurística BRKGA aplicada a um problema de programação de tarefas no ambiente flowshop híbrido.* Tese (Doutorado) — Universidade de São Paulo, 2014.

MEGALE, G. L. B. Algoritmo genético de chaves aleatórias viciadas aplicado ao problema de clusterização de módulos de software. 2015.

MELO, E. L. d. *Meta-heurísticas Iterated Local Search, GRASP e Artificial Bee Colony aplicadas ao Job Shop Flexível para minimização do atraso total.* Tese (Doutorado) — Universidade de São Paulo, 2014.

MENDES, J. J. de M.; GONÇALVES, J. F.; RESENDE, M. G. C. A random key based genetic algorithm for the resource constrained project scheduling problem. *Computers & OR*, v. 36, n. 1, p. 92–109, 2009.

MIRA, C. et al. Project scheduling optimization in electrical power utilities. p. 1–26, 2015.

MONTES, E. *Introdução ao Gerenciamento de Projetos*. [S.l.: s.n.], 2017.

RIBEIRO, M. C. d. C. R.; ALVES, A. d. S. The problem of research project portfolio selection in educational organizations: a case study. *Gestão & Produção*, SciELO Brasil, v. 24, n. 1, p. 25–39, 2017.