CEFET/RJ
DIRETORIA DE PESQUISA E PÓS-GRADUAÇÃO

APLICAÇÃO DO ALGORITMO GENÉTICO DE CHAVES ALEATÓRIAS VICIADAS NO PROBLEMA DA MOCHILA NÃO LINEAR SEPARÁVEL

Marcelo Pinto Pereira

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Tecnologia do Centro Federal de Educação Tecnológica Celso Suckow da Fonseca, CEFET/RJ, como parte dos requisitos necessários à obtenção do Título de Mestre em Tecnologia.

Orientador

Leonardo Silva de Lima

Coorientador

Wagner Pimentel

Rio de Janeiro

Julho 2015

APLICAÇÃO DO ALGORITMO GENÉTICO DE CHAVES ALEATÓRIAS VICIADAS NO PROBLEMA DA MOCHILA NÃO LINEAR SEPARÁVEL

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Tecnologia, Centro Federal de Educação Tecnológica Celso Suckow da Fonseca, CEFET/RJ, como parte dos requisitos necessários à obtenção do Título de Mestre em Tecnologia.

Marcelo Pinto Pereira

Aprovada por:

Prof. Leonardo Silva de Lima, D.Sc. (Orientador)

Prof. Wagner Pimentel, D.Sc. (Coorientador)

Prof. Rafael Garcia Barbastefano, D.Sc.

Prof. Marcia Helena da Costa Fampa, Ph.D., (COPPE)

Rio de Janeiro

Julho 2015

Ficha catalográfica elaborada pela Biblioteca Central do CEFET-RJ

P436 Pereira, Marcelo Pinto
Aplicação do algoritmo genético de chaves aleatórias viciadas no problema da mochila não linear separável / Marcelo Pinto Pereira. — 2015.
xiii, 70 f. : il. (algumas color.), grafs, tabs; enc.

Dissertação (Mestrado) Centro Federal de Educação Tecnológica Celso Suckow da Fonseca, 2015.
Bibliografia: f. 66–70
Orientador: Leonardo Silva de Lima
Coorientador: Wagner Pimentel

1. Algoritmos genéticos 2. Programação não-linear. 3. Análise numérica. 4. Heurística. I. Lima, Leonardo Silva de (Orient.). II. Pimentel, Wagner (Coorient.). III. Título.

CDD 006.31

Dedico este trabalho à minha esposa Tânia,
pelo apoio, compreensão e paciência
durante o tempo necessário para
a realização do mesmo.

AGRADECIMENTOS

Agradeço a Deus por iluminar os meus caminhos, e por se fazer presente nos momentos em que mais precisei, além do amparo, força e determinação para eu retornar, após 36 anos, à instituição onde concluí o curso técnico de eletrônica e o Ensino Médio para fazer um curso de Mestrado.

Ao professor Leonardo Silva de Lima (D.Sc.), pelo empenho no trabalho de orientação, dedicação, paciência, competência, confiança, ensinamentos, oportunidades, entusiasmo que possibilitaram que esta dissertação fosse concluída, além de ter uma conduta irrepreensível e exemplar durante todo o processo da feitura deste trabalho.

Ao professor Wagner Pimentel (D. Sc.), pela coorientação precisa, dedicação, paciência, competência agregada à ajuda inestimável sem a qual esta dissertação não seria possível e também pelo exemplo de profissional íntegro, responsável e apaixonado pelo que faz.

Aos membros da banca, que me honraram pela participação e pelas contribuições neste projeto.

Ao professor Herivelto Nunes Paiva (Mc. C.) por participar do pontapé inicial e pela ajuda e apoio dispensados em vários momentos necessários na elaboração deste trabalho.

Aos amigos: Antônio Gomes Lacerda, pela colaboração, incentivo e dedicação ao estudo de assuntos relacionados à minha dissertação, além da ajuda na tradução de artigos e ao Lúcio Santos (Ms. C.) e André Cantareli (Ms. C.), pelo apoio e incentivo à realização deste Mestrado, antes mesmo dele ter se iniciado.

Aos professores Bruno Monteiro (D. Sc.), Fábio Freitas Ferreira (D. Sc.), Sérgio Delamare (Ms. C.) e Verônica Bareicha Corrêa pelas críticas e contribuições que enriqueceram este trabalho.

Aos colegas conhecidos durante o Mestrado, em especial a Marcia Cristina de Oliveira pela ajuda e dicas preciosas oferecidas em vários momentos.

Aos meus pais Hélio Toste e Maria Cândida pelo amor e dedicação incondicionais, por serem meus maiores mestres e me ensinarem os verdadeiros valores da vida, aos meus irmãos João Paulo e Geísa pelo carinho e por me escutarem e aconselharem nas horas difíceis.

Aos meus filhos Marcelo, Guilherme, Hélio, Lucas e à minha nora Suzan, pela compreensão da minha ausência em função do tempo de atenção que foi necessário para poder terminar esta dissertação.

Ao amigo e médico Cynégiro Sette dos Santos por possibilitar o equilíbrio da minha saúde, o que me proporcionou a condição para a realização desta dissertação.

Aos amigos Allan Poubel, Elinaldo Francelino, Fábio Pupo e Luciano pelo socorro dado várias vezes na área da informática.

Aos funcionários do PPTEC, Bráulio, Ivan, Jaqueline, Márcia, Sheila e outros por sua colaboração e zelo nos procedimentos administrativos.

Aos amigos do PPTEC, e a todos aqueles que, pessoal ou profissionalmente, contribuíram para a realização deste trabalho ou torceram pelo seu término.

"Afirmo muitas vezes que, se você medir aquilo de que está falando e expressar em números, você conhece alguma coisa sobre o assunto; mas, quando você não o pode exprimir em números, seu conhecimento é pobre e insatisfatório."

William Thompson (Lord Kelvin)

APLICAÇÃO DO ALGORITMO GENÉTICO DE CHAVES ALEATÓRIAS VICIADAS NO PROBLEMA DA MOCHILA NÃO LINEAR SEPARÁVEL

Marcelo Pinto Pereira

Orientador:

Leonardo Silva de Lima, D.Sc.

Coorientador:

Wagner Pimentel, D.Sc.

Resumo da Dissertação de Mestrado submetida ao Programa de Pós-Graduação em Tecnologia do Centro Federal de Educação Tecnológica Celso Suckow da Fonseca, CEFET/RJ, como parte dos requisitos necessários à obtenção do Título de Mestre em Tecnologia.

O problema da mochila não linear separável (NLKP) é uma variação do problema clássico da mochila no qual a função objetivo e/ou a restrição são não lineares e todas as funções são separáveis. A versão do NLKP quando a função objetivo é não convexa não tem sido muito estudada na literatura e não são conhecidos algoritmos exatos para encontrar soluções ótimas para aplicações práticas em um tempo computacional adequado. Procedimentos meta-heurísticos são usados nestes casos e uma recente estrutura algorítmica chamada Algoritmo Genético de Chaves Aleatórias Viciadas (BRKGA) foi utilizada ao longo deste trabalho a fim de resolver o NLKP. O procedimento BRKGA é inspirado no Algoritmo Genético de Chaves Aleatórias, onde cada cromossomo é tomado como um vetor de números reais gerados aleatoriamente no intervalo $[0, 1)$. As nossas experiências computacionais consistiram na implementação de um decodificador BRKGA e aplicá-lo a várias instâncias a partir da literatura ao NLKP. Resultados computacionais satisfatórios foram observados para pequenas e médias instâncias em comparação com uma heurística construtiva específica para o problema.

Palavras-chave:

Algoritmo genético de chaves aleatórias; Problema da mochila não linear; Meta-heurística

Rio de Janeiro

Julho 2015

APPLICATION OF BIASED RANDOM KEY GENETIC ALGORITHM ON NONLINEAR SEPARABLE KNAPSACK PROBLEM

Marcelo Pinto Pereira

Advisors:

Leonardo Silva de Lima, D.Sc.

Wagner Pimentel, D.Sc.

Abstract of dissertation submitted to Programa de Pós-Graduação em Tecnologia of Centro Federal de Educação Tecnológica Celso Suckow da Fonseca, CEFET/RJ, as partial of fulfillment of the requirements for the degree of Master in Technology.

The separable non-linear knapsack problem (NLKP) is a variation of the classical knapsack problem which the objective function or/and the constraint are non-linear and all of them are separable functions. The version of the NLKP when the objetive function is not convex has not been very studied in the literature and exact algorithms to find optimal solutions for practical applications in an appropriate computational time are not known. Metaheuristics procedures are used in these cases and the very recent framework called Biased Random Key Genetic Algorithms (BRKGA) was used throughout this work in order to solve the NLKP. The BRKGA procedure is inspired in the random key genetic algorithm where each chromosome is taken as a vector of randomly generated real numbers in the interval $[0, 1)$. Our computational experiments consisted of implementing a BRKGA decoder and applying it to several instances from the literature to the NLKP. Satisfactory computational results were observed in a small and medium instance in comparison with a specific constructive heuristic for the problem.

Keywords:

Genetic algorithm of random keys; Problem of non-linear knapsack; Metaheuristic

Rio de Janeiro

2015 July

Sumário

I Introdução **1**

II Problema da Mochila **4**

II.1 Contextualização histórica 4

II.2 Variantes 4

II.2.1 Problema da Mochila 0 - 1 5

II.2.2 Problema da Mochila Inteiro 5

II.2.3 Problema da Mochila Limitada 6

II.2.4 Problema da Mochila 0-1 Múltipla 6

II.2.5 Problema da Mochila Limitada Múltipla 7

II.2.6 Problema da Mochila Não Linear 7

II.2.7 Problema da Mochila Quadrático 8

III Problema da Mochila Não Linear Separável **10**

III.1 Motivação 10

III.2 Modelagem 11

III.3 Problemas $NLKP_1$ e $NLKP_2$ 12

III.3.1 Classe $NLKP_1$ 12

III.3.2 Classe $NLKP_2$ 13

IV Algoritmo Genético **14**

IV.1 Nomenclatura 16

IV.2 Algoritmo Genético Básico 16

IV.3 Características do AG 17

IV.4 Codificação do AG 18

IV.5 População Inicial 19

IV.6 Avaliação da População 20

IV.7 Estratégias Evolutivas 20

IV.8 Operadores Genéticos 22

IV.8.1 Seleção 22
IV.8.2 Cruzamento com Representação Binária 26
IV.8.3 Cruzamento com Representação Real 27
IV.8.4 Mutação com Representação Binária 31
IV.8.5 Mutação com Representação Real 32

V Algoritmo Genético de Chaves Aleatórias Viciadas 34
V.1 Algoritmo Genético de Chaves Aleatórias 34
V.1.1 Funcionamento do Algoritmo Genético de Chaves Aleatórias 35
V.1.2 Pseudocódigo Algoritmo Genético de Chaves Aleatórias 36
V.2 Algoritmo Genético de Chaves Aleatórias Viciadas 37
V.3 Parâmetros do BRKGA 40

VI Proposta de solução do NLKP utilizando o BRKGA: módulo decodificador 41
VI.1 Decodificador 41
VI.1.1 Decodificador $NLKP_1$ 42
VI.1.2 Decodificador $NLKP_2$ 46
VI.2 Busca Local 49

VII Resultados computacionais 53
VII.1 Classe $NLKP_1$ com Variáveis Inteiras 55
VII.2 Classe $NLKP_1$ com Variáveis Reais 57
VII.3 Classe $NLKP_2$ com Variáveis Inteiras 59
VII.4 Classe $NLKP_2$ com Variáveis Reais 61

VIIIConclusões 64

Referências Bibliográficas 66

Lista de Figuras

II.1 Função convexa. 8

III.1 Curva de saturação das vendas em função da publicidade. 11

IV.1 Fluxograma de um algoritmo genético simples. 16
IV.2 Exemplo de interpretação de números reais codificados de forma binária. 19
IV.3 Algoritmo Genético Geracional. 21
IV.4 Algoritmo genético *steady-state*. 22
IV.5 Método da roleta. 24
IV.6 Seleção por torneio com $k = 3$. 25
IV.7 *Crossover* de dois pontos. 27
IV.8 Método de cruzamento uniforme. 27
IV.9 Exemplo do processo de mutação. 32

V.1 Cruzamento uniforme parametrizado 36
V.2 Pseudocódigo do algoritmo genético de chaves aleatórias com reinicialização. 37
V.3 Transição geracional do BRKGA. 38
V.4 Fluxograma BRKGA. 39

VI.1 Algoritmo do Decodificador $NLKP_1$. 43
VI.2 Representação gráfica da solução do problema $NLKP_1$ com variáveis inteiras. 45
VI.3 Algoritmo do Decodificador $NLKP_2$. 47
VI.4 Busca Local. 50

VII.1 Simbologia para as propostas de solução dos problemas $NLKP$. 54
VII.2 Gráfico comparativo entre as médias das melhores soluções para o problema $NLKP_1$ com variáveis inteiras. 57
VII.3 Gráfico comparativo entre as médias das melhores soluções para o problema $NLKP_1$ com variáveis reais. 59
VII.4 Gráfico comparativo entre as médias das melhores soluções para o problema $NLKP_2$ com variáveis inteiras. 61

VII.5 Gráfico comparativo das melhores soluções para o problema $NLKP_2$ com variáveis reais. 63

Lista de Tabelas

IV.1 Relação entre termos da genética e AG. 16
IV.2 Adequação entre codificação cromossomial e tipos de problemas. 18
IV.3 Relação entre linguagem binária e numérica. 18

V.1 Parâmetros necessários para BRKGA 40

VI.1 Descrição dos elementos do pseudocódigo dos decodificadores. 42
VI.2 Exemplo da Aplicação do Decodificador $NLKP_1$ 44
VI.3 Exemplo da Aplicação do Decodificador $NLKP_1$ 46
VI.4 Exemplo da Aplicação do Decodificador $NLKP_2$ 48
VI.5 Exemplo da Aplicação do Decodificador $NLKP_2$ 49
VI.6 Exemplo da Aplicação da Busca Local $2-$opt 51
VI.7 Exemplo da Aplicação da Busca Local $2-$opt - Continuação 51
VI.8 Exemplo da Aplicação da Busca Local $2-$opt - Continuação 52
VI.9 Soluções Inicial e Resultante do Exemplo da Aplicação da Busca Local $2-$opt 52

VII.1 Médias dos valores das melhores soluções para o $NLKP_1$ utilizando variáveis inteiras. . 55
VII.2 Comparações entre as médias das melhores soluções para o $NLKP_1$ utilizando variáveis inteiras. 56
VII.3 Médias das melhores soluções para o $NLKP_1$ utilizando variáveis reais. 57
VII.4 Comparações entre as médias das melhores soluções para $NLKP_1$ utilizando variáveis reais. 58
VII.5 Médias das melhores soluções para o $NLKP_2$ utilizando variáveis inteiras. 60
VII.6 Comparações entre as médias das melhores soluções para o $NLKP_2$ utilizando variáveis inteiras. 60
VII.7 Médias das melhores soluções para o $NLKP_2$ utilizando variáveis reais. 61
VII.8 Comparações entre as médias das melhores soluções para o $NLKP_2$ utilizando variáveis reais. 62

Capítulo I - Introdução

O problema da mochila é estudado intensamente, do ponto de vista teórico devido, à sua estrutura de simples entendimento e alta complexidade computacional. Do ponto de vista prático, é bastante estudado em função da sua grande aplicabilidade em problemas da vida real, como por exemplo: no mercado financeiro em investimento de capital e bolsa de valores [51]; carregamento de carga calculando a melhor utilização da área disponível de *container* [3]; corte de estoque [4] que consiste em atender a uma demanda de itens a partir do corte de material com objetivo de minimizar perdas. Este problema pode aparecer também em diversos processos industriais, como por exemplo, elaborar planos de mínimo custo de produção para o problema do dimensionamento de lotes em pequenas fundições [11].

Podemos ilustrar o problema da mochila da seguinte maneira: imagine que um montanhista deve carregar a sua mochila selecionando itens, dentre vários disponíveis, e considerando a capacidade da mochila. A cada item é atribuído um valor de utilidade e o montanhista deve escolhê-los buscando maximizar o valor de utilidade total. Assim, o problema consiste em determinar os itens adequados que podem vir a ser alocados na mochila, respeitando as restrições de capacidade do compartimento da mesma buscando maximizar o valor de utilidade total. O problema da mochila é comprovadamente *NP hard*, [21], ou seja, o esforço computacional necessário para sua resolução aumenta exponencialmente com o tamanho da entrada.

Uma variante do problema clássico da mochila é o problema da mochila não linear. Esta variante considera a existência de funções não lineares, sejam elas na função objetivo e/ou nas restrições. Esta versão é ainda um problema com uma única restrição e possíveis limitantes superiores e/ou inferiores nas variáves de decisão. Podemos citar alguns trabalhos que estudaram o problema da mochila não linear como [2], [10], [30] e [57]. Em virtude da dificuldade de resolução do problema da mochila não linear geral, algumas hipóteses são frequentemente consideradas sobre as funções não lineares envolvidas no modelo e métodos heurísticos são desenvolvidos. Por exemplo, D'Ambrosio e Martello [2] criaram um conjunto de instâncias para o problema e desenvolveram um método heurístico assumindo que as funções não lineares são todas separáves.

Alternativas às modelagens matemáticas para os problemas de otimização combinatória têm aparecido na literatura, no decorrer das últimas décadas, soluções obtidas através de procedimentos heurísticos e meta-heurísticos. De acordo com [16] uma meta-heurística é um *framework* que

pode ser utilizado em diversos problemas de otimização efetuando somente pequenas modificações para adaptá-lo a um determinado problema. Em outras palavras, uma meta-heurística é um conjunto de ideias que podem ser usadas para definir métodos heurísticos aplicáveis a um grande grupo de problemas diferentes.

Algumas dessas heurísticas e meta-heurísticas são listadas a seguir conforme mostrado em [41]: Algoritmos Genéticos [18] abordada neste trabalho (AG), Busca Local Iterativa (ILS), *Ant Colony Optimization* (ACO), *Greedy Randomized Adaptative Search Procedure* (GRASP), *Simulated Annealing* (SA), Busca Tabu (BT), *Particle Swarm Optimization* (PSO), Método de Pesquisa em Vizinhança Variável (VNS), Algoritmos Transgenéticos (AT), Programação Genética (PG), Dinâmica de Sistemas (DS), *Beam Search* (BS), Sistemas Imunológicos Artificiais (SIA), Reconexão de Caminhos (RC), *Very Fast Local Search* (VFLS), Evolução Diferencial (ED), Colônia Artificial de Abelhas (CAA) e Algoritmo do Morcego (AM).

Algoritmos genéticos são métodos de otimização e busca inspirados nos mecanismos de evolução de uma população de seres vivos. Foram propostos por Holland [31] e popularizados por um dos seus alunos, Goldberg [24]. Esses algoritmos seguem o princípio da seleção natural e sobrevivência dos mais aptos.

Neste trabalho, objetivamos desenvolver a aplicação de um variante do algoritmo genético, recentemente projetado por Gonçalves e Resende [25], denominado *Biased Random-Key Genetic Algorithms* (BRKGA). As instâncias de teste serão as mesmas criadas e utilizadas por D'Ambrosio e Martello [2]. É importante frisar que todas estas instâncias envolvem somente funções separáveis. Assim, nesta dissertação, estudaremos somente os problemas na classe do problema da mochila não linear separável.

A metodologia de desenvolvimento da dissertação consistiu nos seguintes passos: (i) estudar os artigos de D'Ambrosio e Martello [2] e Gonçalves e Resende [25]; (ii) selecionar o conjunto de instâncias que serão utilizadas para teste; (iii) implementar um decodificador que conecta uma solução do BRKGA ao problema da mochila não linear; (iv) comparar os resultados obtidos pelo BRKGA com os resultados da literatura.

Vale ressaltar que não encontramos na literatura nenhum trabalho que trate da solução do problema da mochila não linear utilizando o AG e em especial utilizando o algoritmo genético com chaves aleatórias viciadas. A motivação para o estudo de uma aplicação heurística no problema da mochila não linear separável se deve também ao fato de quando pesquisamos na literatura o artigo *Heuristic algorithms for the general nonlinear separable knapsack problem* [2], verificamos que ele foi citado somente em dois artigos posteriormente, respectivamente em *Initial shipment decisions for new products at Zara* [20] e em *Special issue on knapsack problems and applications* [29], sendo que nesses dois artigos não encontramos nenhum desenvolvimento heurístico para a resolução do

problema proposto em [2].

Dividimos o texto em oito capítulos organizados com a seguinte estrutura:

No Capítulo II é apresentado o histórico do problema da mochila por intermédio de uma breve revisão bibliográfica e o resumo de variantes desse problema. Este resumo contém uma descrição sucinta dos problemas selecionados com suas respectivas formulações matemáticas segundo os trabalhos de Lin [38] e, Martello e Toth [43]. O objetivo desta descrição dos problemas da mochila é mostrar que o problema estudado, nesta dissertação, em relação aos lineares tem um alto grau de dificuldade exatamente devido a sua não linearidade na sua formulação matemática.

No Capítulo III mostramos um exemplo do problema da mochila não linear separável, retirado do artigo de D'Ambrosio e Martello [2]. A seguir, mostramos a formulação matemática do modelo NLKP e em consequência das suas condições de domínio mostramos as modelagens matemáticas como um problema de programação não linear ou como um problema de programação não linear inteira. Além disso, descrevemos as modelagens matemáticas dos problemas $NLKP_1$ e $NLKP_2$ que são especificamente os objetos principais do nosso estudo.

No Capítulo IV apresentamos a meta-heurística algoritmo genético (AG) [18], [24] e [31]. Mostraremos a ideia evolutiva usada pelo AG inspirada na natureza. Indicaremos as relações entre termos utilizados na biologia e os correspondentes usados em computação. Analisaremos um algoritmo genético básico do AG para nos familiarizarmos com os diversos passos do funcionamento do algoritmo. Avaliaremos as principais características do AG. Por fim, exploraremos o funcionamento dos diversos operadores genéticos do AG.

No Capítulo V apresentamos a plataforma BRKGA [54]. Inicialmente iremos analisar o funcionamento do precursor dessa classe de algoritmos genéticos, o Algoritmo Genético de Chaves Aleatórias (ou RKGA, do inglês *Random Key Genetic Algorithms*) descrito por Bean, em 1993. Em seguida, é mostrado o funcionamento do BRKGA e as suas diferenças em relação ao RKGA. Neste capítulo também apresentaremos o funcionamento básico de um BRKGA por intermédio de um fluxograma. Finalmente, serão mostrados os parâmetros utilizados no BRKGA.

No capítulo VI retratamos a proposta de solução do NLKP utilizando o BRKGA a partir da implementação do módulo do decodificador que converte uma solução do BRKGA em uma solução do problema abordado. Descreveremos ainda o procedimento de busca local desenvolvido.

No Capítulo VII, os resultados computacionais obtidos a partir da implementação do decodificador do BRKGA e das execuções dos procedimentos que representam a heurística proposta serão apresentados.

No Capítulo VIII são explicitadas as conclusões e sugestões de trabalhos futuros seguidos pelas Referências Bibliográficas utilizadas.

Capítulo II - Problema da Mochila

O problema da mochila, do inglês *knapsack problem* (KP), é um problema de otimização combinatória, cujo objetivo é alocar objetos em uma mochila de capacidade finita de modo a maximizar o valor econômico dos objetos presentes na mochila. O KP tem sido estudado devido à sua utilização imediata nas áreas industrial e financeira. Neste capítulo os conceitos do KP e algumas variantes são apresentados. O desenvolvimento do referencial teórico deste capítulo foi realizado com base em Bartholdi [13], Lin [38] e Martello [43].

II.1. Contextualização histórica

O problema da mochila apareceu na literatura pela primeira vez em 1957, em duas importantes publicações que foram escritas por Bellman [6] e por Dantzig [14]. George Dantzig é considerado o criador da área conhecida como Pesquisa Operacional e foi um dos desenvolvedores da programação linear. Ele mostrou uma versão contínua do problema da mochila que pode ser maximizada, escolhendo objetos com uma relação favorável entre custo e benefício. Richard Bellman, outro especialista em otimização discreta, descreveu como utilizar programação dinâmica para resolver o KP.

A partir de então, inúmeros algoritmos exatos e heurísticas para a solução do KP têm sido desenvolvidos e relatados na literatura. O KP tem sido intensamente estudado desde 1966 por Gilmore e Gomory [23]. Salkin e de Kluyver [48] realizaram o primeiro levantamento sobre as abordagens de solução para o KP em 1975, um referencial teórico sobre algoritmos exatos para KP foi revisto por Dudzinski e Walukiewicz [17]. Em 1990, Martello e Toth [43] efetuaram a compilação e elaboração de vários modelos de KP e seus algoritmos. Em 1994, Gerasch e Wang [22] pesquisaram os métodos para a solução de KP em arquitetura de computação paralela.

II.2. Variantes

Independente da aplicabilidade no dia a dia do KP, ele foi enunciado por razões teóricas, uma vez que este ocorre pela relaxação de vários problemas de programação inteira. Este problema pode ser apresentado em linguagem coloquial da seguinte maneira:

Um andarilho levará consigo apenas uma mochila para sua viagem. Essa mochila possui uma determinada capacidade e deve ser ocupada com diferentes tipos de objetos que lhe serão úteis

durante a viagem. Cada objeto preenche uma certa capacidade da mochila e tem um dado valor para o andarilho. Quais objetos devem ser levados pelo andarilho em sua mochila de forma a maximizar o valor do seu conteúdo?

Considere um conjunto de n objetos. É atribuído um peso w_j e um valor econômico p_j para cada um dos n objetos de modo que $j = 1, \ldots, n$. O problema da mochila consiste em alocar um subconjunto dos objetos em uma mochila com capacidade limitada C de forma a maximizar o valor econômico carregado na mochila.

Existem diversas variações dos problemas da mochila, iremos apresentar algumas dessas variações nas subseções seguintes.

II.2.1. Problema da Mochila 0 - 1

Este é o problema da mochila mais simples, no qual a mochila deve conter o melhor subconjunto dos objetos a serem selecionados, sendo que cada objeto pode ser escolhido no máximo uma vez. A sua representação matemática é dada por:

$$\text{Maximizar} \quad \sum_{j=1}^{n} p_j x_j \tag{II.1}$$

$$\text{Sujeito a} \quad \sum_{j=1}^{n} w_j x_j \leq C \tag{II.2}$$

$$x_j \in \{0, 1\}, \qquad j = 1, \cdots, n.$$

II.2.2. Problema da Mochila Inteiro

No problema da mochila inteiro as variáveis de decisão assumem valores no conjunto dos números inteiros não negativos e a modelagem clássica do problema é dada a seguir:

$$\text{Maximizar} \quad \sum_{j=1}^{n} p_j x_j \tag{II.3}$$

$$\text{Sujeito a} \quad \sum_{j=1}^{n} w_j x_j \leq C \tag{II.4}$$

$$x_j \in Z_+, \qquad j = 1, \cdots, n.$$

Segundo Yanasse *et al* [4] este tipo de problema da mochila está relacionado com o problema de corte de uma barra. Em alguns processos industriais há necessidade de cortar uma barra no sentido longitudinal, sem que um limite máximo de objetos seja determinado.

II.2.3. Problema da Mochila Limitada

Este problema é uma extensão do problema da mochila 0-1, na qual para cada item j existem u_j unidades disponíveis. A modelagem de programação inteira deste problema é dada por:

$$\text{Maximizar} \quad \sum_{j=1}^{n} p_j x_j \tag{II.5}$$

Sujeito a:

$$\sum_{j=1}^{n} w_j x_j \leq C \tag{II.6}$$

$$0 \leq x_j \leq u_j \qquad j = 1, \cdots, n$$

$$x_j \in Z^+ \qquad j = 1, \cdots, n$$

Em que a variável de decisão x_j indica o número de unidades do item j alocados a mochila. Um exemplo da aplicação deste problema pode ser visto no trabalho de Boleta *et al* [8].

II.2.4. Problema da Mochila 0-1 Múltipla

Outro caso que vamos considerar analisa quando várias mochilas são preenchidas e uma restrição adicional é imposta. Neste problema, além do conjunto de itens, cada qual com peso w_j e valor de retorno p_j, há um conjunto de mochilas, cada qual com capacidade C_i. Existe uma unidade de cada item e o objetivo é também maximizar o valor de retorno dos itens alocados as mochilas. A modelagem de programação inteira deste problema é dada por:

$$\text{Maximizar} \quad \sum_{i=1}^{m} \sum_{j=1}^{n} p_j x_{ij} \tag{II.7}$$

Sujeito a :

$$\sum_{j=1}^{n} w_j x_{ij} \leq C_i, \tag{II.8}$$

$$\sum_{i=1}^{m} x_{ij} \leq 1 \tag{II.9}$$

$$x_{ij} \in \{0, 1\}, \qquad i = 1, \cdots, m, \qquad j = 1, \cdots, n.$$

Em que a variável de decisão x_{ij} assume valor 1 se o item j for alocado a mochila i, e 0 caso

contrário. O primeiro conjunto de restrições assegura que cada mochila i não comporta mais que c_i unidades de peso, enquanto o segundo conjunto impede que um mesmo item j seja alocado a mais de uma mochila.

II.2.5. Problema da Mochila Limitada Múltipla

A modelagem de programação inteira deste problema é dado por:

$$\text{Maximizar} \quad \sum_{i=1}^{m}\sum_{j=1}^{n} p_j x_{ij} \tag{II.10}$$

Sujeito a:

$$\sum_{j=1}^{n} w_j x_{ij} \leq C_i, \tag{II.11}$$

$$\sum_{i=1}^{m} x_{ij} \leq u_j \tag{II.12}$$

$$x_{ij} \in Z^+, \qquad i = 1, \cdots, m, \qquad j = 1, \cdots, n.$$

Neste modelo, x_{ij} indica a quantidade de itens j alocados a mochila i.

II.2.6. Problema da Mochila Não Linear

Um problema da mochila não linear ou também denominado *Non Linear Knapsack Problem* (NLKP), de maneira geral, é um problema da mochila com função objetivo não linear e, ou restrição de recursos não lineares. Este problema da mochila também é conhecido como o problema de alocação de recursos não linear. Muitas versões do problema têm sido abordadas na literatura, mas a versão clássica é um problema de otimização não linear com uma única restrição, conforme o modelo a seguir:

$(NLKP)$

$$\text{Maximizar} \quad f(x) \tag{II.13}$$

Sujeito a:

$$g(x) \leq C \tag{II.14}$$

$$x \in D,$$

onde $x = (x_1, \cdots, x_n) \in \Re^n$, $f(x)$ e $g(x)$ são funções contínuas diferenciáveis, C é um valor positivo, $D \subseteq \Re^n$.

Segundo Bretthauer e Shetty [10] as variações no modelo NLKP incluem variáveis contínuas

ou inteiras, funções convexas ou não convexas, separáveis ou não separáveis e, em alguns casos, restrições adicionais especialmente estruturadas.

Normalmente, de acordo com Lin [38], a função objetivo do NLKP está contida em um conjunto de algumas funções côncavas crescentes, ao passo que as restrições da mochila consistem em algumas funções convexas crescentes.

Funções convexas e côncavas

Seja $A \subset \Re^n$ um conjunto convexo e $f : A \to \Re$. Diz-se que f é uma função convexa se para todos $x, y \in A$ e todo $\alpha \in [0, 1]$,

$$f(\alpha x + (1 - \alpha)y) \leq \alpha f(x) + (1 - \alpha)f(y) \tag{II.15}$$

Se a desigualdade for estrita para todos $x \neq y$ e $\alpha \in (0, 1)$, então f é estritamente convexa; f é côncava se $-f$ é convexa. A Figura II.1 mostra a situação descrita graficamente.

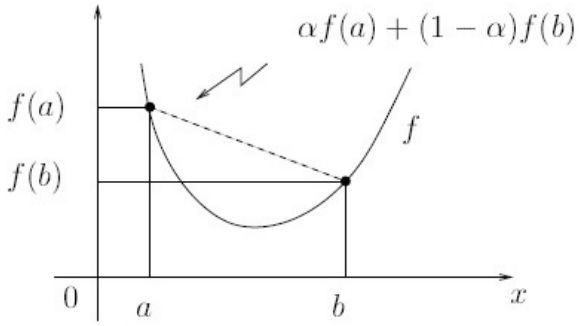


Figura II.1: Função convexa.

Fonte: [19]

Um problema de otimização é definido como convexo quando as funções objetivos e as restrições são funções convexas. Isso implica necessariamente que ótimos locais são também ótimos globais. No caso do problema de otimização abordado nesta dissertação, as funções envolvidas são não convexas, o que mostra um grau de dificuldade maior em resolver o problema uma vez que podem existir diversos ótimos locais.

II.2.7. Problema da Mochila Quadrático

Dentro do grupo de problemas NLKP, um dos mais estudados é o problema da mochila quadrático [38]. Este problema pode ser formulado matematicamente como segue:

$$\text{Maximizar} \qquad x^T Q x + x^T c \tag{II.16}$$

Sujeito a:

$$\sum_{j=1}^{n} w_j x_j \leq C \tag{II.17}$$

$$l_j \leq x_j \leq u_j, \qquad x_j \in Z_+ \quad \text{e} \quad j = 1, \cdots, n$$

$$x \in \Re^n, \qquad C \in \Re^n \quad \text{e} \quad Q \in \Re^{n \times n}$$

Contudo o objeto de estudo deste trabalho é o problema da mochila não linear separável [2], apresentado no próximo capítulo.

Capítulo III - Problema da Mochila Não Linear Separável

Uma das versões do problema da mochila não linear bastante estudada é o $NLKP$ quando $f(x)$ e $g(x)$ são funções separáveis [52]. Uma função $F(x_1, ..., x_n)$ é dita separável se pode ser escrita da seguinte forma:

$$F(x_1, ..., x_n) = \sum_{j=1}^{n} f_j(x_j) \tag{III.1}$$

Neste capítulo, abordamos na Seção III.1 a motivação para o desenvolvimento do estudo do $NLKP$. Na Seções III.2 apresentamos a modelagem e as condições de integralidade do problema separável $NLKP$ e na Seção III.3 as classes $NLKP_1$ e $NLKP_2$ a serem estudadas são apresentadas. As principais referências para o desenvolvimento deste capítulo foram Ambrosio e Martello [2], Kurt *et al* [10] e Lin [38].

III.1. Motivação

O problema da mochila não linear normalmente surge em contexto de alocação de recursos. O $NLKP$ tem várias aplicações tais como: seleção de portfólio, amostragem estratificada, planejamento de produção, assistência médica, modelos financeiros, sistemas de computador, gestão de modelos de redes de fila e distribuição de recursos [10] e [34].

Um exemplo prático do $NLKP$ é mostrado em [2], quando um recurso limitado de uma quantidade C (como um orçamento de publicidade) tem que ser particionado para diferentes categorias $j = 1, \cdots, n$ (tais como anúncios de diferentes produtos). A função objetivo é o retorno esperado total em termos de vendas em todas as categorias. Em dado momento ocorre a saturação na qual um acréscimo de publicidade já não leva a um aumento significativo das vendas (Figura III.1).

Assim, x_j indica a quantidade de propaganda alocada à categoria $j = 1, \cdots, n, \quad f_j(x_j)$ indica o retorno de vendas esperado em termo de vendas da categoria j, e $g_j(x_j)$ o custo correspondente para a publicidade.

Neste problema utilizamos a função $f_j(x_j)$ mostrada a seguir:

$$f_j(x_j) = \frac{c_j}{1 + b_j e^{-a_j(x_j + d_j)}} \tag{III.2}$$

Observe que as funções $f(x) = \sum_{j=1}^{n} f_j(x_j)$ e $g(x) = \sum_{j=1}^{n} g_j(x_j)$ são separáveis.

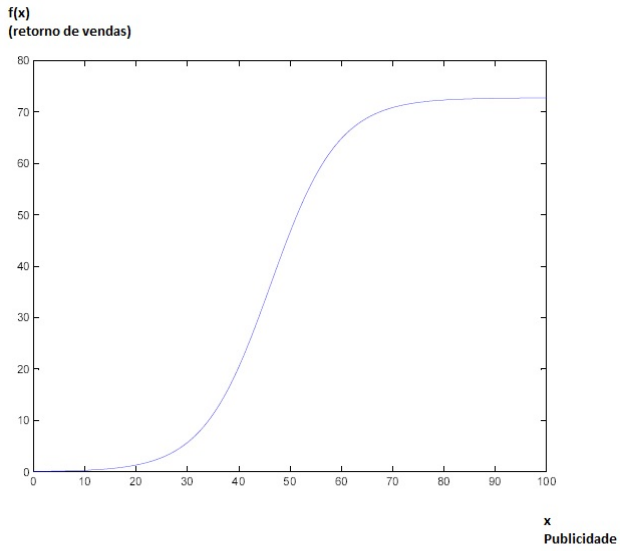


Figura III.1: Curva de saturação das vendas em função da publicidade.

Estudos anteriores do problema na literatura mostram a existência de vários problemas da mochila não linear separáveis, entretanto, a maior parte dos algoritmos propostos nesses trabalhos são voltados para resolver casos de funções côncavas e/ou convexas [30].

Os problemas $NLKP_1$ e $NLKP_2$ mostrados a seguir nas subseções III.3.1 e III.3.2 retirados de [2] têm uma representação que não é côncava nem convexa, uma situação rara nesse tipo de modelo, pouco explorado pelos pesquisadores da área de otimização, exceto em poucos artigos. Dentro desses poucos artigos relacionados a essas funções que não são côncavas nem convexas podemos citar o trabalho sobre otimização global geral de Horst e Tuy [32].

III.2. Modelagem

O domínio D do modelo $NLKP$ geral pode incluir limites e requisitos de integralidade em parte das variáveis e, dependendo dessa premissa da integralidade das variáveis, esse problema pode ser formulado como um problema de programação não linear inteira puro ou um problema de programação não linear inteira misto, em que $f_j(x_j)$ e $g_j(x_j)$ são funções não lineares positivas crescentes e a única restrição é a capacidade da mochila. As funções $f_j(x_j)$ e $g_j(x_j)$ podem ser não convexas e não côncavas. O problema então pode ser definido como:

$(NLKP)$

$$\text{Maximizar} \quad \sum_{j=1}^{n} f_j(x_j) \tag{III.3}$$

Sujeito a:

$$\sum_{j=1}^{n} g_j(x_j) \leq C \tag{III.4}$$

$$0 \leq x_j \leq u_j, \qquad \forall j \in N = \{1, \cdots, n\}.$$

$$x_j \quad \text{inteiro}, \qquad \forall j \in \overline{N} \subseteq N$$

Logo $\overline{N}$ é o subconjunto de N no qual todas as variáveis são inteiras. Neste trabalho serão considerados dois casos a serem resolvidos:

- todas as variáveis são reais, ou seja, $\overline{N} = \emptyset$,
- todas as variáveis são inteiras, ou seja, $\overline{N} = N$.

Nesse problema supomos que as funções $f(x)$ e $g(x)$ são separáveis.

III.3. Problemas $NLKP_1$ e $NLKP_2$

Nesta seção apresentaremos duas classes de problema da mochila não linear: uma com restrição linear e a outra com restrição não linear, que são o objeto principal do nosso estudo. As classes $NLKP_1$ e $NLKP_2$ são constituídas de variáveis exclusivamente inteiras ou exclusivamente reais.

III.3.1. Classe $NLKP_1$

Esta classe de problema denominada problema da mochila não linear separável com restrição linear $NLKP_1$, utiliza o modelo específico $NLKP$, e as suas premissas de integralidade para as suas variáveis, apresentado na seção anterior onde a restrição $g(x)$ é uma função peso linear crescente. A capacidade C foi definida por intermédio da fórmula $C = \frac{1}{2}\sum_{j=1}^{n} w_j u_j$, esta expressão é usualmente utilizada na literatura do problema da mochila a fim de produzir casos desafiadores. Então, o problema pode ser formulado matematicamente da seguinte forma:

($NLKP_1$)

$$\text{Maximizar} \quad \sum_{j=1}^{n} \frac{c_j}{1 + b_j e^{-a_j(x_j + d_j)}} \tag{III.5}$$

Sujeito a:

$$\sum_{j=1}^{n} w_j x_j \leq C \tag{III.6}$$

$$0 \leq x_j \leq u_j, \qquad \forall j \in N = \{1, \cdots, n\},$$

$$x_j \in Z^+, \qquad \forall j \in \overline{N} \subseteq N,$$

onde a_j é um parâmetro aleatório no intervalo $[0.1, 0.2]$, b_j é um parâmetro aleatório no intervalo

$[0, 100]$, c_j é um parâmetro aleatório no intervalo $[0, 100]$, d_j é um parâmetro aleatório no intervalo $[-100, 0]$, $u_j = 100$, w_j é um coeficiente aleatório no intervalo $[1, 100]$, conforme apresentado em [2].

III.3.2. Classe $NLKP_2$

Na classe de problema $NLKP_2$ chamado problema da mochila separável com restrição não linear, a função objetivo $f(x)$ é modelada também pela função separável mostrada em III.4. A restrição $g(x)$ é uma função peso côncava crescente e analogamente ao $NLKP_1$ a capacidade C foi definida como $\frac{1}{2}\sum_{j=1}^{n} g_j(u_j)$ para gerar casos desafiadores. O problema pode ser expresso matematicamente da seguinte forma:

($NLKP_2$)

$$\text{Maximizar} \quad \sum_{j=1}^{n} \frac{c_j}{1 + b_j e^{-a_j(x_j + d_j)}} \tag{III.7}$$

Sujeito a:

$$\sum_{j=1}^{n} (\sqrt{p_j x_j + q_j} - \sqrt{q_j}) \leq C \tag{III.8}$$

$$0 \leq x_j \leq u_j, \qquad \forall j \in N = \{1, \cdots, n\},$$

$$x_j \in Z^+, \qquad \forall j \in \overline{N} \subseteq N,$$

onde a_j é um parâmetro aleatório no intervalo $[0.1, 0.2]$, b_j é um parâmetro aleatório no intervalo $[0, 100]$, c_j é um parâmetro aleatório no intervalo $[0, 100]$, d_j é um parâmetro aleatório no intervalo $[-100, 0]$, p_j é um coeficiente aleatório no intervalo $[1, 20]$, q_j é um parâmetro aleatório no intervalo $[1, 20]$, $u_j = 100$, conforme apresentado em [2].

Adiante, aplicaremos o BRKGA em diversas instâncias do problema $NLKP1$ e $NLKP2$. Neste trabalho, optamos por aplicar o BRKGA ao problema da mochila não linear separável fundamentalmente por três motivos, primeiro porque não foi encontrado nenhum trabalho que utilize o BRKGA, para o problema NLKP abordado. Em segundo lugar o NLKP é um problema complexo de otimização combinatória e o BRKGA tem robustez e flexibilidade, segundo Gonçalves e Resende [54], para proporcionar uma boa solução aproximada para essa classe de problemas. Finalmente o terceiro motivo foi em função de observarmos bons resultados na literatura com a utilização do método BRKGA em vários problemas de otimização, por exemplo, em [46].

Capítulo IV - Algoritmo Genético

Neste capítulo descreveremos os principais conceitos da meta-heurística algoritmo genético (AG). As referências mais importantes para o desenvolvimento deste capítulo foram Goldberg [24], Holland [31], Linden [40], Melanie [42], Pimentel [45] e Michalewicz [56].

Algoritmo genético é um algoritmo evolucionista de busca fundamentado na teoria da evolução das espécies de Charles Darwin [15], descrita no seu livro ”A origem das espécies e a seleção das espécies”, sendo que a edição original é datada de 1859. Chamamos computação biologicamente inspirada ou computação evolucionista a inteligência computacional baseada na teoria da evolução das espécies. Essa importante teoria construída por Darwin pela observação da natureza, afirma de uma forma geral que os indivíduos mais adaptados ao meio que vivem, tendem a sobreviver e ter descendência.

A evolução não é um processo totalmente aleatório. A variação genética em si é estocástica, porém, a seleção natural não. Um indivíduo somente consegue sobreviver e reproduzir-se devido a características que se harmonizam com o seu meio ambiente local. Essas características, inerentes a cada indivíduo, dependem principalmente de possuir, ou não, genes favoráveis à sobrevivência de uma maneira geral, herdados dos seus ancestrais.

A computação evolucionista se divide em [40]: algoritmos genéticos, programação genética e estratégias evolucionistas.

É importante observar que o AG apesar do nome, não é um método exato. O AG é uma heurística, e segundo Lindem, “heurísticas são algoritmos polinomiais que não têm garantia nenhuma sobre a qualidade da solução encontrada, mas que usualmente tendem a encontrar a solução ótima ou ficar bem perto dela” [40].

O autores Holland [31] e Goldberg [24] perceberam que poderiam programar códigos para simular a evolução natural, de forma, a imitar o processo como os seres vivos se reproduzem para repassar seus genes. Esse método de busca e otimização baseia-se na seleção natural que assegura que todos os indivíduos pertencentes a um ecossistema competem entre si por recursos escassos. Logo, os indivíduos que não conseguem êxito no ecossistema tendem a ter uma menor descendência gerando consequentemente uma menor disseminação dos seus genes ao longo das gerações seguintes.

No algoritmo genético, o especialista cria códigos para representar o problema e seus parâmetros,

mais ou menos como os cromossomos de um indivíduo representam as suas características. Então, a partir da geração de uma população inicial de cromossomos que simbolizam possíveis soluções do problema, toda a população é avaliada por intermédio de uma função que determina a aptidão de cada cromossomo, esses cromossomos são ordenados de acordo com a valoração em relação ao grau de *fitness*.

Considerando a teoria darwiniana são escolhidos os cromossomos para a próxima geração com maior probabilidade de seleção para os mais aptos. Através da reprodução, os cromossomos selecionados transferem suas características para os seus descendentes por intermédio dos operadores genéticos de cruzamento e mutação. Após vários ciclos de evolução, a população deverá tender na média a conter indivíduos mais aptos. O processo descrito é iterado até que se seja atendido um critério de parada do AG. Os principais critérios de parada são: tempo, número máximo de gerações, diversidade genética e se a aptidão requerida é conhecida, podemos trabalhar com a opção de um erro máximo admissível.

A motivação para a utilização do AG é derivada da sua grande aplicabilidade na área de otimização. Segundo Michalewicz [56], temos as seguintes aplicações dos AG na área de problemas de otimização relacionados com: Problema do Caixeiro Viajante, Problema de Roteamento de Veículos, Layout de Circuitos, Consulta de Banco de Dados, Distribuição de Fiação, Funções Matemáticas, Negócios, etc. Linden [40] cita também no seu livro, Algoritmos Genéticos, algumas áreas onde o AG tem aplicações tais como:

- alocação de recursos - escalonamento de horários e escala de tarefas;
- setor elétrico - planejamento de expansão, alocação de capacitores e *unit commitment*;
- bioinformática - engenharia reversa de redes de regulação genética e filogenética;
- setor petrolífero - inversão sísmica, otimização de estratégias de produção e agendamento de produção.

Em especial na área específica do problema da mochila linear inteira, a utilização do algoritmo genético como proposta de solução aproximada tem sido uma alternativa viável. A seguir mostramos alguns exemplos na literatura. Hristakeva e Shrestha resolveram o problema da mochila 0-1 com algoritmos genéticos [33], Taskiran criou um algoritmo genético aperfeiçoado para o problema da mochila [53], Bortfeld e Winter implementaram um algoritmo genético para o problema da mochila bidimensional com peças retangulares [9], Khuri *et al* escreveram sobre o problema da mochila 0-1 múltipla e o algoritmo genético [36], Szeto e Lo implementaram uma aplicação de um algoritmo genético adaptado no problema da mochila financeira [51] e, Camargo e Mattioli utilizaram o problema da mochila como ferramenta para resolver o problema de planejamento de produção em pequenas fundições [11].

IV.1. Nomenclatura

Os AG são inspirados na genética, logo existem vários termos que têm sentidos correlatos entre a biologia e a computação. Os principais termos são mostrados na tabela IV.1 a seguir.

Tabela IV.1: Relação entre termos da genética e AG.

Linguagem Biológica	AG
População	Conjunto de cromossomos ou soluções
Cromossomo	Conjunto de genes, indivíduo
Gene	Característica
Alelo	Valor
Locus	Posição
Genótipo	Estrutura
Fenótipo	Conjunto de parâmetros
Seleção natural	Sobrevivência do mais apto
Crossover	Cruzamento ou recombinação de cromossomos
Mutação	Modifica posição ou valor de um gene de um cromossomo
Fitness	Valor da função objetivo
Geração	Ciclo
Epistasia	Interação entre genes do cromossomo

Fonte: [40].

Nesta dissertação usaremos de uma maneira intercambiável os sentidos dos termos biológicos e do AG mostrado na tabela anterior.

IV.2. Algoritmo Genético Básico

A figura IV.1 representa um esquema simplificado do procedimento de um AG básico.

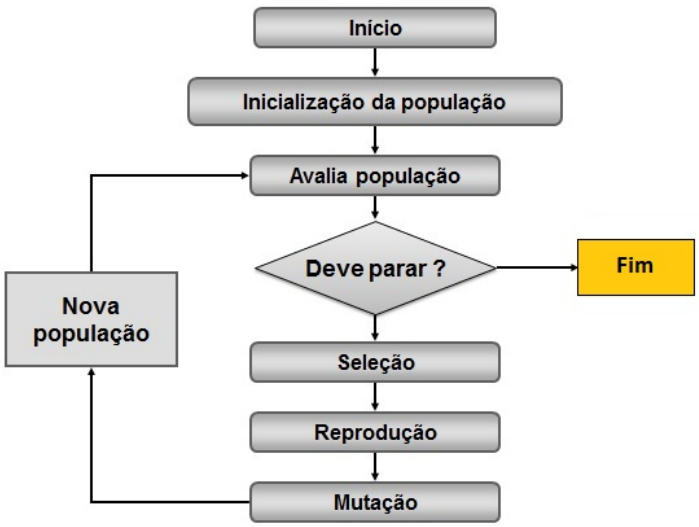


Figura IV.1: Fluxograma de um algoritmo genético simples.

Os passos do funcionamento desse fluxograma são mostrados a seguir, de acordo com Melanie [42]:

1. Iniciar com uma população gerada de cromossomos com n genes se o problema possui n variáveis independentes, soluções candidatas ao problema.
2. Calcular a aptidão de cada cromossomo da população.
3. Selecionar de acordo com sua adequação dois cromossomas para serem os pais; quanto melhor o *fitness*, mais chances de ser selecionado. A seleção para cruzamento dos cromossomos é feita com substituição, logo um mesmo cromossomo pode ser selecionado mais de uma vez para se tornar um pai.
4. Com uma probabilidade de cruzamento, p_c, fornecida por um operador de cruzamento, cruzamos o par de pais em um ponto de escolha aleatório (escolhido com probabilidade uniforme) para gerar dois descendentes.
5. Efetuar a mutação de dois descendentes com uma probabilidade de mutação, p_m, para cada *locus* e colocar os cromossomos resultantes na nova população.
6. Substituir a população atual pela nova população.
7. Avaliar os novos cromossomos.
8. Se o tempo terminr ou a solução atenda ao critério de parada pare, caso contrário, vá para o passo 2, a fim de gerar uma nova população.

IV.3. Características do AG

Os AG são algoritmos evolucionários com métodos que pertencem às técnicas de busca aleatórias guiadas [40]. Devido a essa característica eles não ficam presos em um máximo local caso o problema codificado seja de maximização, o que ocorre nos métodos de gradiente (*Hill Climbing*). Outras importantes características do AG em relação a outros métodos de busca são:

- a população inicial é um conjunto de soluções possíveis;
- o algoritmo tem como resposta uma população de soluções ordenadas segundo o critério de aptidão e não uma única solução;
- é um método de otimização estocástico e não exato, ou seja, uma determinada população raramente repetirá os mesmos resultados para um mesmo problema;
- AG é uma boa escolha para resolução de problemas NP-completos, problemas de maximização de funções multimodais[1], ou seja, problemas nos quais os algoritmos exatos são lentos ou não conseguem chegar em uma solução adequada.

[1] Uma função multimodal, tem diversas inflexões em sua superfície, o que caracterizam múltiplos pontos de máximo ou mínimo.

IV.4. Codificação do AG

Um cromossomo define um ponto do espaço de busca de um problema e a sua estrutura será manipulada pelo algoritmo. É necessário traduzir as informações do cromossomo na linguagem que o computador entenda. A codificação dos dados no cromossomo é uma parte fundamental do AG e junto com a função de aptidão é o que estabelece a ligação do problema a ser resolvido com o AG.

Na maioria das implementações do AG, a maneira mais comum de fazer a codificação cromossomial é utilizando uma sequência binária. Entretanto existem outros tipos de representações, as principais são: numéricas, ordinais, de valores categóricos e híbridas. A codificação usada para o cromossomo depende do tipo do problema analisado e também do tamanho das instâncias do problema, a Tabela IV.2 a seguir, mostra algumas codificações de cromossomos para o AG que são adequadas para a resolução das classes relacionadas de problemas de otimização. Concluímos que cada problema tem a sua representação cromossomial mais adequada. Nesta dissertação, para representar um cromossomo dos problemas $NLKP1$ e $NLKP2$, utilizaremos codificação numérica.

Tabela IV.2: Adequação entre codificação cromossomial e tipos de problemas.

Codificação	Problema
Binária	Numéricos, inteiros
Números reais	Numéricos
Permutação de símbolos	Baseados em ordem
Símbolos repetidos	Grupamento

Fonte: [12]

Considere o problema de maximizar a seguinte função:

$$f(x) = x^3 \tag{IV.1}$$

onde x é inteiro, $x \in [0, 5]$. Podemos representar as soluções do problema através de cromossomos com 3 genes utilizando a codificação binária conforme mostrado a seguir na Tabela IV.3.

Tabela IV.3: Relação entre linguagem binária e numérica.

Cromossomo	Código binário	Indivíduo (x)
C_0	000	0
C_1	001	1
C_2	010	2
C_3	011	3
C_4	100	4
C_5	101	5

Fonte: o próprio autor.

Para representarmos números reais como números binários precisamos saber a precisão (p),

e o domínio da variação $[I_i, S_i]$ de cada uma das variáveis do cromossomo. Por intermédio desses dois parâmetros, em conjunto, definimos quantos *bits* serão necessários por variável. Consideremos uma variável x_i, que utilize k *bits*, dentro de uma faixa de variação $[I_i, S_i]$, a equação que define a precisão máxima dessa variável, adaptada de [40] é dada por:

$$p = \frac{S_i - I_i}{2^k - 1} \tag{IV.2}$$

A conversão de um número binário contido em um cromossomo para o número real correspondente dentro do domínio estabelecido, é dado pela equação:

$$x_r = I_i + n_i p \tag{IV.3}$$

em que k é quantidade de *bit* do cromossomo, x_r é um número real, n_i é o número inteiro correspondente ao *string*, S_i é o limite superior da faixa de operação, I_i é o piso da faixa de operação da variável e p é a precisão mostrada na equação IV.2.

Dentro de um cromossomo binário podemos ter mais de um número real, bastando para isso colocar um *string* ao lado do outro. Consideremos 3 *strings* de 6 *bits* justapostos lado a lado, 100101110001
111001, representando 3 variáveis: x_1, x_2 e x_3, sendo as faixas de operação das variáveis, respectivamente: $[-1, 1]$, $[0, 2]$ e $[0, 1]$.

A Figura IV.2 a seguir, mostra como transformar os valores binários codificados nos valores reais correspondentes de acordo com as equações IV.2 e IV.3.

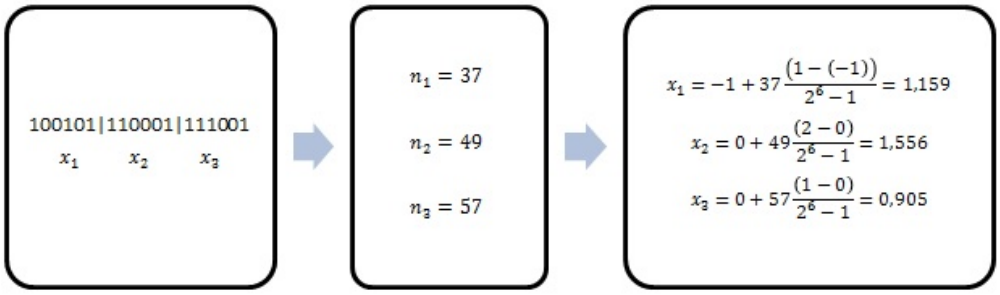


Figura IV.2: Exemplo de interpretação de números reais codificados de forma binária.

Fonte: o próprio autor.

IV.5. População Inicial

É uma população gerada com n indivíduos, geralmente de forma aleatória, para iniciar o processo evolutivo. O tamanho da população inicial precisa ser definido de acordo com as características

de cada problema podendo ficar entre algumas dezenas e poucas centenas de elementos. Podemos especificar restrições na geração da população inicial, por exemplo, permitindo somente soluções viáveis.

IV.6. Avaliação da População

A cada geração representada por uma população, os indivíduos são avaliados de acordo com a sua adaptação ao ambiente. Isso é feito através de uma função de avaliação. Essa função possibilita comparar os indivíduos, ordenando-os de acordo com as suas aptidões, assim podemos aplicar os mecanismos de seleção para a criação das novas gerações. A classificação dada pela medida da aptidão das soluções pode ser realizada em ordem crescente ou decrescente, respectivamente, caso o problema seja de maximização ou minimização.

IV.7. Estratégias Evolutivas

O AG tem duas estratégias de reposição de população que se destacam na literatura, conhecidas como AG geracional e AG não geracional (*steady-state*) que são descritos a seguir.

Geracional

O algoritmo geracional tem como característica principal a troca de toda a população a cada geração. Esta substituição total da população gera o inconveniente da possibilidade da perda de material genético de boa qualidade. Um processo de seleção elitista, que consiste em manter na próxima geração os melhores indivíduos da geração anterior, é adotado, muitas vezes, para evitar esta desvantagem. O pseudocódigo apresentado a seguir na Figura IV.3 mostra a estratégia evolucionista geracional sem o uso da seleção elitista.

```
1  Início
2  Inicialize a população P aleatoriamente
3  Avalie indivíduos na população P
4  repita
5      repita
6          Selecione dois indivíduos em P
7          Aplique o operador de recombinação com probabilidade p_c
8          Aplique o operador de mutação com taxa p_m
9          Insira novos indivíduos em P'
10     até
11     ;
12     A população P' completa
13     Avalie indivíduos na população P'
14     P ← P'
15 até
16 ;
17 Critério de parada satisfeito
18 Fim
```

Figura IV.3: Algoritmo Genético Geracional.

Fonte: [37].

Steady-state

A característica principal do algoritmo *steady-state* é a geração de somente um indivíduo de cada vez (ou dois, por ser mais conveniente para o operador de cruzamento). Cada indivíduo gerado é avaliado, e caso a sua aptidão seja melhor do que o pior indivíduo da lista de classificação, o pior será descartado, sendo substituído pelo novo indivíduo gerado. Existem outros critérios para inserção de novos indivíduos na população, por exemplo, inserir indivíduos aleatoriamente em vez de substituir os piores pais. Esse procedimento, embora pareça ser contraproducente minimiza a convergência genética[2]. Na página a seguir mostramos a Figura IV.4 que representa o pseudocódigo desse algoritmo.

[2]Convergência genética ocorre quando somente os melhores indivíduos se reproduzem. Neste caso, a população tende a ser constituída de indivíduos semelhantes, faltando portanto, diversidade a essa população para um melhor desenpenho do AG.

```
1  Início
2  Inicialize a população P aleatoriamente
3  Avalie indivíduos na população P
4  Ordene a população P de acordo com a aptidão
5  repita
6  |  Selecione operador genético
7  |  Selecione indivíduo(s) para reprodução
8  |  Aplique operador genético
9  |  Avalie indivíduo(s) gerados
10 |  Selecione o indivíduo f para sobreviver
11 |  Se f é melhor que o pior elemento de P
12 |  Então Insira f em P de acordo com o seu ranking
13 até
14 ;
15 Critério de parada satisfeito
16 Fim
```

Figura IV.4: Algoritmo genético *steady-state*.

Fonte: [37].

IV.8. Operadores Genéticos

A forma mais simples de AG envolve três tipos de operadores: seleção, cruzamento e mutação. Estes operadores são respectivamente instrumentos para simulações computacionais de fenômenos da natureza, tais como, a seleção das espécies (sobrevivência do mais apto), a reprodução sexuada e a mutação.

Dois conceitos fundamentais para entender a finalidade dos AG são diversificação ou exploração, que é a técnica de percorrer pontos novos no espaço de soluções viáveis sendo fundamental para explorar diferentes regiões no espaço de busca e a intensificação também chamada explotação (do inglês *exploitation*) que é o método de procurar melhores pontos do espaço de busca a partir do aproveitamento de informações de pontos que já foram visitados. Um bom algoritmo de otimização global deve ter essas duas características, o AG por intermédio da seleção aplica a intensificação e pelo cruzamento em conjunto com a mutação utiliza a diversificação.

IV.8.1. Seleção

Este operador escolhe cromossomos na população para cruzamento sendo, portanto, o principal responsável pela variabilidade genética do AG. Na natureza, quanto mais adaptados os indivíduos, mais chances têm de sobreviver até a idade reprodutiva e participarem do processo reprodutivo. No AG este é um dos critérios de seleção, ou seja, quanto maior o valor da aptidão do indivíduo, maior é a sua probabilidade de realizar cruzamentos. O valor da aptidão de um indivíduo é dado pela equação:

$$p_i = \frac{f_i}{\sum_{i=1}^{n} f_i} \tag{IV.4}$$

em que i corresponde ao índice que identifica o indivíduo na população, n é o tamanho da população, f_i é a avaliação do indivíduo i e p_i é a aptidão do indivíduo i na população atual, ou seja, a probabilidade do indivíduo i ser escolhido para cruzamento.

A possibilidade de escolha de um indivíduo para recombinação é maior em função do valor do seu melhor valor de *fitness*. A maneira como é realizada a seleção no código pode aumentar ou diminuir a velocidade da busca da melhor solução, e também a convergência genética devido a definir os indivíduos que irão gerar e depois da recombinação pertencerem a próxima geração. Esse artifício da seleção é denominado pressão seletiva ou intensidade de seleção. O conceito de pressão seletiva utilizado em genética é a variação na aptidão média proporcionada pelo método de seleção. A equação da pressão seletiva (P) [7] é:

$$P = \frac{M^* - M}{\sigma} \tag{IV.5}$$

em que M representa a aptidão média da população, M^* é o valor esperado da aptidão média após a seleção e σ é o desvio padrão dos valores de aptidão da população antes da seleção.

A seleção em AG pode ser implementada de várias formas que serão descritas a seguir.

Seleção por Roleta ou Seleção Proporcional

Esse método de seleção dos progenitores é adotado por uma grande parte de estudiosos da área dos AG. Ele simula o comportamento da seleção natural que atua nos seres vivos, pois na natureza os indivíduos mais aptos têm maior probabilidade de serem escolhidos para a reprodução. No algoritmo as soluções com maior valor da função objetivo também têm maior chance de repassar seus genes para os seus descendentes. É importante observar que os indivíduos considerados menos aptos não são descartados na natureza e também no AG, caso isso acontecesse, haveria uma rápida convergência das soluções.

Esse método tem dois casos que geram problemas: a existência de super indivíduos e a competição próxima. Por exemplo, a situação na qual um indivíduo tem uma aptidão muito alta em relação aos outros da população, ocorre uma rápida convergência genética. No caso em que as aptidões entre indivíduos distintos são semelhantes, mas não iguais, a pressão seletiva pode ser menor do que a desejada.

Uma roleta virtual faz uma correspondência entre o índice de aptidão dos indivíduos e um determinado pedaço da roleta, seja em graus (entre 0° e 360°), ou em porcentagem (entre 0 e 100%). O ato de girar a roleta é totalmente aleatório e o indivíduo selecionado será aquele onde a roleta

parar. A roleta é girada quantas vezes for necessário dependendo do tamanho da população, e esses indivíduos sorteados pela roleta são selecionados para participar da geração da nova população. A seguir, na Figura IV.5, mostramos um exemplo de uma população composta de 10 indivíduos e suas respectivas relações com as regiões dos setores circulares correspondentes às suas aptidões.

Indivíduo	Aptidão	Setor circular (%)	Setor circular (°)
1	6	10	36
2	5	8	30
3	9	15	54
4	6	10	36
5	8	13	48
6	3	5	18
7	5	8	30
8	1	2	6
9	7	12	42
10	10	17	60

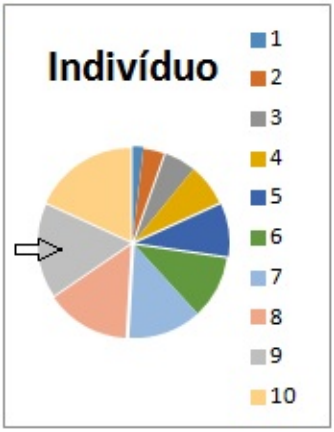


Figura IV.5: Método da roleta.

Fonte: o próprio autor.

Seleção por Torneio

Nesse método de seleção são escolhidos certo número de indivíduos da população e depois é realizada uma competição entre eles usando como critério, para vencer a disputa, aquele que possui a melhor aptidão.

Para a realização desse torneio é necessário definir um parâmetro chamado tamanho do torneio, k. Este parâmetro indica quantos indivíduos dentro da população serão escolhidos aleatoriamente para competir. Não existe um limite máximo para k, se por exemplo for escolhido para k um valor igual ao tamanho da população, o vencedor do torneio será o melhor indivíduo, entretanto o menor valor de k é 2, pois caso contrário, obviamente não existiria torneio.

Consideremos na Figura IV.6 a população dada de 8 indivíduos e suponhamos que estamos utilizando um torneio de tamanho $k = 3$. Neste caso, determinamos os participantes do torneio, sorteando três vezes um número compreendido entre 1 e 8. No exemplo mostrado na Figura IV.6, são sorteados inicialmente 1, 2 e 3 que correspondem aos indivíduos que têm aptidões respectivamente a 50, 200 e 20. O cromossomo x_2 detém a maior aptidão neste torneio. Ele é então escolhido para ser submetido ao operador de cruzamento.

Na Figura IV.6 é mostrado, dentro de retângulos vermelhos e marcados em negrito, todos os indivíduos selecionados que serão utilizados como pais na próxima geração. Neste exemplo, os cromossomos x_4 e x_5 são chamados super indivíduos devido aos altos valores de aptidão. Estes super indivíduos, no caso da utilização do método da roleta, rapidamente dominariam a população.

Já com a utilização do método do torneio, observamos que esses super indivíduos, em função das possibilidades iguais dada a todos os concorrentes pelo sorteio, não dominam integralmente a população seguinte, porém naturalmente ganham todos os torneios nos quais participam.



Figura IV.6: Seleção por torneio com $k = 3$.

Fonte: adaptado de [40].

A forma de escolher os indivíduos que irão participar do torneio, como já foi descrito, é totalmente aleatória, não havendo nenhum privilégio para os indivíduos com os melhores *fitness*.

Ranking

Na seleção por *ranking*, segundo Linden [40], é necessário, inicialmente, ordenar do maior para o menor os indivíduos da população analisada de acordo com sua função de aptidão. E, de posse desse *ranking* fazer uma correlação usando a equação:

$$E(i,t) = min + (max - min)\frac{rank(i,t) - 1}{n - 1}. \tag{IV.6}$$

Em que min corresponde ao pior *ranking*, max corresponde ao melhor *ranking*, n significa o número de indivíduos da população e $rank(i,t)$ corresponde a posição do indivíduo i na população mantida pelo AG na geração t.

Com a representação do mapeamento linear da população podemos usar quaisquer dos métodos de seleção, vistos anteriormente, para definir quais indivíduos serão utilizados para a recombinação. Ao utilizarmos a seleção por *ranking* evitamos a convergência precoce e a prevalência de um super indivíduo, temos também a vantagem de manter a pressão seletiva no mesmo patamar, da primeira até a última geração, contudo o AG demora um tempo maior para convergir na direção de uma boa solução. Em função da diversidade genética mantida neste código, o algoritmo tende a percorrer uma faixa maior do espaço de busca ficando menos propenso a determinar máximos locais.

Outra maneira de usar o método de *ranking* é utilizando a função de mapeamento exponencial, de acordo com [40], modelada por:

$$E(i) = \frac{1 - e^{-kt}}{c} \tag{IV.7}$$

onde i é o *ranking* invertido do indivíduo (do pior para o melhor) e c é uma constante.

Esse processo de normalização exponencial tem por finalidade fazer com que a pressão seletiva fique em um patamar mais elevado, pois aumenta a diferença entre o melhor e o pior indivíduo. Temos então a eliminação do problema do super indivíduo e evidentemente não temos a homogeneidade que a normalização linear dada pela equação IV.6 proporcionava às soluções. O que ocorre nesse caso é que a não linearidade dada pela equação IV.7, aplicada nas soluções ranqueadas, diminui a diversidade genética.

Seleção Local

Nesse método, também denominado seleção por vizinhança, cada indivíduo está inserido em um ambiente composto por uma vizinhança, que pode ser definida arbitrariamente. Os limites dessa vizinhança determinam com quais cromossomos um determinado indivíduo poderá realizar cruzamento.

No caso da seleção local, a metade dos pais da nova geração é selecionada aleatoriamente, para isso é utilizado quaisquer dos métodos mostrados até o momento. O restante dos pais necessários para a reprodução são escolhidos dentro das fronteiras em que esses indivíduos estão no espaço de busca.

A vizinhança tem as seguintes características: uma estrutura física que pode ter qualquer número de dimensões e uma distância máxima entre o indivíduo e a fronteira da vizinhança específica no qual ele está inserido.

IV.8.2. Cruzamento com Representação Binária

Após os dois progenitores serem selecionados pelo operador de seleção, entra em ação o operador de cruzamento. Embora os operadores de cruzamentos mostrados nesta subseção também sejam utilizados na representação não binária, para efeitos didáticos, iniciaremos explicando especificamente os operadores de cruzamento com codificação binária. Existem vários tipos de operadores de cruzamento e apresentaremos alguns a seguir.

Cruzamento de 1 Ponto

Este é o operador de cruzamento mais simples. Ele escolhe aleatoriamente um ponto entre dois *locus*, ou seja, em dois cromossomos que serão utilizados como pais e subsequentemente troca a posição dos *locus* que ficaram antes e depois deste ponto selecionado, criando, dessa forma, dois cromossomos descendentes. Por exemplo as sequências binárias 100100 e 111111 podem ser trocadas depois do terceiro *locus* de cada uma, produzindo os dois descendentes 100111 e 111100. O

operador de *crossover* imita de uma maneira grosseira o processo biológico de recombinação entre dois organismos de cromossomos simples.

Cruzamento de 2 Pontos

Neste operador de cruzamento selecionamos dois pontos nos cromossomos que serão os progenitores e, posteriormente, trocamos os genes dentro desses pontos entre os dois cromossomos; gerando dessa forma dois cromossomos descendentes. Na Figura IV.7 temos um exemplo desse tipo de cruzamento:

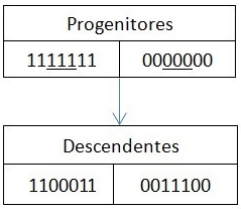


Figura IV.7: *Crossover* de dois pontos.

Cruzamento Uniforme

Este operador de cruzamento funciona nos cromossomos selecionados para a reprodução da seguinte maneira: para cada gene do cromossomo-pai é sorteado um número; 0 ou 1. Caso o valor sorteado seja 1, o filho 1 recebe o valor do gene correspondente ao gene do pai 1 e o filho 2 ganha o valor do gene do pai 2. Analogamente, se o valor sorteado for 0, as atribuições descritas são invertidas, ou seja, o filho 1 recebe o gene correspondente à mesma posição do pai 2 e o filho 2 ganha o gene do pai 1 relativo à posição atual. A Figura IV.8 mostra um exemplo deste tipo de cruzamento:

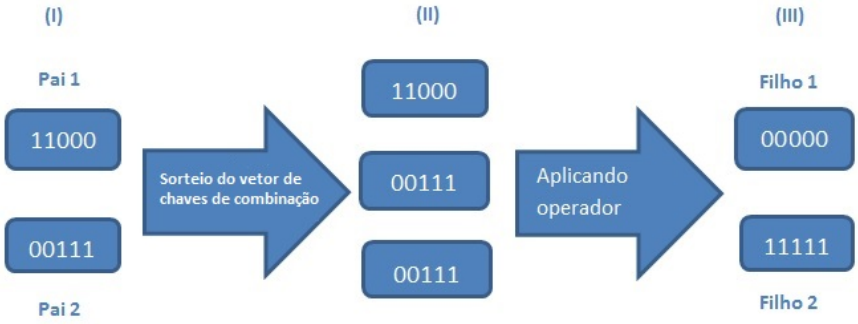


Figura IV.8: Método de cruzamento uniforme.

Fonte: adaptado de [40].

IV.8.3. Cruzamento com Representação Real

Existem diversos tipos de operadores de cruzamento específicos para a representação real, ou de ponto flutuante. Essa codificação é adequada para atender a problemas que usam variáveis de

um domínio contínuo. O cromossomo neste caso é um vetor do $\Re^n$ onde n é o número de variáveis do problema $x = (x_1, x_2, x_3, ..., x_n)$. A premissa mais importante no momento de modelar o cromossomo é que a representação deve se adequar ao problema e não o contrário.

Existem várias vantagens com relação ao uso da representação real, apontaremos duas, nesse momento: permite que o computador use o máximo de precisão que pode fornecer; também, o cromossomo tem o tamanho mínimo para o problema, no qual cada gene passa a representar uma das variáveis de decisão.

Os tipos de operadores analisados nesta subseção são apresentados de acordo com [28].

Sejam dois cromossomos P_1 e P_2 onde:

$$P_1 = (p_1^1, p_1^1, ..., p_n^1)$$

$$P_2 = (p_1^2, p_2^2, ..., p_n^2)$$

Então, baseado-se nessas expressões apresentaremos nas subseções a seguir alguns tipos de operadores de *crossover* reais [28], [37] e [40].

Cruzamento Uniforme ou *Crossover Flat*

O funcionamento deste operador se baseia em estabelecer um intervalo fechado para cada posição do cromossomo que se deseja gerar. O intervalo é limitado pelo menor e pelo maior valor do *locus* correspondente ao *locus* do descendente a ser criado. De posse desses intervalos, valores aleatórios que pertencem a esses intervalos são escolhidos para formar o cromossomo descendente, ou seja, nesse tipo de cruzamento é produzido um filho $F = (f_1, f_2, ..., f_n)$ nos quais os valores f_i são selecionados aleatoriamente no intervalo $[p_i^1, p_i^2]$. Então: $p_i^1 \leq f_i \leq p_i^2$

Cruzamento Simples

Este operador de cruzamento é semelhante ao operador de um ponto da representação binária. Escolhemos aleatoriamente um determinado ponto de corte i, no intervalo entre os pontos definidos pelos índices $[1, n-1]$. A seguir, efetuamos a troca do material genético entre os cromossomos P_1 e P_2. Tomamos os valores de um pai à direita do ponto de corte e do outro pai à esquerda.

$$F_1 = (p_1^1, p_2^1, ..., p_i^1, p_{i+1}^2, ..., p_n^2)$$

$$F_2 = (p_1^2, p_2^2, ..., p_i^2, p_{i+1}^1, ..., p_n^1)$$

Cruzamento Aritmético

Inicialmente definimos um valor arbitrário para o parâmetro $\lambda \in [0, 1]$ e são gerados dois filhos F_1 e F_2:

$$F_1 = (f_1^1, f_2^1, ..., f_n^1)$$

$$F_2 = (f_1^2, f_2^2, ..., f_n^2)$$

onde:

$$f_i^1 = \lambda p_i^1 + (1 - \lambda)p_i^2$$

$$f_i^2 = \lambda p_i^2 + (1 - \lambda)p_i^1$$

Cruzamento *Blended Crossover* (BLX - α)

Neste cruzamento é gerado um filho $F_1 = (f_1^1, f_2^1, ..., f_n^1)$, onde f_i é um número escolhido randomicamente no intervalo $[p_{min} - I\alpha, p_{max} + I\alpha]$. Considere $p_{min} = min(p_i^1, p_i^2)$, $p_{max} = max(p_i^1, p_i^2)$, $I = p_{max}p_{min}$ e α é uma constante, normalmente escolhida no intervalo $[0, 0.5]$. Observamos que para $\alpha = 0$ temos um cruzamento uniforme.

Cruzamento Linear

Neste tipo de cruzamento são gerados três filhos

$F_1 = (f_1^1, ..., f_n^1)$
$F_2 = (f_1^2, ..., f_n^2)$
$F_3 = (f_1^3, ..., f_n^3)$

onde os elementos de cada um são definidos da seguinte maneira:

$f_i^1 = \frac{1}{2}p_i^1 + \frac{1}{2}p_i^2$
$f_i^2 = \frac{3}{2}p_i^1 - \frac{1}{2}p_i^2$
$f_i^3 = -\frac{1}{2}p_i^1 + \frac{3}{2}p_i^2$

Cruzamento Discreto

Neste cruzamento é gerado um filho $F_i = (f_1^i, ..., f_n^i)$ onde f_i é um valor selecionado randomicamente dentro do conjunto $\{p_i^1, p_i^2\}$. Este *crossover* é análogo ao utilizado na codificação binária denominado uniforme, ele foi o operador escolhido por Toso e Resende [54] após uma adaptação para funcionar semelhantemente com uma escolha dirigida parametrizada de Spears e Jong [50] para utilização no BRKGA. No Capítulo V apresentaremos com maiores detalhes este processo.

Cruzamento de Linha Estendida

Este procedimento de recombinação consiste em gerar cada gene do cromossomo do filho da seguinte maneira: $f_i = p_i^1 + \alpha(p_i^2 - p_i^1)$ onde α é um parâmetro randômico no intervalo $[-0.25, 1.25]$. Um tipo de cruzamento um pouco diferente do cruzamento de linha estendida, conhecido como cruzamento intermediário estendido. É aquele no qual o valor de α é variável, escolhido aleatoriamente, para cada f_i, ou seja, $f_i = p_i^1 + \alpha_i(p_i^2 - p_i^1)$.

Cruzamento Heurístico de Wright

Neste tipo de operador é escolhido um número aleatório α no intervalo fechado [0,1], em seguida, é selecionado o cromossomo mais apto entre os cromossomos P_1 e P_2. Consideremos que seja P_1 o mais apto, então, cada gene do cromossomo do filho é gerado por:

$$f_i = \alpha(p_i^1 - p_i^2) + p_i^1$$

O *crossover* heurístico [55], e o operador aritmético são especialmente indicados para problemas de otimização numérica com restrições, em que a região viável é convexa. Isso se deve ao fato de que se P_1 e P_2 são viáveis, combinações convexas entre P_1 e P_2 também serão viáveis.

Cruzamento Linear BGA (*Breeder Genetic Algorithm*)

Este operador atua de forma que cada posição f_i do cromossomo a ser gerado seja resultado do uso da equação:

$$f_i = p_i^1 \pm rang_i.\gamma.\Lambda$$

em que:

$$\Lambda = \frac{p_i^2 - p_i^1}{||P_1 - P_2||}$$

Na expressão para calcular f_i o sinal negativo é usado com uma probabilidade de 0.9. O valor de $rang_i$ na maior parte dos casos é escolhido através de $0.5(b_i - a_i)$. O cálculo de γ é resultado de:

$$\gamma = \sum_{k=0}^{15} \alpha_k 2^{-k}$$

onde $\alpha_k \in \{0, 1\}$ é determinado randomicamente com probabilidade $p = \frac{1}{16}$, se $\alpha_k = 1$.

Cruzamento de Busca de Linha

Este operador, denominado na língua inglesa de *Line search*, inicialmente gera um filho que é a média aritmética entre os pais, $F_1 = 0.5(p_1^1 + p_1^2, ..., p_n^1 + p_n^2)$. Considerando P_1 como o pai com maior aptidão entre os genitores, se o cromossomo F_1 for mais apto que P_1 então, o filho gerado será definido como F_1. Caso contrário, efetuamos um novo cruzamento gerando um filho F_2 mais próximo do pai P_1, da seguinte forma:

$$F_2 = (p_1^1 + 0.5(p_1^1 - p_1^2), ..., p_i^n + 0.5(p_n^1 - p_n^2))$$

Por fim, escolhemos o melhor filho entre os dois gerados.

IV.8.4. Mutação com Representação Binária

A mutação é um operador exploratório que tem por finalidade aumentar a diversidade da população, pois introduz, de acordo com uma pequena taxa de probabilidade, material genético na população. Este operador entra em ação após a formação dos filhos, em alguns casos, invertendo aleatoriamente algum dos *bits* destes cromossomos de acordo com uma taxa de mutação.

Por exemplo, na Figura IV.9, mostrada na próxima página, o cromossomo 1001011 sofre mutação de duas formas; primeiro é modificado no seu quarto gene da esquerda para direita e, segundo, também são trocadas as componentes que ocupam a terceira posição do cromossomo. A mutação pode ocorrer para cada posição da sequência de *bits* com um certo grau de probabilidade. No caso dos AG a mutação geralmente é pequena, da ordem de 0.5% [40]. Uma probabilidade de mutação muito baixa pode ocasionar perda ou falta de diversidade na população, e no caso de uma taxa elevada de mutação pode levar o algoritmo a um comportamento anômalo, dificultando a convergência genética. A mutação pode ocorrer de diversas formas. Podemos citar os casos a seguir:

- mutação por indivíduo, em que cada cromossomo pode sofrer no máximo uma mutação em um gene aleatório;
- mutação por *bit*, na qual cada gene do cromossomo pode sofrer uma mutação;
- mutação por troca, nos quais os valores de dois genes escolhidos aleatoriamente são trocados;
- mutação invertida, semelhante à mutação por *bit*, porém os genes escolhidos recebem o valor oposto ao que tinham anteriormente.



Figura IV.9: Exemplo do processo de mutação.

Fonte: o próprio autor.

IV.8.5. Mutação com Representação Real

Para melhor entendimento dos métodos de mutação mostrados a seguir, adotaremos que $f_i \in [a_i, b_i]$ como o gene no qual irá incidir a mutação. Consideremos que esse gene pertence ao cromossomo $F_1 = (f_1, ..., f_n)$ e após a mutação o seu valor será f_i^*.

Mutação Aleatória

Este é o operador de mutação mais simples. Neste caso, escolhemos de forma aleatória um gene do cromossomo F_1 e em seguida o valor de f_i^* é determinado também aleatoriamente dentro do intervalo $[a_i, b_i]$. Os limites para a escolha de f_i devem ser preestabelecidos e representam o seu espaço de busca.

Mutação Não Uniforme

Seja gen uma certa geração e $maxgen$ o número máximo de gerações, então ao aplicarmos este operador de mutação, teremos a seguinte expressão matemática:

$$f_i^* = \begin{cases} f_i + \triangle(gen, b_i - f_i) & \text{se} \quad \tau = 0, \\ f_i - \triangle(gen, f_i - a_i) & \text{se} \quad \tau = 1 \end{cases}$$

onde:

- $\triangle(gen, y) = y(1 - r^{(1 - \frac{gen}{maxgen})^b})$
- τ escolhido aleatoriamente e o seu valor pode ser 0 ou 1
- r é um número aleatório no intervalo $[0, 1]$
- b é um parâmetro definido pelo usuário.

Mutação de Uma Unidade

A mutação efetuada por este operador gera um filho pela adição ou subtração aleatória de uma unidade da variável x_i, também escolhida randomicamente.

Além dos operadores de mutação reais mostrados, ainda podemos citar os seguintes operadores de mutação de acordo com [37]: *Small-Creep*, *Big-Creep* e o operador de Muhlenbein.

Capítulo V - Algoritmo Genético de Chaves Aleatórias Viciadas

Neste capítulo iremos descrever os principais conceitos dos Algoritmos Genéticos de Chaves Aleatórias Viciadas. O BRKGA é um *framework* que foi desenvolvido por Resende e Toso em 2011. As referências mais importantes para o desenvolvimento deste capítulo foram Bean [5], Gonçalves e Resende [25], Matsumoto e Nishimura [44], Jong e Spears [50] e, Resende e Toso [54]. A principal motivação para o uso do BRKGA nesse trabalho se deve ao fato de que ele tem obtido sucesso em diversos problemas de otimização nas mais diversas áreas segundo Resende [46], tais como, ajuste automático de parâmetros em heurísticas, empacotamento, engenharia de produção, escalonamento, leilões combinatórios, otimização em redes, otimização global contínua, recobrimento, telecomunicações e transportes.

V.1. Algoritmo Genético de Chaves Aleatórias

O pesquisador James C. Bean [5], em 1994, publicou um artigo no qual apresentou um algoritmo genético com uma forte representação do espaço de soluções viáveis, utilizando chaves aleatórias dentro do $\Re^n$. Esse algoritmo foi utilizado para resolver problemas de otimização combinatória tais como: sequenciamento de máquinas, alocação de recursos e designação quadrática.

A aplicação de fato desses algoritmos se estende a um leque muito mais vasto de problemas. Ao abordar tais problemas, os algoritmos genéticos normalmente têm dificuldade em manter viabilidade dos cromossomos-pais/cromossomos-filhos quando ocorre o cruzamento. Essa meta-heurística tem uma característica importante que resolve essa situação com o uso de vetores de permutação de chaves alatórias para representar soluções viáveis. A representação com chaves aleatórias codificam uma solução com números aleatórios no intervalo $[0, 1)$. Esses valores são usados como uma espécie de chave para decodificar a solução.

Bean [5] criou um módulo chamado decodificador que é um algoritmo determinístico. Este decodificador toma como entrada uma solução viável do problema de otimização na forma de um vetor e relaciona a ele uma solução na qual o valor da função objetivo do problema possa ser calculada. Portanto, o decodificador é utilizado para mapear um vetor de chaves aleatórias em uma solução do problema em questão e calcular o valor da função objetivo.

V.1.1. Funcionamento do Algoritmo Genético de Chaves Aleatórias

A dinâmica evolutiva de um RKGA segue a mesma lógica fundamentada por Darwin [15] , que afirma que os organismos mais bem adaptados ao meio têm maiores chances de sobrevivência do que os menos adaptados, produzindo uma maior quantidade de descendentes.

Considere a população inicial, denotada por P, e composta de $|P| = p$ vetores. Cada vetor contendo n chaves (ou n posições), que são geradas de forma aleatória e uniforme pelo algoritmo. Na geração K é calculado o custo de cada solução e em seguida são particionadas as soluções em dois conjuntos: um elite e outro não elite. Assume-se que o conjunto elite, denotado por P_e, deve ser menor que o não elite e deve conter as melhores soluções.

Assim, adotamos $|P_e| = p_e < \frac{p}{2}$, onde p_e é a cardinalidade dos vetores elite. O conjunto dos vetores elite da geração K é copiado fielmente e integralmente para a próxima geração denominada $K + 1$, com o intuito de evoluir a população. Nesta fase do algoritmo, observamos uma característica darwinista, pois através dessa estratégia elitista são preservadas as melhores soluções para a geração subsequente.

Diferentemente do AG tradicional, que efetua a operação de mutação junto com o cruzamento, o RKGA implementa esta mutação por intermédio da introdução de uma pequena população de mutantes p_m, gerados por vetores de chaves aleatórias dentro da geração $K + 1$. É importante frisar que um cromossomo mutante é um vetor de chaves aleatórias gerado da mesma forma que uma solução da população inicial é gerada.

Para completar a população da geração $K+1$, é necessário gerar uma quantidade de descendentes igual a $p - p_e - p_m$. Os indivíduos restantes necessários para completar a população que está sendo iterada são gerados utilizando o cruzamento uniforme parametrizado de Spears e Jong [50], no qual indivíduos são, dois a dois, selecionados aleatoriamente para realizar a operação de cruzamento a partir da população inteira e com reposição. Observe que este procedimento permite repetições na seleção de pais e, portanto, um indivíduo pode gerar mais de um filho.

A Figura V.1 a seguir, ilustra o funcionamento do procedimento criado por Spears e Jong, em que os vetores $P1$ e $P2$ representam os indivíduos pai e o vetor F representa o filho gerado pelo cruzamento. O elemento f_i, que é o i-ésimo componente do cromossomo filho, tem probabilidade ρ de herdar um alelo do pai $P1$, e tem probabilidade $1 - \rho$ de receber o alelo do pai $P2$. Na literatura, o parâmetro ρ é normalmente fixado em 0.7 como, por exemplo, em [47].

A Figura V.1 mostra um exemplo de como é realizado o cruzamento de dois cromossomos de chaves aleatórias compostos de quatro componentes. Temos os dois pais representados respectivamente pelos cromossomos $P1$ e $P2$ com valores reais arbitrados no intervalo $[0.1)$ para cada um dos seus genes. Considerando $\rho = 0.7$, observe que o cromossomo F recebe o gene do cromossomo

$P1$ com probabilidade 0.7 e do outro cromossomo $P2$, com probabilidade $1 - \rho = 0.3$. Um vetor de números aleatórios pertencente ao intervalo $[0.1)$, denominado V, com 4 componentes, é gerado de forma a representar os lançamentos de uma moeda viciada. Quando uma entrada de V é um número menor ou igual a 0.7, o cromossomo F recebe o componente correspondente ao cromossomo $P1$ na posição onde o cruzamento está ocorrendo. Caso contrário, recebe o componente do cromossomo $P2$.

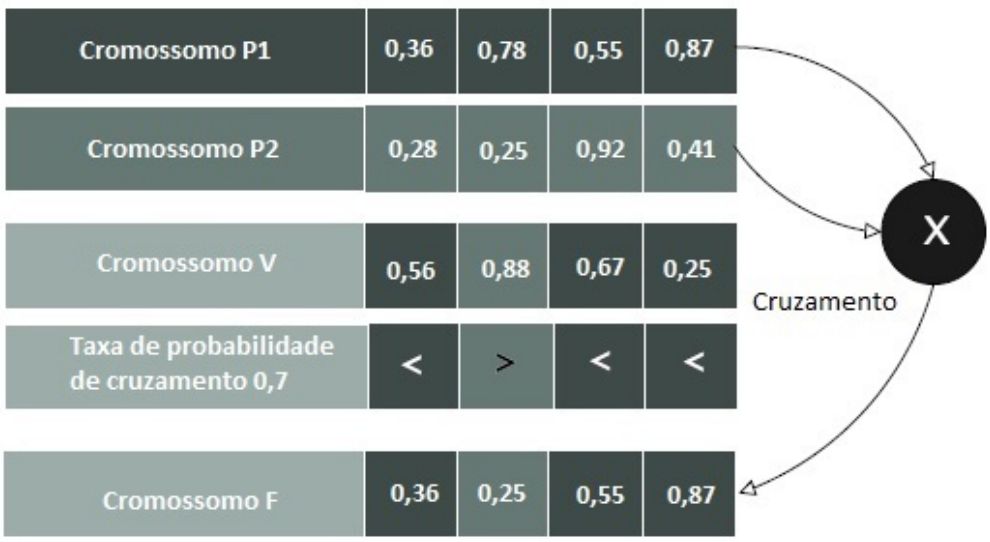


Figura V.1: Cruzamento uniforme parametrizado

Fonte: adaptado de [47].

V.1.2. Pseudocódigo Algoritmo Genético de Chaves Aleatórias

A seguir, na Figura V.2, temos o pseudocódigo do RKGA de Bean [5] onde:

$$x \in \Re^n, \quad \text{e} \quad f : X \to \Re$$

```
   Entrada: |P|,|P_e|, n e ρ
 1 Inicialize o valor da melhor solução encontrada: f* ← ∞;
 2 enquanto (critério de parada não for satisfeito) faça
 3     Gere a população P com vetores de n chaves alatórias;
 4     enquanto (critério de reinicialização não for satisfeito) faça
 5         Avalie o custo de cada solução nova em P;
 6         Particione P em dois conjuntos: P_e e P_ē ;
 7         Inicialize a população da próxima geração: P+ ← P_e;
 8         Gere o conjunto de mutantes P_m, cada mutante com n
 9         chaves aleatórias;
10         Adicione P_m à população da próxima geração: P+ ← P+ ∪ P_m;
11         para i ← 1 até |P|-|P_e|-|P_m| faça
12             Escolha o pai P_1 aleatoriamente de P;
13             Escolha o pai P_2 aleatoriamente de P;
14             para j ← 1 até n faça
15                 Sorteie um número aleatório ρ
16                 se (ρ > 0,7) então
17                     F[j] ← P_1[j];
18                 senão
19                     F[j] ← P_2[j];
20                 Adicione o filho F à população da próxima geração:
21                 P+ ← P+ ∪ F;
22             Atualize a população: P ← P+;
23             Ache a melhor solução χ+ em P:
24             χ+ ← argmin {f(χ)|χ ∈ P};
25             se f(χ+) < f* então
26                 χ* ← χ+;
27                 f* ← f(χ+);
28 Saída: =X*;
```

Figura V.2: Pseudocódigo do algoritmo genético de chaves aleatórias com reinicialização.

Fonte: adaptado de [46].

V.2. Algoritmo Genético de Chaves Aleatórias Viciadas

O algoritmo genético com chaves aleatórias viciadas, BRKGA [25], é um método heurístico utilizado para resolver problemas de otimização combinatória. A meta-heurística de busca global BRKGA é normalmente aplicada a problemas *NP hard*, para os quais não temos um algoritmo específico eficiente para resolução. A criação do BRKGA foi baseada no RKGA [5]. As modificações efetuadas no BRKGA em relação ao RKGA estão ligadas à maneira como as soluções são combinadas para gerar descendentes, aumentando o protagonismo das soluções com maiores aptidões de cada geração. Em função dessas diferenças entre os dois algoritmos observamos um melhor desempenho no BRKGA [26].

O que diferencia o BRKGA do RKGA é a maneira como os pais são selecionados para cruzamento, e como este cruzamento é efetuado.

Nos dois algoritmos, ou seja, no RKGA e no BRKGA, os pais são escolhidos aleatoriamente e com reposição. No BRKGA, cada indivíduo é gerado através do cruzamento de um pai escolhido do conjunto elite, P_e, e o outro, do conjunto não elite da população, $P_{\bar{e}}$, vigente; como mostrado na Figura V.3. Ainda no BRKGA, em algumas situações, o segundo pai pode ser selecionado levando-se em consideração a população inteira. Entretanto, no caso do RKGA, ambos os pais são escolhidos da população inteira.

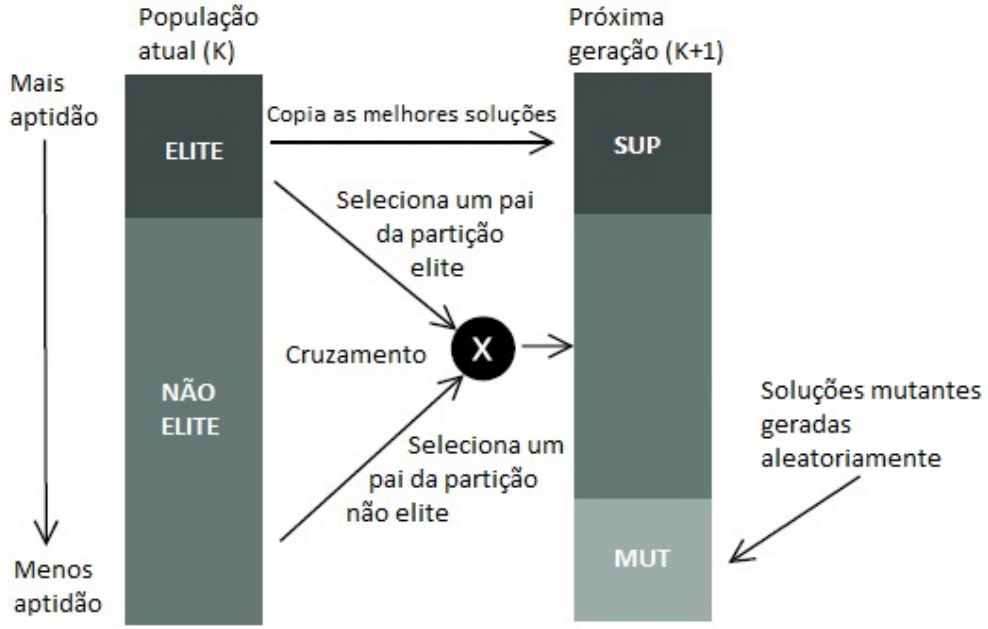


Figura V.3: Transição geracional do BRKGA.

Fonte: adaptado de [47].

O algoritmo de Bean [5] e o de Gonçalves e Resende [25] utilizam o cruzamento uniforme parametrizado de Spears e Jong [50]. A diferença do algoritmo proposto por Gonçalves e Resende [25] para o de Bean [5] é a maneira como a seleção dos cromossomos pai é implementada.

Considere novamente os cromossomos $P1$ e $P2$ da Figura V.1 que por intermédio do cruzamento uniforme parametrizado geram o filho F. No BRKGA, o $P1$ é sempre um vetor da partição elite, no caso do RKGA não existe essa restrição.

O procedimento do BRKGA, assim como já foi descrito para o RKGA, tem três características essenciais que especificam os algoritmos genéticos [54]:

- um determinado código cromossômico usando um vetor de n chaves aleatórias ou alelos no intervalo de $[0.1)$, onde o valor de n corresponde ao número de variáveis do problema de otimização;
- um processo evolutivo bem definido adotando cruzamento parametrizado uniforme [50], para gerar descendentes e assim desenvolver a população;

- a introdução de novos cromossomos, chamados mutantes, em substituição do operador de mutação normalmente encontrado em algoritmos genéticos.

Tais características simplificam e padronizam a meta-heurística com um conjunto de tarefas autônomas nas quais apenas uma depende do problema da decodificação do cromossomo, ou seja, usando-a para construir uma solução para o problema básico de otimização, a partir do qual o valor da função objetivo ou *fitness* pode ser computada.

Analogamente, ao decodificador do RKGA , o decodificador do BRKGA é usado para mapear indiretamente o espaço cromossômico contínuo $[0.1)^n$ dentro do espaço de soluções viáveis do problema de otimização. Ao contrário do que ocorre no algoritmo genético de uma forma geral, a viabilidade pode ser facilmente encontrada de uma geração a outra, graças a essa codificação padronizada, desde que todos os descendentes e mutantes introduzidos no processo evolutivo também sejam vetores de chaves aleatórias viáveis.

O BRKGA cria e, posteriormente desenvolve uma população contendo exatamente p cromossomos, cada um com n chaves aleatórias, para certo número de gerações até encontrar um critério de parada. Esse processo evolutivo é ilustrado na FiguraV.4:

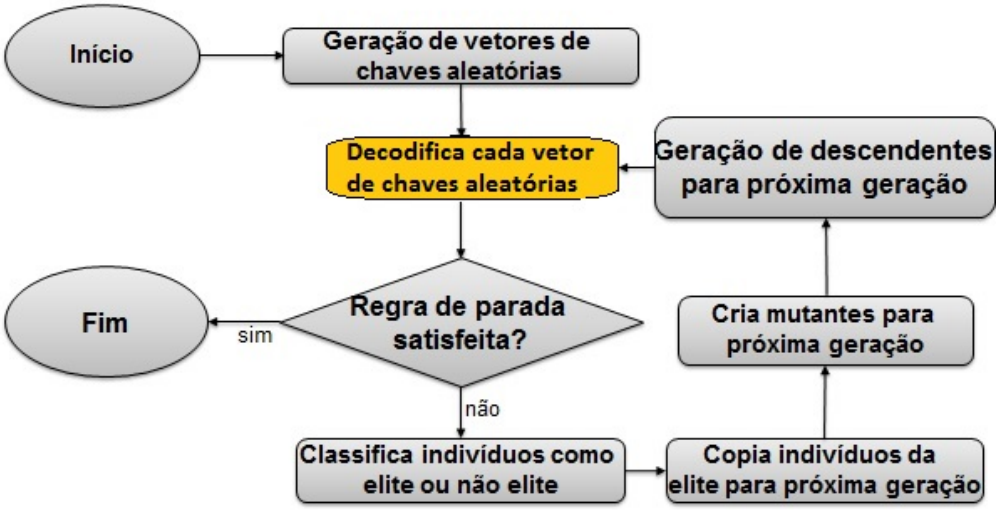


Figura V.4: Fluxograma BRKGA.

Fonte: adaptado de [54].

Resende e Toso [54] destacam o bloco que faz decodificação e observam que os cromossomos podem ser decodificados em paralelos, sendo um independente do outro.

Um fator comumente encontrado em muitas implementações da estrutura BRKGA é a adoção de múltiplas populações que evoluem de forma independente [27]. Durante o processo evolutivo, alguns indivíduos do conjunto elite são periodicamente selecionados de cada população e somados

aos outros substituindo os piores indivíduos. Esse recurso adicional é utilizado principalmente para acelerar a convergência do algoritmo.

No que diz respeito à implementação de algoritmos genéticos, de um modo geral, existe uma vasta gama de interface de aplicação de programas (API), projetados para reutilizar o código e facilitar a vida do usuário [35]. Notamos que elas são normalmente complexas por causa de a codificação ser selecionada pelos usuários junto com os operadores evolução e mutação, bem como o processo de decodificação. Por outro lado, como já mencionado, para utilizar a plataforma BRKGA de Resende e Toso apenas é necessário a implementação de um decodificador, todo o resto é independente do problema.

V.3. Parâmetros do BRKGA

A utilização do BRKGA necessita que alguns requisitos básicos sejam atendidos. Para podermos especificar um BRKGA precisamos saber como é representada a solução do problema a ser resolvido, como é implementada a decodificação e quais as características necessárias à resolução do problema que definirão os parâmetros do algoritmo.

Na Tabela V.1 são listados os parâmetros necessários para especificar o algoritmo genético com chaves aleatórias viciadas. Maiores detalhes sobre tais parâmetros podem ser encontrados em [25]. No capítulo seguinte apresentamos o Módulo Decodificador utilizado nesta dissertação.

Tabela V.1: Parâmetros necessários para BRKGA

Parâmetro	Descrição
n	número de alelos por cromossomo.
p	número de cromossomos da população.
p_e	tamanho do conjunto de elite da população.
p_m	número de mutantes a serem introduzidos na população a cada geração.
ρ	probabilidade de que um alelo seja herdado do pai elite.

Fonte: [54].

Capítulo VI - Proposta de solução do NLKP utilizando o BRKGA: módulo decodificador

Neste capítulo, apresentaremos a proposta de solução dos problemas $NLKP_1$ e $NLKP_2$ utilizando o *framework* BRKGA. Apresentaremos a proposta de um decodificador que converte uma solução do BRKGA em uma solução do $NLKP$ e calcula o *fitness* associado. Os detalhes da implementação desse procedimento são descritos neste capítulo. Descreveremos ainda o procedimento de busca local desenvolvido e aplicado no melhor indivíduo elite a cada iteração do BRKGA.

VI.1. Decodificador

O módulo decodificador implementado nesta dissertação tem a função de converter um vetor de chaves aleatórias, não viável, em uma solução viável do problema da mochila não linear ($NLKP$) e aplicar a função objetivo do referido problema para retornar o *fitness* associado à solução BRKGA. O decodificador aqui implementado apresenta uma sutil modificação em relação aos decodificadores apresentados na literatura, pois além de receber o vetor de chaves aleatórias, recebe um parâmetro que indica se a busca local irá ou não ser aplicada à solução do problema $NLKP$. Observe que esse detalhe de implementação permite que diversos tipos de busca local possam ser aplicados a uma solução BRKGA.

A Tabela VI.1 apresenta os elementos que constituem o pseudocódigo dos decodificadores aqui descritos para resolver os problemas $NLKP_1$ e $NLKP_2$.

Tabela VI.1: Descrição dos elementos do pseudocódigo dos decodificadores.

Elementos	Descrição
cromossomo $[i]$	componente i do vetor de solução BRKGA.
ls	parâmetro que indica se a busca local (*local search)* será aplicada.
fitness	$F(x)$, valor da função objetivo (ou aptidão do cromossomo).
n	tamanho do cromossomo.
x_i	componente i do vetor de solução $NLKP$.
u_i	limite superior da variável x_i.
$f(x_i)$	componente i da função objetivo do problema $NLKP$.
a_i, b_i, c_i, d_i	parâmetros de entrada da componente i da função objetivo do problema $NLKP$.
$g(x_i)$	componente i da restrição dos problemas $NLKP$.
w_i	parâmetros de entrada da componente i da restrição associada ao problema $NLKP_1$.
p_i, q_i	parâmetros de entrada da componente i da restrição associada ao problema $NLKP_2$.
ranking$[i]$	componente i do vetor que modela a relação $f(x_i)/g(x_i)$.
C	capacidade da mochila.
cap	capacidade incremental da mochila.

VI.1.1. Decodificador $NLKP_1$

Na Figura VI.1 a seguir, apresentamos o pseudocódigo associado ao módulo decodificador da classe $NLKP_1$ que se aplica tanto a variáveis inteiras quanto a variáveis contínuas através de pequena modificação nas linhas 4 e 10 do algoritmo. Esta modificação refere-se somente à retirada do operador piso ($\lfloor\ \rfloor$) e do valor 1 que não é mais acrescido aos limites superiores (u_i e u_k). O trecho de código entre as linhas 3 e 7 inicializam o vetor *ranking* que permitirá que as componentes com a maior relação $f(x_i)/g(x_i)$ sejam preservadas no processo de construção da viabilidade da solução $NLKP_1$. Na linha 8, o vetor *ranking* é ordenado de modo decrescente. A variável x e os parâmetros u, a, b, c, d e w são ordenados também de acordo com a ordenação dada ao vetor *ranking*. Nas linhas de 9 a 17, o cálculo do *fitness* de uma solução é realizado. Não há garantias de que uma solução BRKGA seja uma solução viável do $NLKP_1$. Sendo assim, desenvolvemos uma estratégia que torna a solução BRKGA viável, assegurando que as componentes com a maior relação $f(x)/g(x)$ sejam alocadas na mochila não linear até o limite da capacidade da mochila. As demais componentes não alocadas na mochila recebem valor 0. Como o BRKGA implementa um algoritmo de minimização retornamos à expressão $1/$*fitness*.

Entrada: cromossomo, n, ls, a, b, c, d, w
1 *fitness*= 0;
2 cap= 0;
3 **para** $i = 1$ *até* n **faça**
4 $x_i = \lfloor (u_i + 1) \times$ cromossomo$[i] \rfloor$;[a]
5 $f(x_i) = c_i/(1 + b_i e^{-a_i(x_i+d_i)})$;
6 $g(x_i) = w_i x_i$;
7 ranking$[i] = f(x_i)/g(x_i)$;
8 ordenar(*ranking*, x, a, b, c, d, w) // Ordene o vetor *ranking* em ordem decrescente;
9 **para** $k = 1$ *até* n **faça**
10 $x_k = \lfloor (u_k + 1) \times$ cromossomo$[k] \rfloor$;
11 $f(x_k) = c_k/(1 + b_k e^{-a_k(x_k+d_k)})$;
12 $g(x_k) = w_k x_k$;
13 **se** $(g(x_k) \leq C - cap)$ **então**
14 *fitness*+=$f(x_k)$;
15 cap+=$g(x_k)$;
16 **senão**
17 cromossomo$[k]$=0 //atualização do componente k do cromossomo;
18 **se** $(ls > 0)$ **então**
19 aplique a busca local tipo ls ao cromossomo;
Saída: $1/$*fitness*.

[a] ao usarmos variáveis reais não utizamos a função piso, $x_i = u_i \times$ cromossomo$[i]$.

Figura VI.1: Algoritmo do Decodificador $NLKP_1$.

Exemplo do Funcionamento do Decodificador $NLKP_1$ para Variáveis Inteiras

A Tabela VI.2 mostrada na próxima página, apresenta um exemplo do funcionamento do decodificador implementado para o problema $NLKP_1$ com cinco variáveis inteiras cujos índices estão representados na primeira linha da tabela. As linhas 2 e 3 desta tabela representam: o vetor de chaves aleatórias BRKGA (cromossomo) e o vetor solução do problema $NLKP_1$ após a conversão da solução do BRKGA, respectivamente. As linhas de 4 a 9 representam os parâmetros de entrada do decodificador (u_i, a_i, b_i, c_i, d_i e w_i). A linha 13 da tabela representa os novos índices das variáveis após a ordenação dada pelo vetor *ranking* e indicada por k.

Podemos facilmente verificar que a solução apresentada no início da Tabela VI.2 não é viável, uma vez que cap excede a capacidade da mochila representada pelo parâmetro C, admitindo que a condição da linha 13 não fosse aplicada. Na parte final da Tabela VI.2 apresentamos a aplicação do decodificador sobre a solução viável $NLKP_1$ com variáveis inteiras, observamos que os valores do *fitness* e cap são calculados e totalizados seguindo a ordenação determinada pelo vetor *ranking*, $(1, 5, 2, 4, 3)$, de maneira a garantir que as componentes com a maior relação $f(x)/g(x)$ sejam alocadas na mochila não linear até o limite da capacidade da mochila C. A componente x_3 não é alocada na mochila devido ao fato de que a sua inclusão viola a capacidade da mochila C.

Na página seguinte a Figura VI.2 ilustra a representação gráfica da solução do problema $NLKP_1$ com variáveis inteiras, analisado neste exemplo. Nesta classe de problema da mochila não linear, $NLKP_1$, o que difere uma instância inteira de uma instância contínua tratada na próxima seção são os valores da capacidade da mochila C e o vetor de pesos w. Os demais parâmetros permanecem com os mesmos valores.

Tabela VI.2: Exemplo da Aplicação do Decodificador $NLKP_1$

Solução Não Viável Inteira (C=12900)					
i	1	2	3	4	5
cromossomo [i]	0.743951	0.926444	0.751978	0.694158	0.962574
x_i	75	93	76	70	97
u_i	100	100	100	100	100
a_i	0.152216	0.118124	0.101030	0.196180	0.158531
b_i	44.742557	34.944390	93.324757	62.101010	71.753105
c_i	72.273751	77.468892	52.983122	56.186971	96.601371
d_i	-21.229801	-56.719870	-56.428945	-47.649553	-59.898149
w_i	4	44	63	55	48
$g(x_i)$	300	4092	4788	3850	4656
$f(x_i)$	71.383008	52.306942	3.806099	31.669228	80.489393
$f(x_i)/g(x_i)$	0.237943	0.012783	0.000795	0.008226	0.017287
k	1	5	2	4	3
fitness	71.383008	151.872401	204.179342	235.848571	**239.654670**
cap	300	4956	9048	12898	**17686**

Solução Viável Inteira (C=12900)					
cromossomo [i]	0.743951	0.926444	0.000000	0.694158	0.962574
x_i	75	93	0	70	97
$g(x_i)$	300	4092	0	3850	4656
$f(x_i)$	71.383008	52.306942	0.001898	31.669228	80.489393
fitness	71.383008	151.872401	204.179342	235.848571	**235.850468**
cap	300	4956	9048	12898	**12898**

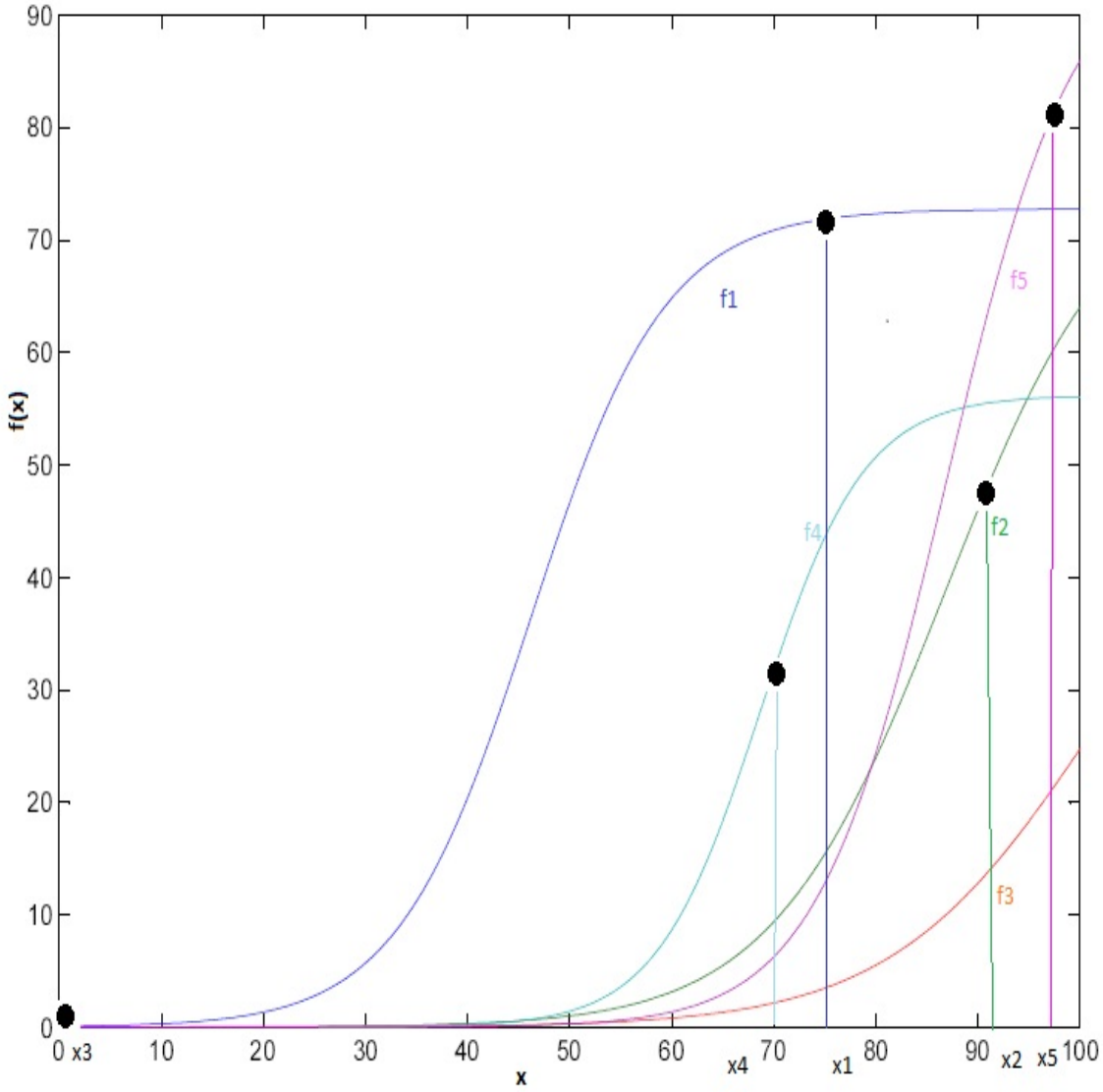


Figura VI.2: Representação gráfica da solução do problema $NLKP_1$ com variáveis inteiras.

Exemplo do Funcionamento do Decodificador $NLKP_1$ para Variáveis Contínuas

Apresentamos na Tabela VI.3, na página seguinte, um exemplo do funcionamento do decodificador implementado para o problema $NLKP_1$ com cinco variáveis contínuas cujos índices estão representados na primeira linha. Assim como na tabela anterior, as linhas 2 e 3 desta tabela representam: o vetor de solução BRKGA (cromossomo) e o vetor solução do problema $NLKP_1$, respectivamente. As linhas 4 a 9 representam os parâmetros de entrada do decodificador. A linha k representa os novos índices das variáveis após a ordenação dada pelo vetor *ranking* e identificadas por k.

Observamos facilmente que a solução apresentada no início da Tabela VI.3 não é viável, uma vez que cap excede a capacidade da mochila representada pelo parâmetro C, considerando que a condição da linha 13 não fosse aplicada. Na parte final da Tabela VI.3 apresentamos a aplicação do decodificador sobre a solução viável $NLKP_1$ com variáveis contínuas, assim observamos que os

valores do *fitness* e cap são calculados e totalizados seguindo a ordenação determinada pelo vetor *ranking*, de maneira análoga ao exemplo anterior, resultando na sequência $(1, 5, 2, 4, 3)$. Dessa forma, garantimos que, as componentes com a maior relação $f(x)/g(x)$ sejam alocadas na mochila não linear até o limite da capacidade da mochila C. A componente x_3 não é alocada na mochila porque a sua inclusão viola a capacidade da mochila representada no parâmetro C.

Tabela VI.3: Exemplo da Aplicação do Decodificador $NLKP_1$

Solução Não Viável Contínua (C=13009.737734)					
i	1	2	3	4	5
cromossomo [i]	0.734814	0.950893	0.763456	0.680028	0.985109
x_i	73.481400	95.089300	76.345600	68.002800	98.510900
u_i	100	100	100	100	100
a_i	0.152216	0.118124	0.101030	0.196180	0.158531
b_i	44.742557	34.944390	93.324757	62.101010	71.753105
c_i	72.273751	77.468892	52.983122	56.186971	96.601371
d_i	-21.229801	-56.719870	-56.428945	-47.649553	-59.898149
w_i	4.031705	44.109323	63.951720	55.586024	48.098622
$g(x_i)$	296.255328	4194.324648	4882.432434	3780.005273	4738.238542
$f(x_i)$	71.154952	56.306627	3.931305	26.187993	83.454149
$f(x_i)/g(x_i)$	0.240181	0.013424	0.000805	0.006928	0.017613
k	1	5	2	4	3
fitness	71.154952	154.609101	210.915729	237.103722	**241.035027**
cap	296.255328	5034.493870	9228.818517	13008.823790	**17891.256225**

Solução Viável Contínua (C=13009.737734)					
cromossomo [i]	0.734814	0.950893	0.000000	0.680028	0.985109
x_i	73.481400	95.089300	0.000000	68.002800	98.510900
$g(x_i)$	296.255328	4194.324648	0.000000	3780.005273	4738.238542
$f(x_i)$	71.154952	56.306627	0.001898	26.187993	83.454149
fitness	71.154952	154.609101	210.915729	237.103722	**237.105620**
cap	296.255328	5034.493870	9228.818517	13008.823790	**13008.823790**

VI.1.2. Decodificador $NLKP_2$

Na Figura VI.3, apresentamos o pseudocódigo associado ao módulo decodificador da classe $NLKP_2$ que se aplica tanto a variáveis inteiras quanto a variáveis contínuas através de pequena modificação nas linhas 4 e 10 do algoritmo. Essa modificação é efetuada por intermédio da retirada do operador piso ($\lfloor\ \rfloor$) e do valor 1 que não é mais acrescido aos limites superiores (u_i e u_k). Nessa classe de problema da mochila não linear os parâmetros p e q são utilizados para o cálculo da restrição $g(x)$. Nas linhas de 9 a 17, o cálculo do *fitness* de uma solução é realizado. Como o BRKGA implementa um algoritmo de minimização retornamos à expressão $1/$*fitness*.

Entrada: cromossomo, n, ls, a, b, c, d, p, q

1 *fitness*= 0;
2 cap= 0;
3 **para** $i = 1$ *até* n **faça**
4 $x_i = \lfloor (u_i + 1) \times$ cromossomo$[i] \rfloor$;[a]
5 $f(x_i) = c_i/(1 + b_i e^{-a_i(x_i+d_i)})$;
6 $g(x_i) = \sqrt{p_i x_i + q_i} - \sqrt{q_i}$;
7 ranking$[i] = f(x_i)/g(x_i)$;
8 ordenar(*ranking*, x, a, b, c, d, w) // Ordene o vetor ranking em ordem decrescente;
9 **para** $k = 1$ *até* n **faça**
10 $x_k = \lfloor (u_k + 1) \times$ cromossomo$[k] \rfloor$;
11 $f(x_k) = c_k/(1 + b_k e^{-a_k(x_k+d_k)})$;
12 $g(x_k) = \sqrt{p_k x_k + q_k} - \sqrt{q_k}$;
13 **se** $(g(x_k) \leq C - cap)$ **então**
14 *fitness*+=$f(x_k)$;
15 cap+=$g(x_k)$;
16 **senão**
17 cromossomo$[k]$=0 //atualização do componente k do cromossomo;
18 **se** $(ls > 0)$ **então**
19 aplique a busca local tipo ls no cromossomo;

Saída: $1/$*fitness*.

[a] ao usarmos variáveis reais não utilizamos a função piso, $x_i = u_i \times$ cromossomo$[i]$.

Figura VI.3: Algoritmo do Decodificador $NLKP_2$.

Exemplo do Funcionamento do Decodificador $NLKP_2$ para Variáveis Inteiras

A Tabela VI.4, mostrada na próxima página, apresenta um exemplo do funcionamento do decodificador implementado para o problema $NLKP_2$, com cinco variáveis inteiras, cujos índices estão representados na primeira linha da tabela. As linhas 2 e 3 desta tabela representam: o vetor de solução vindo do BRKGA (cromossomo) e o vetor solução do problema $NLKP_2$, respectivamente. As linhas de 4 a 10 representam os valores dos parâmetros de entrada do decodificador (u_i, a_i, b_i, c_i, d_i, p_i e q_i) em cada um dos componentes. A linha 14 da tabela representa os novos índices das variáveis após a ordenação dada pelo vetor *ranking* e indicada por k.

Podemos facilmente verificar que a solução apresentada na parte inicial da Tabela VI.4, não é viável, uma vez que cap excede a capacidade da mochila representada pelo parâmetro C, admitindo que a condição da linha 13 do pseudocódigo mostrado na Figura VI.3 não fosse aplicada. Na parte final da Tabela VI.4 apresentamos a aplicação do decodificador sobre a solução viável $NLKP_2$ com variáveis inteiras, observamos que os valores do *fitness* e *cap* são calculados e totalizados seguindo a ordenação determinada pelo vetor *ranking*, $(5, 4, 1, 2, 3)$, de maneira a garantir que as componentes com a maior relação $f(x)/g(x)$ sejam alocadas na mochila não linear até o limite da capacidade da mochila C. A componente x_3 não é alocada na mochila. O que difere uma instância inteira de uma

instância contínua, tratada no próximo exemplo, é apenas a capacidade da mochila C, que nessa última classe é real.

Tabela VI.4: Exemplo da Aplicação do Decodificador $NLKP_2$

Solução Não Viável Inteira (C=65)					
i	1	2	3	4	5
cromossomo [i]	0.678477	0.719317	0.786745	0.538154	0.973519
x_i	68	72	79	54	98
u_i	100	100	100	100	100
a_i	0.152216	0.103062	0.14328	0.152983	0.162101
b_i	44.742557	18.123835	43.544771	43.571055	56.186971
c_i	72.273751	34.944390	1.030165	63.587596	52.350447
d_i	-21.229801	-22.531108	-6.675243	-3.819920	-44.862602
p_i	12	10	20	2	1
q_i	2	18	12	5	19
$g(x_i)$	27.186486	22.923515	36.435773	8.394078	6.457755
$f(x_i)$	69.747749	31.462230	1.028750	62.328847	51.821640
$f(x_i)/g(x_i)$	2.565530	1.372487	0.028235	7.425336	8.024715
k	5	4	1	2	3
fitness	69.747749	121.569390	153.031620	215.360467	**216.389217**
cap	27.186486	33.644241	56.567755	64.961833	**101.397606**

Solução Viável Inteira (C=65)					
cromossomo [i]	0.678477	0.719317	0.000000	0.538154	0.973519
x_i	68	72	0	54	98
$g(x_i)$	27.186486	22.923515	0.000000	8.394078	6.457755
$f(x_i)$	69.747749	31.462230	1.028750	62.328847	51.821640
fitness	69.747749	121.569390	153.031620	215.360467	**216.389217**
cap	27.186486	33.644241	56.567755	64.961833	**64.961833**

Exemplo do Funcionamento do Decodificador $NLKP_2$ para Variáveis Contínuas

Apresentamos, na Tabela VI.5, mostrada na página a seguir, um exemplo do funcionamento do decodificador implementado para o problema $NLKP_2$, com cinco variáveis contínuas, cujos índices estão representados na primeira linha. Assim como nas tabelas anteriores, as linhas 2 e 3 desta tabela representam: o vetor de solução BRKGA (cromossomo) e o vetor solução do problema $NLKP_2$, respectivamente. As linhas 4 a 10 representam os parâmetros de entrada do decodificador. A linha k representa os novos índices das variáveis após a ordenação dada pelo vetor *ranking*.

Observamos facilmente que a solução apresentada no início da Tabela VI.5 não é viável, uma vez que a variável cap excede a capacidade da mochila representada pelo parâmetro C, considerando que a condição da linha 13 do Algoritmo do Decodificador $NLKP_2$, não fosse aplicada. Na parte final da Tabela VI.5 apresentamos a aplicação do decodificador sobre a solução viável $NLKP_2$ com variáveis contínuas, observamos que os valores do *fitness* e *cap* são calculados e totalizados

seguindo a ordenação determinada pelo vetor *ranking*, de maneira semelhante ao exemplo anterior, resultando na sequência $(5, 4, 1, 2, 3)$. Dessa forma, garantimos que, as componentes com a maior relação $f(x)/g(x)$ sejam alocadas na mochila não linear até o limite da capacidade da mochila C. A componente x_3 não é alocada na mochila.

Tabela VI.5: Exemplo da Aplicação do Decodificador $NLKP_2$

Solução Não Viável Contínuas (C=65.200755)					
i	1	2	3	4	5
cromossomo [i]	0.668911	0.732815	0.786745	0.567014	0.975590
x_i	66.8911	73.2815	78.6745	56.7014	97.5590
u_i	100	100	100	100	100
a_i	0.152216	0.103062	0.14328	0.152983	0.162101
b_i	44.742557	18.123835	43.544771	43.571055	56.186971
c_i	72.273751	34.944390	1.030165	63.587596	52.350447
d_i	-21.229801	-22.531108	-6.675243	-3.819920	-44.862602
p_i	12	10	20	2	1
q_i	2	18	12	5	19
$g(x_i)$	26.952901	23.158363	36.354110	8.645237	6.437350
$f(x_i)$	69.302378	31.854967	1.028682	62.749336	51.782879
$f(x_i)/g(x_i)$	2.571240	1.375528	0.028296	7.258255	8.044129
k	5	4	1	2	3
fitness	51.782879	114.532214	183.834593	215.689560	**216.718242**
cap	6.437350	15.082587	42.035489	65.193852	**101.547962**

Solução Viável Contínuas (C=65.200755)					
cromossomo [i]	0.668911	0.732815	0.000000	0.567014	0.975590
x_i	66.89110	73.28150	0.00000	56.70140	97.55900
$g(x_i)$	26.952901	23.158363	0.000000	8.645237	6.437350
$f(x_i)$	69.302378	31.854967	0.009011	62.749336	51.782879
fitness	51.782879	114.532214	183.834593	215.689560	**215.698571**
cap	6.437350	15.082587	42.035489	65.193852	**65.193852**

VI.2. Busca Local

Rotinas de buscas locais são estratégias de busca por melhores soluções através da exploração exaustiva da vizinhança de uma dada solução. Nesse contexto, uma das estratégias mais conhecidas é a $2-$opt, que foi proposta inicialmente por Lin, em 1965, na resolução do Problema do Caixeiro Viajante [39]. A partir de uma dada solução, o procedimento $2-$opt consiste em realizar todas as possíveis trocas entre duas componentes deste vetor solução.

Nesta pesquisa, a busca local $2-$opt é aplicada ao melhor indivíduo da população em cada geração do BRKGA. O pseudocódigo desse procedimento é representado na Figura VI.4. Cabe observar que neste processo de intensificação por busca de melhores soluções não reiniciamos o processo após encontrar uma melhoria. Caso uma melhoria seja encontrada continuamos o processo de busca

para as demais componentes, conforme podemos observar nas Tabelas VI.6, VI.7 e VI.8 (mostradas nas páginas a seguir).

A seguir, apresentamos o algoritmo de busca local na Figura VI.4. O procedimento descrito no algoritmo é válido para o $NLKP_1$ e $NLKP_2$ com pequenas modificações.

Entrada: cromossomo, n, a, b, c, d, w, p, q
1 **para** $i = 1$ *até* $n - 1$ **faça**
2 **para** $j = i + 1$ *até* n **faça**
3 $x_i = \lfloor (u_i + 1) \times$ cromossomo$[i] \rfloor$;[a]
4 $x_j = \lfloor (u_j + 1) \times$ cromossomo$[j] \rfloor$;[b]
5 $f(x_i) = c_i/(1 + b_i e^{-a_i(x_i+d_i)})$;
6 $f(x_j) = c_j/(1 + b_j e^{-a_j(x_j+d_j)})$;
7 $g(x_i) = w_i x_i$;[c]
8 $g(x_j) = w_j x_j$;[d]
9 $f^j(x_i) = c_j/(1 + b_j e^{-a_j(x_i+d_j)})$;
10 $f^i(x_j) = c_i/(1 + b_i e^{-a_i(x_j+d_i)})$;
11 $g^j(x_i) = w_j x_i$;[e]
12 $g^i(x_j) = w_i x_j$;[f]
13 **se** $((g^j(x_i) + g^i(x_j)) - (g(x_i) + g(x_j)) \leq (C - cap))$ **então**
14 **se** $((f^j(x_i) + f^i(x_j)) - (f(x_i) + f(x_j)) > 0$ **então**
15 troca x_i com x_j;
16 *fitness*+=$(f^j(x_i) + f^i(x_j)) - ((f(x_i) + f(x_j))$;
17 cap+=$(g^j(x_i) + g^i(x_j)) - (g(x_i) + g(x_j))$;
Saída: cromossomo

[a] ao usarmos variáveis reais não utizamos a função piso, $x_i = u_i \times$ cromossomo$[i]$.
[b] ao usarmos variáveis reais não utizamos a função piso, $x_j = u_j \times$ cromossomo$[j]$.
[c] para o problema $NLKP_2$ use $g(x_i) = \sqrt{p_i x_i + q_i} - \sqrt{q_i}$
[d] para o problema $NLKP_2$ use $g(x_j) = \sqrt{p_j x_j + q_j} - \sqrt{q_j}$
[e] para o problema $NLKP_2$ use $g^j(x_i) = \sqrt{p_j x_i + q_j} - \sqrt{q_j}$
[f] para o problema $NLKP_2$ use $g^i(x_j) = \sqrt{p_i x_j + q_i} - \sqrt{q_i}$

Figura VI.4: Busca Local.

Observe que, nesta busca, verificamos para todo par (x_i, x_j) se uma determinada troca viola ou não a restrição de capacidade da mochila, na linha 13. Em seguida, avaliamos se a troca acarreta um ganho na função objetivo, linha 14. Se essas duas condições forem satisfeitas, trocamos as componentes x_i e x_j, linha 15, atualizamos o *fitness* associado a esta solução e a capacidade residual da mochila, linhas 16 e 17. Veja que este tipo de busca local realiza um total de $n(n-1)/2$ comparações no processo de intensificação de uma instância de tamanho n.

No exemplo a seguir, realizamos 10 comparações com a finalidade de melhorar a solução inicial $NLKP_1$, com variáveis inteiras ($n = 5$). Nas Tabelas VI.6 e VI.7 apresentamos a busca local até a primeira melhoria ou troca que permite aumentar o valor do *fitness*. A Tabela VI.8 apresenta as demais iterações da busca sem que uma melhoria seja encontrada. As componentes avaliadas a cada passo da busca local, nas três tabelas, estão destacadas em negrito.

Tabela VI.6: Exemplo da Aplicação da Busca Local $2-$opt

Solução Inicial Viável Inteira (C=12900)						
x_i	44	0	53	70	97	**soma**
$g(x_i)$	176	0	3339	3850	4656	**12021**
$f(x_i)$	30.141044	0.002729	0.398484	31.669228	80.489393	**142.700878**
Solução Não Viável						
$(\mathbf{x_1}, \mathbf{x_2})$	**0**	**44**	53	70	97	**soma**
$g(x_i)$	0	1936	3339	3850	4656	**13781**
$f(x_i)$	0.063745	0.490293	0.398484	31.669228	80.489393	**113.111143**
Solução Viável (Trocou)						
$(\mathbf{x_1}, \mathbf{x_3})$	**53**	0	**44**	70	97	**soma**
$g(x_i)$	212	0	2772	3850	4656	**11490**
$f(x_i)$	53.329757	0.002729	0.161241	31.669228	80.489393	**165.652347**

Tabela VI.7: Exemplo da Aplicação da Busca Local $2-$opt - Continuação

Solução Viável Inteira (C=12900)						
x_i	53	0	44	70	97	**soma**
$g(x_i)$	212	0	2772	3850	4656	**11490**
$f(x_i)$	53.329757	0.002729	0.161241	31.669228	80.489393	**165.652347**
Solução Viável (Não Troca)						
$(\mathbf{x_1}, \mathbf{x_4})$	**70**	0	44	**53**	97	**soma**
$g(x_i)$	280	0	2772	2915	4656	**10623**
$f(x_i)$	70.393470	0.002729	0.161241	2.470943	80.489393	**153.517776**
Solução Viável (Não Troca)						
$(\mathbf{x_1}, \mathbf{x_5})$	**97**	0	44	70	**53**	**soma**
$g(x_i)$	388	0	2772	3850	2544	**9554**
$f(x_i)$	72.242084	0.002729	0.161241	31.669228	0.448940	**104.524222**
Solução Viável (Trocou)						
$(\mathbf{x_2}, \mathbf{x_3})$	53	**44**	**0**	70	97	**soma**
$g(x_i)$	212	1936	0	3850	4656	**10654**
$f(x_i)$	53.329757	0.490293	0.001898	31.669228	80.489393	**165.980568**

Tabela VI.8: Exemplo da Aplicação da Busca Local $2-$opt - Continuação

Solução Viável Inteira (C=12900)						
x_i	53	44	0	70	97	**soma**
$g(x_i)$	212	1936	0	3850	4656	**10654**
$f(x_i)$	53.329757	0.490293	0.001898	31.669228	80.489393	**165.980568**
Solução Viável (Não Troca)						
$(\mathbf{x_2}, \mathbf{x_4})$	53	**70**	0	**44**	97	**soma**
$g(x_i)$	212	3080	0	2420	4656	**10368**
$f(x_i)$	53.329757	9.356763	0.001898	0.438724	80.489393	**143.616534**
Solução Viável (Não Troca)						
$(\mathbf{x_2}, \mathbf{x_5})$	53	**97**	0	70	**44**	**soma**
$g(x_i)$	212	4268	0	3850	2112	**10442**
$f(x_i)$	53.329757	59.595846	0.001898	31.669228	0.108164	**144.704893**
Solução Viável (Não Troca)						
$(\mathbf{x_3}, \mathbf{x_4})$	53	44	**70**	**0**	97	**soma**
$g(x_i)$	212	1936	4410	0	4656	**11214**
$f(x_i)$	53.329757	0.490293	2.146040	0.000079	80.489393	**136.455561**
Solução Viável (Não Troca)						
$(\mathbf{x_3}, \mathbf{x_5})$	53	44	**97**	70	**0**	**soma**
$g(x_i)$	212	1936	6111	3850	0	**12109**
$f(x_i)$	53.329757	0.490293	20.791124	31.669228	0.000101	**106.280503**
Solução Viável (Não Troca)						
$(\mathbf{x_4}, \mathbf{x_5})$	53	44	0	**97**	**70**	**soma**
$g(x_i)$	212	1936	0	5335	3360	**10843**
$f(x_i)$	53.329757	0.490293	0.001898	55.970000	6.246202	**116.038150**

A Tabela VI.9 apresenta a solução inicial e a solução resultante do processo de aplicação da busca local $2-$opt.

Tabela VI.9: Soluções Inicial e Resultante do Exemplo da Aplicação da Busca Local $2-$opt

Solução Inicial Viável Inteira (C=12900)						
x_i	44	0	53	70	97	**soma**
$g(x_i)$	176	0	3339	3850	4656	**12021**
$f(x_i)$	30.141044	0.002729	0.398484	31.669228	80.489393	**142.700878**
Solução Viável (Trocou)						
$(\mathbf{x_2}, \mathbf{x_3})$	53	44	0	70	97	**soma**
$g(x_i)$	212	1936	0	3850	4656	**10654**
$f(x_i)$	53.329757	0.490293	0.001898	31.669228	80.489393	**165.980568**

Capítulo VII - Resultados computacionais

Neste capítulo apresentaremos os resultados computacionais obtidos com a aplicação do *framework* BRKGA proposto para a solução do problema da mochila não linear separável.

O BRKGA foi implementado na linguagem de programação C++ e compilado com g++ (GNU COMPILE C++). Utilizamos o algoritmo de Mersene Twister mostrado em Matsumoto e Nishimura [44] para gerar os números aleatórios.Todas as execuções foram realizadas no Laboratório de Otimização, LabOtim da COPPE/UFRJ em uma máquina com 12 processadores, cada um desses processadores com a seguinte configuração: 48GB Ram, 3.07GHz processador Intel, SO: Ubuntu 12.04.5 LTS (GNU/Linux 3.8.0-29-generic x86_64). É importante ressaltar que utilizamos somente um núcleo para rodar cada um dos processos.

Em todas as execuções do BRKGA, os parâmetros utilizados foram:

- tamanho da população (p) igual a 100;
- tamanho da população elite (p_e) igual a 10% de p;
- tamanho da população de mutantes (p_m) igual a 10% de p;
- probabilidade de que descendentes herdem um alelo do pai elite (ρ_e) igual a 0.7;
- o critério de parada utilizado foi o tempo de execução do algoritmo: *time out* igual a 180 segundos.

Apresentamos os resultados computacionais da aplicação do BRKGA para as duas classes do problema da mochila não linear separável, $NLKP_1$ e $NLKP_2$, definidos no Capítulo III. Nos experimentos computacionais, inicialmente usamos o BRKGA sem o procedimento de busca local para o problema da mochila não linear separável com restrição linear, o $NLKP_1$, o qual denotamos $BsBL_1$ e, o BRKGA sem busca local para o problema da mochila não linear separável com restrição não linear, o $NLKP_2$, o qual denotamos $BsBL_2$. Em seguida, o BRKGA foi utilizado com um procedimento de busca local $2-$opt para os problemas $NLKP_1$ e $NLKP_2$, o qual denotamos respectivamente por $BcBL_1$ e $BcBL_2$. Em ambos os problemas, a busca local $2-$opt foi aplicada no melhor indivíduo da população a cada iteração do algoritmo genético.

Nesse experimento computacional utilizamos 640 instâncias disponibilizadas em [1]. Essas instâncias foram utilizadas, anteriormente em [2], com dimensões variando entre 10 e 2500, totalizando

8 grupos de instâncias, com 20 instâncias em cada grupo. Assim, obtivemos um total de 160 instâncias divididas em 8 grupos para cada classe do problema ($NLKP_1$ com variáveis inteiras, $NLKP_1$ com variáveis reais, $NLKP_2$ com variáveis inteiras e $NLKP_2$ com variáveis reais). Para cada instância, o BRKGA foi executado cinco vezes, com cinco sementes distintas para o gerador de números aleatórios. Os resultados apresentados correspondem a média das cinco melhores soluções encontradas. A Figura VII.1 ilustra a nomenclatura utilizada neste capítulo:

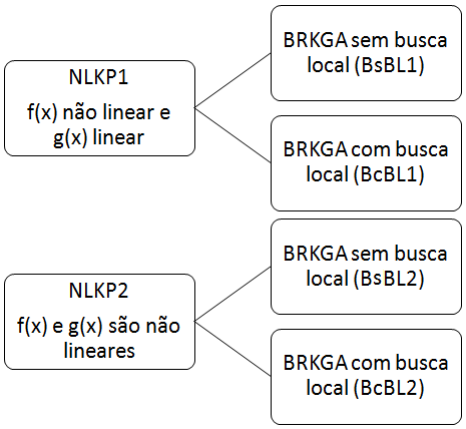


Figura VII.1: Simbologia para as propostas de solução dos problemas $NLKP$.

Inicialmente, apresentaremos os resultados computacionais para a classe $NLKP_1$ com variáveis inteiras e com variáveis reais. Sabemos que temos 20 instâncias com tamanho variando de 10 a 2500. Para cada instância rodamos o BRKGA ($BsBL_1$ e o $BcBL_1$) cinco vezes. A média das melhores soluções do problema $NLKP_1$ é armazenada e em seguida calculamos a média das médias dos resultados. Essa média das médias das melhores soluções é que está sendo comparada com os resultados obtidos na literatura. Em seguida, mostraremos os resultados computacionais para a classe $NLKP_2$, também para variáveis inteiras e reais.

Para efetuarmos a comparação entre os resultados obtidos com o algoritmo desenvolvido nesta dissertação e os resultados da literatura, organizamos os resultados computacionais referentes às médias através do desvio percentual relativo (DPR) [45], calculados através da equação VII.1:

$$Z(A_1, A_2) = 100 \times \frac{(A_1 - A_2)}{A_1}, \tag{VII.1}$$

onde A_1 e A_2 representam os valores obtidos pelos métodos utilizados e pelos quais os seus resultados computacionais serão comparados. Portanto, o DPR será utilizado como uma métrica para comparar a eficácia dos métodos heurísticos. Por se tratar de um problema de maximização, o DPR negativo significa que a solução obtida pelo algoritmo A_1 é pior do que a solução obtida por A_2.

VII.1. Classe $NLKP_1$ com Variáveis Inteiras

A Tabela VII.1 apresenta a média das melhores soluções encontradas por D'Ambrosio e Martello obtidas da literatura [2] e os resultados dos algoritmos $BsBL_1$ e $BcBL_1$ para variáveis inteiras. A primeira coluna da Tabela VII.1 indica o tamanho para cada grupo de instâncias (n). A segunda coluna apresenta o valor médio das soluções obtidas pelo algoritmo heurístico $Heur$ e apresentado no artigo [2]. Os autores rodaram a heurística para 32 configurações de combinações de seus parâmetros, sendo escolhida a melhor solução para realizar a média das soluções, com limite de tempo de CPU de 1 h por execução, sendo que o algoritmo nunca excedeu 60 s. A terceira coluna exibe o valor médio das soluções produzidas pelo algoritmo heurístico $Heur$ com uma busca local determinística (LS_D) [2]. Os autores também resolveram cada instância para 32 configurações de combinações de seus parâmetros, sendo escolhida a melhor solução para realizar a média das soluções, com limite de tempo de CPU de 5 s por execução de busca local. A quarta coluna mostra o valor médio das soluções produzidas pelo algoritmo heurístico $Heur$ com uma busca local aleatória (LS_R) com as mesmas características do procedimento anterior. Na quinta coluna, apresentamos o valor das médias das médias das soluções para cada grupo de instâncias produzidas pelo BRKGA sem busca local, $BsBL_1$, com limite de tempo de CPU de 180 s por ciclo. Na sexta coluna, destacamos o valor das médias das médias das soluções para cada grupo de instâncias, produzidas pelo algoritmo $BcBL_1$, com limite de tempo de CPU de 180 s por ciclo. Os melhores resultados para cada instância estão em negrito na Tabela VII.1.

Tabela VII.1: Médias dos valores das melhores soluções para o $NLKP_1$ utilizando variáveis inteiras.

n	$Heur$[1]	LS_D [2]	LS_R [3]	$BsBL_1$ [4]	$BcBL_1$ [5]
10	280.69	286.08	285.03	288.96	**288.97**
20	648.93	653.65	653.78	656.64	**656.71**
50	1676.66	1690.52	**1690.95**	1683.29	1688.9
100	3360.28	3389.11	**3389.14**	3359.56	3384.14
200	6822.76	6906.50	6907.85	6817.05	**6916.88**
500	16915.56	**17185.24**	17184.23	16207.33	16979.51
1000	33617.79	34243.26	**34246.25**	28369.34	31383.08
2500	83404.46	**85228.34**	85164.41	49861.46	73035.71

O DPR dos resultados apresentadas na Tabela VII.1 estão consolidados na Tabela VII.2, em que as colunas podem ser descritas da seguinte forma: a primeira, corresponde ao tamanho para cada grupo de instâncias; a segunda, $Z(BsBL_1, Heur)$, equivale à comparação entre as médias

[1] Autoria dos dados dessa coluna são de Ambrosio e Martello, [2].
[2] Autoria dos dados dessa coluna são de Ambrosio e Martello, [2].
[3] Autoria dos dados dessa coluna são de Ambrosio e Martello, [2].
[4] Autoria dos dados dessa coluna é de Pereira.
[5] Autoria dos dados dessa coluna é de Pereira.

das médias dos resultados do $BsBL_1$ com as médias das soluções do $Heur$, de acordo com a equação VII.1; a terceira coluna, $Z(BcBL_1, LS_D)$, apresenta similarmente à comparação entre as médias das médias dos resultados do $BcBL_1$ com as médias das soluções do LS_D; a quarta coluna, $Z(BcBL_1, LS_R)$, denota a comparação entre as médias das médias dos resultados do $BcBL_1$ com as médias das soluções do LS_R e, finalmente, a quinta coluna, $Z(BcBL_1, BsBL_1)$, corresponde a comparação entre as médias das médias dos resultados do $BcBL_1$ com as médias das médias das soluções do $BsBL_1$.

Observe que a Tabela VII.2 visa comparar os DPRs dos resultados obtidos com as melhores soluções do BRKGA com as soluções fornecidas pelo trabalho de D'Ambrosio e Martello [2]. A última coluna compara ainda os resultados computacionais obtidos pelos nossos algoritmos, ou seja, a comparação entre o $BcBL$ e o $BsBL$. Os dados em negrito indicam as instâncias em que o BRKGA atingiu melhores resultados que os outros métodos da literatura.

Tabela VII.2: Comparações entre as médias das melhores soluções para o $NLKP_1$ utilizando variáveis inteiras.

n	$Z(BsBL_1, Heur)$	$Z(BcBL_1, LS_D)$	$Z(BcBL_1, LS_R)$	$Z(BcBL_1, BsBL_1)$
10	**2.86%**	**1.00%**	**1.30%**	0.00%
20	**1.17%**	**0.62%**	**0.60%**	**0.16**%
50	**0.39%**	-0.10%	-0.12%	**0.33%**
100	-0.02%	-0.15%	-0.15%	**0.73%**
200	-0.08%	**0.15%**	**0.13%**	**1.44%**
500	-4.37%	-1.21%	-1.21%	**4.55%**
1000	-18.5%	-9.11%	-9.12%	**9.6%**
2500	-67.27%	-16.69%	-16.61%	**31.73%**

Notamos que os resultados obtidos com a implementação BRKGA com e sem busca local são de boa qualidade para vários valores de n, no problema $NLKP_1$, com variáveis inteiras. Observamos que o $BcBL_1$ apresentou melhores resultados para três grupos de instâncias, a saber 10, 20 e 200. A Figura VII.2 ilustra através de um gráfico de barras a comparação entre os resultados computacionais obtidos para o problema $NLKP_1$ com variáveis inteiras e os resultados utilizados como referência. Observamos que para os valores de $n \leq 500$ as soluções apresentados pelo BRKGA com busca local têm módulo do desvio máximo de 1.21%.

Observe que para as instâncias de tamanho $n \geq 1000$, a diferença na qualidade da solução do BRKGA é inferior aos métodos da literatura e essa diferença aumenta com o tamanho da instância. Entretanto, ao compararmos o BRKGA com busca local e sem busca local, as soluções com busca local são significativamente melhores, como podemos verificar na Figura VII.2.

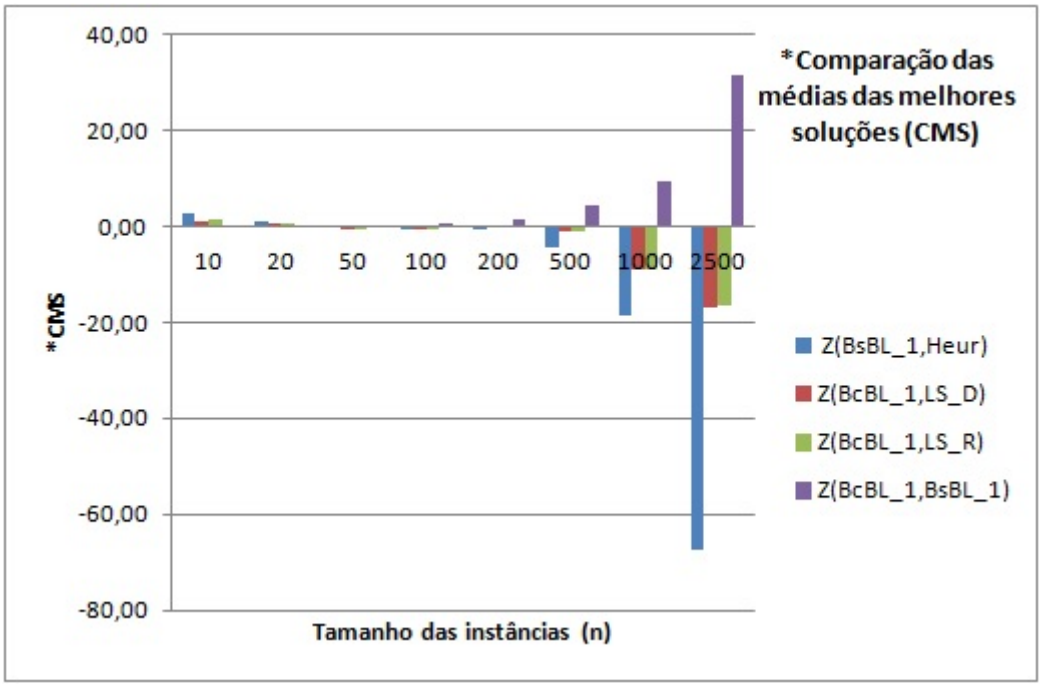


Figura VII.2: Gráfico comparativo entre as médias das melhores soluções para o problema $NLKP_1$ com variáveis inteiras.

VII.2. Classe $NLKP_1$ com Variáveis Reais

A Tabela VII.3 a seguir apresenta as médias dos valores das melhores soluções das heurísticas de referências e das propostas de soluções considerando 20 instâncias de mesma dimensão para o $NLKP_1$ com variáveis reais.

Tabela VII.3: Médias das melhores soluções para o $NLKP_1$ utilizando variáveis reais.

n	$Heur$ [1]	LS_D [2]	LS_R [3]	$BsBL_1$ [4]	$BcBL_1$ [5]
10	280.08	286.06	285.87	288.87	**288,92**
20	648.32	653.44	653.51	655,05	**657,21**
50	1676.62	1690.02	**1690.91**	1679.44	1686.37
100	3358.78	3388.42	**3388.84**	3349.62	3378.50
200	6824.79	6906.06	**6906.35**	6761.60	6866.12
500	16931.91	**17188.00**	17183.49	16143.78	16859.62
1000	33660.48	**34252.91**	34239.55	28048.67	31241.02
2500	83513.24	**85234.88**	85167.75	50567.45	74743.69

Na Tabela VII.4 efetuamos a comparação entre os resultados computacionais das heurísticas propostas e dos algoritmos, $Heur$, LS_D e LS_R, para o problema $NLKP_1$ com variáveis reais. Os resultados das comparações entre as médias das melhores soluções das heurísticas citadas foram

[1] Autoria dos dados dessa coluna são de Ambrosio e Martello, [2].
[2] Autoria dos dados dessa coluna são de Ambrosio e Martello, [2].
[3] Autoria dos dados dessa coluna são de Ambrosio e Martello, [2].
[4] Autoria dos dados dessa coluna é de Pereira.
[5] Autoria dos dados dessa coluna é de Pereira.

transformados em desvio percentual relativo (DPR).

Tabela VII.4: Comparações entre as médias das melhores soluções para $NLKP_1$ utilizando variáveis reais.

n	$Z(BsBL_1, Heur)$	$Z(BcBL_1, LS_D)$	$Z(BcBL_1, LS_R)$	$Z(BcBL_1, BsBL_1)$
10	**3.04%**	**0.99%**	**1.06%**	**0.02%**
20	**1.03%**	**0.57%**	**0.56%**	**0.33%**
50	**0.17%**	-0.22%	-0.27%	**0.41%**
100	-0.27%	-0.29%	-0.31%	**0.85%**
200	-0.93%	-0.58%	-0.59%	**1.52%**
500	-4.88%	-1.95%	-1.92%	**4.25**%
1000	-20.01%	-9.64%	-9.60%	**10.22%**
2500	-65.15%	-14.04%	-13.95%	**32.25%**

A Tabela VII.3 evidencia bons resultados para vários valores de n com a utilização do BRKGA, com e sem busca local, como forma de solução aproximada para o problema $NLKP_1$ com variáveis reais, e também compara o $BcBL_1$ com o $BsBL_1$. Analisando a Tabela VII.3 verificamos que o $BcBL_1$ apresentou melhores resultados para instâncias de tamanhos 10 e 20. A Figura VII.3, mostrada na próxima página, ilustra através de um gráfico de barras a comparação entre os resultados computacionais das heurísticas propostas para o problema $NLKP_1$ para variáveis reais e os resultados utilizados como referência.

Com os dados da Tabela VII.4, observamos que, para os valores de $n \leq 500$, as médias das melhores soluções apresentadas pelo BRKGA com busca local são melhores ou ligeiramente piores, em relação as médias das melhores soluções da heurística usada como referencial, com um desvio máximo de 1.95%.

No caso das pequenas instâncias como n igual a 10, por exemplo, o $BsBL_1$ tem um valor do DPR de aproximadamente 3% em relação $Heur$, o $BcBL_1$ tem um valor do DPR em torno de 1 % em relação ao LS_D e o ganho entre o $BcBL_1$ e o LS_R é também de aproximadamente 1%, comportamento análogo ao $NLKP_1$ para variáveis inteiras.

Para as instâncias maiores, $n \in \{500, 1000, 2500\}$, observamos que os resultados da literatura são melhores que os do BRKGA e essa diferença aumenta à medida que a dimensão do problema aumenta. O gráfico mostrado na VII.3 explicita claramente este fato.

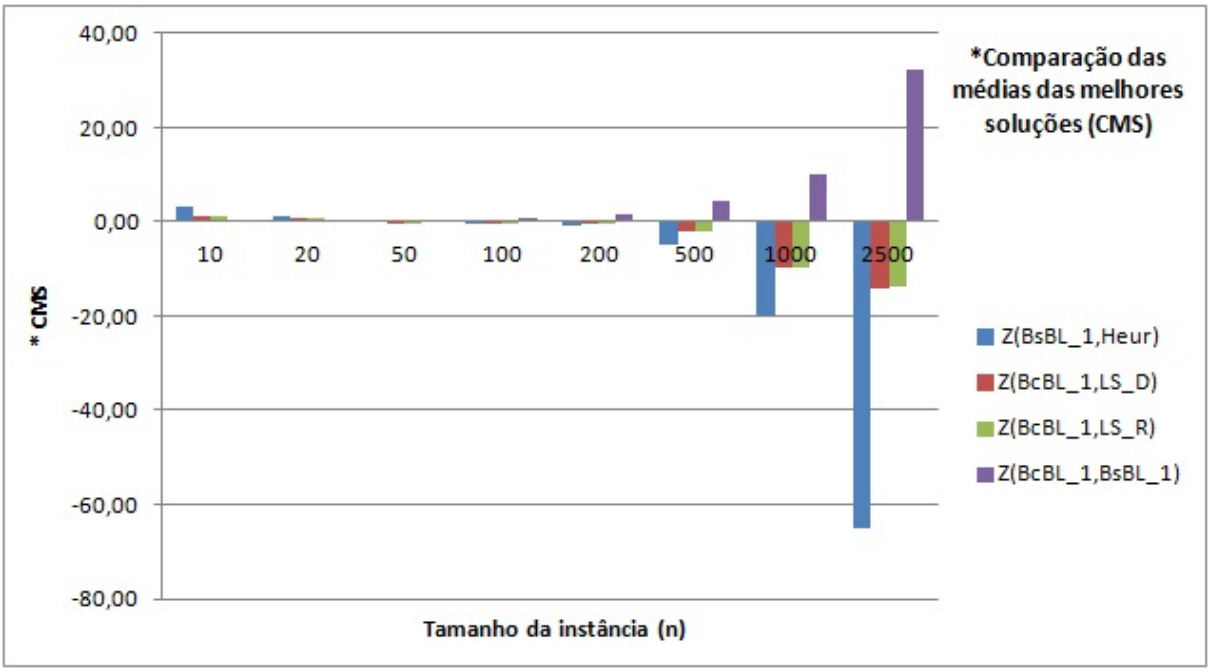


Figura VII.3: Gráfico comparativo entre as médias das melhores soluções para o problema $NLKP_1$ com variáveis reais.

VII.3. Classe $NLKP_2$ com Variáveis Inteiras

A Tabela VII.5 expõe as médias das melhores soluções encontradas a partir da literatura e as médias das médias das melhores soluções fornecidas pelos algoritmos propostos para o $NLKP_2$ para variáveis inteiras. Onde, na quinta coluna, $BsBL_2$, é o valor das médias das médias das melhores soluções para cada grupo de instâncias produzidas pelo algoritmo genético $BsBL_2$, com limite de tempo de CPU de 180 s por ciclo, sendo que o código roda cinco vezes para cinco sementes distintas do gerador de números aleatórios, totalizando, portanto, 900 s para cada execução do algoritmo. Na sexta coluna apresentamos o valor das médias das médias das soluções para cada grupo de instâncias produzidas pelo algoritmo genético $BcBL_2$, com comportamento análogo ao anterior.

A tabela VII.5 evidencia resultados promissores para vários tamanhos de instâncias com a utilização do BRKGA, com e sem busca local, como forma de solução aproximada para o problema $NLKP_2$ com variáveis inteiras. Na Tabela VII.5 verificamos que para grupos de instâncias de tamanhos: 10, 20 e 50, o algoritmo $BcBL_2$ obteve os melhores resultados em relação a outros métodos da literatura, utilizados como referencial. É importante ressaltar que, devido às condições de integralidade do domínio dessa classe, a heurística construtiva de D'Ambrosio e Martello tem maior dificuldade de construir boas soluções para alguns grupos de instâncias. Observamos que nesses grupos de instâncias, o BRKGA acaba tendo um comportamento mais eficaz.

Tabela VII.5: Médias das melhores soluções para o $NLKP_2$ utilizando variáveis inteiras.

n	$Heur$ [1]	LS_D [2]	LS_R [3]	$BsBL_2$ [4]	$BcBL_2$ [5]
10	297.31	300.56	301.38	308.98	**309.04**
20	595.19	600.58	600.29	605.15	**605.43**
50	1599.67	1608.67	1608.49	1603.53	**1610.12**
100	3246.83	3261.46	**3261.79**	3232.90	3256.83
200	6561.43	6584.83	**6585.72**	6467.24	6560.90
500	16378.68	16438.61	**16440.14**	15583.52	16223.84
1000	32921.17	33053.06	**33054.79**	27828.11	31172.58
2500	81689.36	82067.41	**82069.32**	49534.94	73301.53

Na Tabela VII.6 a primeira coluna corresponde ao tamanho da instância. A segunda coluna $Z(BsBL_2, Heur)$ mostra à comparação entre as médias das médias das melhores soluções do $BsBL_2$ com as médias das melhores soluções do $Heur$. Na terceira coluna temos $Z(BcBL_2, LS_D)$ a comparação entre as médias das médias das melhores soluções do $BcBL_2$ com as médias das melhores soluções do LS_D. Na quarta coluna $Z(BcBL_2, LS_R)$, apresentamos a comparação entre as médias das médias das melhores soluções do $BcBL_2$ com as médias das melhores soluções do LS_R. A quinta coluna $Z(BcBL_2, BsBL_2)$ equivale à comparação entre as médias das médias das melhores soluções do $BcBL_2$ com as médias das médias das melhores soluções do $BsBL_2$.

Por intermédio da Tabela VII.6, observamos que para as instâncias de $n \leq 500$, os resultados apresentados pelo $BcBL_2$ são de boa qualidade e estão bem próximos aos da literatura, situação que também observamos nas classes $NLKP_1$ com variáveis inteiras e reais. Nesse caso, para instâncias de até 500 variáveis, o módulo do desvio máximo entre as médias das melhores soluções do $BcBL_2$ em relação a resultados utilizados como referência é de 1.27%.

Tabela VII.6: Comparações entre as médias das melhores soluções para o $NLKP_2$ utilizando variáveis inteiras.

n	$Z(BsBL_2, Heur)$	$Z(BcBL_2, LS_D)$	$Z(BcBL_2, LS_R)$	$Z(BcBL_2, BsBL_2)$
10	**4.33%**	**2.74%**	**2.48%**	0.02%
20	**1.65%**	**0.80%**	**0.85%**	**0.05%**
50	**0.24%**	**0.09%**	**0.10%**	**0.41%**
100	-0.43%	-0.14%	-0.15%	**0.73%**
200	-1.46%	-0.36%	-0.38%	**1.43%**
500	-5.10%	-1.26%	-1.27%	**4.01%**
1000	-18.30%	-6.03%	-6.04%	**10.73%**
2500	-64.91%	-11.96%	-11.96%	**32.42%**

A Figura VII.4 ilustra através de um gráfico de barras a comparação entre os resultados com-

[1] Autoria dos dados dessa coluna são de Ambrosio e Martello, [2].
[2] Autoria dos dados dessa coluna são de Ambrosio e Martello, [2].
[3] Autoria dos dados dessa coluna são de Ambrosio e Martello, [2].
[4] Autoria dos dados dessa coluna é de Pereira.
[5] Autoria dos dados dessa coluna é de Pereira.

putacionais dos algoritmos propostos para o problema $NLKP_2$ para variáveis inteiras e dos algoritmos utilizados como referência.

Ao analisarmos as instâncias de grande porte, tamanhos 1000 e 2500, na Figura VII.4, notamos novamente que a eficácia do BRKGA, com e sem busca local, apresentam resultados de pior qualidade em relação aos métodos da literatura.

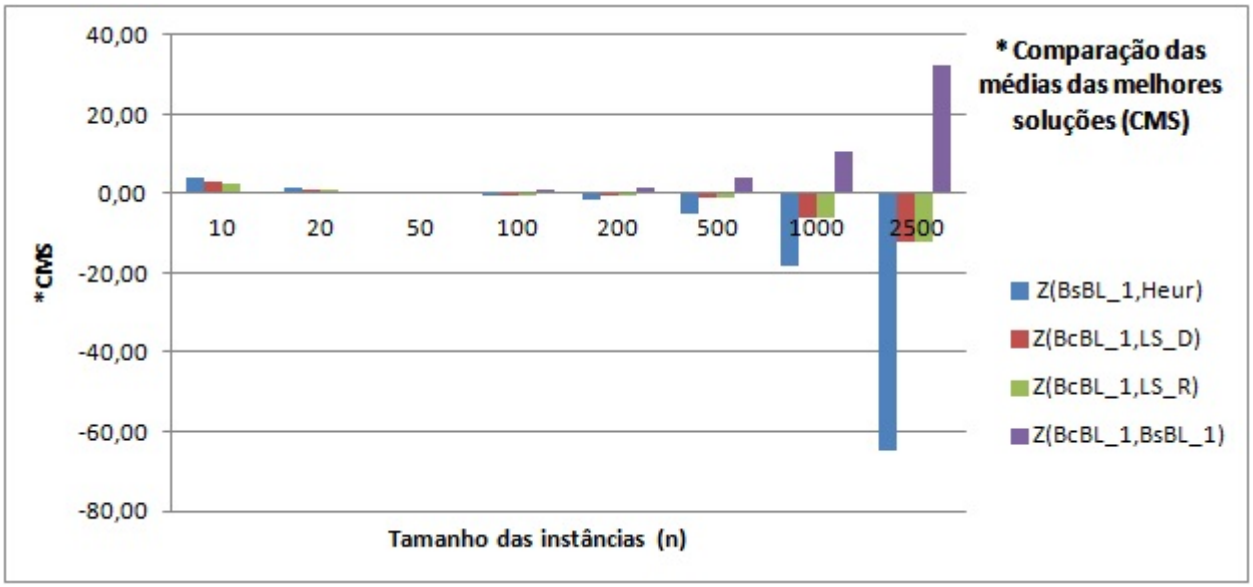


Figura VII.4: Gráfico comparativo entre as médias das melhores soluções para o problema $NLKP_2$ com variáveis inteiras.

VII.4. Classe $NLKP_2$ com Variáveis Reais

A Tabela VII.7, que vem a seguir apresenta as médias dos valores das melhores soluções das heurísticas de referências e das propostas, considerando 20 instâncias de mesmo tamanho para o $NLKP_2$ com variáveis reais. A tabela VII.7 evidencia resultados promissores para vários valores de n com a utilização do BRKGA, com e sem busca local, como forma de solução aproximada para o problema $NLKP_2$ com variáveis reais. Na Tabela VII.7, notamos que o $BcBL_2$ apresentou melhores médias de soluções para três grupos de instâncias de pequeno porte, tamanhos 10, 20 e 50.

Tabela VII.7: Médias das melhores soluções para o $NLKP_2$ utilizando variáveis reais.

n	$Heur$ [1]	LS_D [2]	LS_R [3]	$BsBL_2$ [4]	$BcBL_2$ [5]
10	297.81	300.76	300.77	308.97	**308.98**
20	595.59	600.78	600.30	604.95	**605.81**
50	1600.36	1608.99	1609.11	1600.62	**1609.5**
100	3248.27	3262.23	**3262.60**	3223.79	3252.14
200	6564.74	6586.50	**6586.88**	6449.28	6542.9
500	16386.61	16443.68	**16443.81**	15534.35	16176.28
1000	32941.26	**33066.03**	33065.84	27800.89	30891.54
2500	81742.56	**82099.94**	82099.55	49533.71	72978.36

Na TabelaVII.8, efetuamos a comparação entre os resultados computacionais da heurística proposta, com e sem busca local, e os resultados dos algoritmos $Heur$, LS_D e LS_R, para o problema $NLKP_2$ com variáveis reais. Observamos mais uma vez que, também, para essa classe do problema $NLKP$ o comportamento do algoritmo proposto para instâncias de tamanho até 500 variáveis é satisfatório, sendo que o módulo do desvio máximo para o caso dos resultados com a implementação do BRKGA com busca local é de 1.65%.

Na classe $NLKP_2$ com variáveis reais e nas outras classes apresentadas anteriormente, os algoritmos propostos para pequenas e médias instâncias do problema $NLKP$ produzem resultados que ganham ou ficam a pequena distância dos limitantes superiores calculados na literatura.

Tabela VII.8: Comparações entre as médias das melhores soluções para o $NLKP_2$ utilizando variáveis reais.

n	$Z(BsBL_2, Heur)$	$Z(BcBL_2, LS_D)$	$Z(BcBL_2, LS_R)$	$Z(BcBL_2, BsBL_2)$
10	**3.61%**	**2.66%**	**2.66%**	0.00%
20	**1.55%**	**0.83%**	**0.91%**	**0.14%**
50	**0.02%**	**0.03%**	**0.02%**	**0.55%**
100	-0.76%	-0,31%	-0.32%	**0.87%**
200	-1.79%	-0.67%	-0.67%	**1.43%**
500	-5.49%	-1.65%	-1.65%	**3.97%**
1000	-18.49%	-7.04%	-7.04%	**10.00%**
2500	-65.02%	-12.50%	-12.50%	**32.13%**

A Figura VII.5 ilustra através de um gráfico de barras a comparação entre as médias das melhores soluções produzidas pelos algoritmos propostos para o problema $NLKP_2$ para variáveis reais e, também, as médias das melhores soluções produzidas pelos algoritmos utilizados como referência.

Na Figura VII.5 ao analisarmos as instâncias de grande porte, tamanhos 1000 e 2500, notamos que o BRKGA, com e sem busca local, não apresenta bons resultados em relação aos resultados da literatura.

O AG tem por característica funcionar bem com instâncias até algumas centenas de variáveis de entrada, [37] e [49]. No nosso trabalho, para o caso das maiores instâncias, temos 10 e 25 dezenas de centenas de variáveis sendo manipuladas pelo algoritmo, ou seja, *strings* muito grandes de entrada. Observamos também que a função objetivo utilizada nas classes $NLKP_1$ e $NLKP_2$ e a restrição da classe $NLKP_2$ são complexas e de difícil resolução. Portanto, esses fatores corroboram e explicam a baixa eficácia do BRKGA, nas classes do $NLKP$ analisadas, no caso das instâncias de grande porte.

[1] Autoria dos dados dessa coluna são de Ambrosio e Martello, [2].
[2] Autoria dos dados dessa coluna são de Ambrosio e Martello, [2].
[3] Autoria dos dados dessa coluna são de Ambrosio e Martello, [2].
[4] Autoria dos dados dessa coluna é de Pereira.
[5] Autoria dos dados dessa coluna é de Pereira.

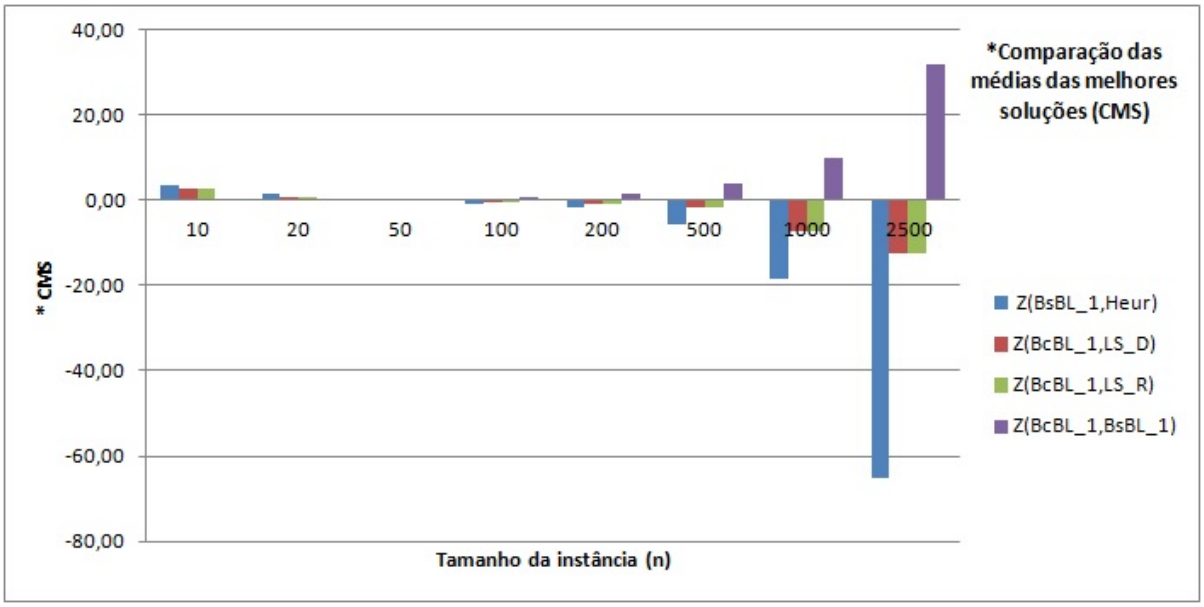


Figura VII.5: Gráfico comparativo das melhores soluções para o problema $NLKP_2$ com variáveis reais.

Capítulo VIII - Conclusões

A área de otimização busca encontrar soluções em um conjunto de soluções viáveis de um domínio discreto ou contínuo de uma determinada função objetivo. Utilizamos um algoritmo genético de chaves aleatórias viciadas para a solução aproximada de um problema da mochila não linear separável e não convexo. Resultados computacionais ilustram a eficiência do algoritmo proposto em comparação com resultados da literatura.

Neste trabalho apresentamos uma proposta de solução do problema da mochila não linear separável utilizando o *framework* BRKGA. Oferecemos uma proposta de solução para as 4 classes do problema e sugerimos um modelo de decodificador com busca local $2-$opt para cada classe. Esse decodificador converte uma solução BRKGA não viável em uma solução viável $NLKP$ e, ao mesmo tempo, aplica um procedimento de busca local que realiza uma busca intensificada na vizinhança da melhor solução do conjunto elite.

Observamos que os resultados computacionais apresentaram valores satisfatórios para instâncias com até 500 variáveis para todas as classes do problema $NLKP$. A principal contribuição desta pesquisa é a aplicação do *framework* BRKGA ao problema $NLKP$ e a obtenção de novos *upper bound* para várias instâncias investigadas.

Os algoritmos propostos para pequenas e médias instâncias do problema $NLKP$ produzem resultados que ganham ou ficam a pequena distância dos limitantes superiores calculados na literatura, o que demonstra eficiência considerando que as classes do problemas analisados são muito difícieis, tanto do ponto de vista teórico como prático. A simplicidade da implementação do *framework* BRKGA, e a sua eficácia, o torna um promissor candidato por extensão para utilização como proposta de solução para outros problemas não lineares.

Como trabalhos futuros destacamos as seguintes propostas:

- Desenvolver uma estratégia de *Path-Relinking* para o problema em substituição a busca local $2-$opt.
- Desenvolver um procedimento de busca local que opera em pares de variáveis semelhante ao proposto por Ambrosio e Martello para fazer uma melhoria de pós-processamento no algoritmo proposto neste trabalho.
- Investigar as demais classes de problemas $NLKP$ existentes em [2] que não foram estudadas

nesta dissertação.

- Utilizar as melhores soluções encontradas como ponto de partida para métodos exatos.
- Resolver o problema $NLKP$ pelo método do enxame de partículas (em inglês *particle swarm optimization* ou PSO).

Referências Bibliográficas

[1] AMBROSIO, C. D'. DEIS - Operations Research Group Library of Instances. Disponível em: $<$ http://www.or-.deis.unibo.it/research_pages/ORinstances/NLKP.html $>$. Acesso em: 30 maio 2014.

[2] AMBROSIO, C. D'; MARTELLO, S. "Heuristic algorithms for the general nonlinear separable knapsack problem". *Computers and Operations Research*, v. 38, pp. 505–513, Feb. 2011.

[3] ARENALES, M. N.; MORABITO, R. "Abordagens para o problema de carregamento de contêineres". *Pesquisa Operacional*, v. 17, n.1, pp. 29–56, 1997.

[4] ARENALES, M. N.; MORABITO, R.; YANASSE, H. H. Mini curso 3: problemas de corte e empacotamento. In: *XXXVI Simpósio Brasileiro de Pesquisa Operacional*, pp. 2690–2769, São João del Rei, M. G., nov. 2004.

[5] BEAN, J. C. "Genetic algorithms and random keys for sequencing and optmization." *ORSA, Journal On Computing*, v. 6, n. 2, pp. 154–160, Jan. 1994.

[6] BELLMAN, R. "The Theory Dynamic Programming". *Bull. Amer. Math. Soc.*, v. 60, n. 6, pp. 503–515, 1954.

[7] BLICKLE, T. *Theory of evolutionary algorithms and application to system synthesis*. PhD thesis, Swiss Federal Institute of Technology, Zurich, Swiss, 1996.

[8] BOLETA, D. A. F. et al. "Uma heurística para o problema de corte de estoque unidimensional inteiro." In: *XXXVII Simpósio Brasileiro de Pesquisa Operacional*, pp. 1869–1879, Gramado, R. S., set. 2005.

[9] BORTFELD, A.; WINTER, T. "A genetic algorithm for the two-dimensional knapsack problem with rectangular pieces." *International Transactions in Operational Research.*, v. 16, n.6, pp. 685–713, Nov. 2009.

[10] BRETTHAVER, K. M.; SHETTY, B. "The non linear knapsack problem algorithms and applications." *European Journal of Operational Research*, v. 138, n. 3, pp. 459–472, May 2002.

[11] CAMARGO, V. C. B.; MATTIOLLI, L.; TOLEDO, F. M. B. “A kanapsack problem as a tool to solve the production planning problem in small foundries.” *Computers & Operations Research*, v. 39, n. 1, pp. 86–92, Jan. 2012.

[12] CAVALCANTI, M. A. P. Algoritmos genéticos princípios e aplicações, 1999. Disponível em: < http:// www.ica.ele.puc-rio.br >. Acesso em: 26 nov. 2014.

[13] CHHAJED, T. J.; LOWE, T. J., Eds. *Building Intuition:* Insights from Basic Operations Management Models and Principles, 2008.

[14] DANTZIG, G. B. “Discrete variable extremum problems.” *Operations Research*, v. 5, n. 2, pp. 266–277, Apr. 1957.

[15] DARWIN, C. *A Origem das Espécies e a Seleção Natural.* São Paulo: Martin Claret, 2014.

[16] DORIGO, M. Metaheuristics network. Disponível em: < http://www.metaheuristics.net >. Acesso em: 30 maio 2015.

[17] DUDZIŃSKI, K.; WALUKIEWICZ, S. “Exact methods for the knapsack problem and its generalizations.” *European Journal of Operational Research*, v. 28, n.1, pp. 3–21, Jan. 1987.

[18] FAMPA, M.; PIMENTEL, W. “A genetic algorithm to the strategic princing problem in competitive electricity markerts.” In: *Proceedings of SBPO/CLAIO*, pp.24–28, 2012.

[19] FERREIRA,P. A. V. Convexidade. Disponível em: <http://www.dt.fee.unicamp.br/tiago/courses/otimizacao_nao_linear/Cap4-Convexidade.pdf. > Acesso em: 30 dez. 2014.

[20] GALLIEN, J. et al. “Initial shipment decisions for new products at Zara.” *Operations Research*, v. 63, n. 2, pp. 269–286, Feb. 2015.

[21] GAREY, M. R.; JOHNSON, D. S. *Computers and Intractability*: a Guide to the Theory of NP-Completeness. New York: W. H. Freeman, 1979.

[22] GERASCH, T. E.; WANG, P. Y. “A survey of parallel algorithms for one dimensional integer knapsack problems.” *Information Systems and Operational Research* , v. 32, n. 3, pp. 163–186, 1993.

[23] GILMORE, P. C.; GOMORY, R. E. “The teory and computation of knapsack functions.” *Operations Research*, v. 14, n. 6, pp. 1045–1074, Dec. 1966.

[24] GOLDBERG, D. E. *Algorithms in Search, Optimization and Machine Learning*. Boston: Addison Wesley, 1989.

[25] GONCALVES, J. F.; RESENDE, M. G. C. “Biased random-key genetic algorithms for combinatorial optimization.” *Journal of Heuristics*, v. 17, n.5, pp. 487–525, Oct. 2011.

[26] GONCALVES, J. F.; RESENDE, M. G. C.; TOSO, R. F. “Biased and unbiased random-key genetic algorithms: An experimental analysis.” *AT&T Labs Research*, Florham Park, 2012.

[27] GONCALVES, J. F.; RESENDE, M. G. C. “A parallel multi-population biased random-ke y genetic algorithm for a container loading problem.” *Computers and Operation Research*, v. 39, n. 2, pp. 179–190, Feb. 2012.

[28] HERRERA, F.; LOZANO, M.; VERDEGAY, J. L. “Tackling real-coded genetic algorithms: Operators and tools for behavioural analysis.” *Artificial Intelligence Review*, v.12, n.4, pp. 265–319, Aug. 1998.

[29] HIFI, M.; MHALLAH R., eds. “Special issue on knapsack problems and applications.” *Computers and Operations Research*, v. 39, n. 1, pp. 1–126, Jan. 2012.

[30] HOCHBAUM, D. S. “A nonlinear Knapsack problem.” *Operations Research Letters*, v. 17, pp. 103–110, 1995.

[31] HOLLAND, J. H. *Adaptation in Natural and Artificial Systems*: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence. Cambridge: MIT Press, 1992.

[32] HORST, R.; TUY, H. *Global optimization*: deterministic approaches. 3. ed. Berlin, Germany : Springer, 1990.

[33] HRISTAKEVA, M.; SHRESTHA, D. Solving the 0–1 knapsack problem with genetic algorithms. Disponível em: <www.micsymposium.org/mics_2004/Hristake.pdf>. Acesso em: 10 dez 2014.

[34] KATOH, N.; IBARAKI, T. *Resource allocation problems:* Algorithmic Approaches. Cambridge: MIT Press, 1988.

[35] KEIJZER, M. et al. “Evolving objects: A general purpose evolutionary computation library.” *Artificial Evolution*, v. 2310, pp. 231–242, 2002.

[36] KHURI, S.; BÄCK, T.; HEITKÖTTER, J. “The zero one multiple knapsack problem and genetic algorithm.” In: *Proceedings of the 1994 ACM Symposium on Applied Computing*, pp. 188–193, New York, 1994.

[37] LEMONGE, A. C. C. *Aplicação de algoritmos genéticos em otimização estrutural*. Tese de Doutorado em Engenharia Civil, Universidade Federal do Rio de Janeiro, Rio de Janeiro, Brasil, 1999.

[38] LIN, E. Y. H. “A bibliographical survey on some well-known non-standard knapsack problems.” *Infor*, v. 36, n. 4, pp. 274–317, 1998.

[39] LIN, S. “Computer solutions of the traveling salesman problem.” *Bell System Technical Journal*, v. 44, n. 10, pp. 2245–2269, Dec. 1965.

[40] LINDEN, R. *Algoritmos Genéticos*. 3ª ed. Rio de Janeiro: Editora Ciência Moderna, 2012.

[41] LOPES, H. S.; RODRIGUES, L. C. de A.; STEINER, M. T. A., eds. *Meta-Heurísticas em Pesquisa Operacional*. Curitiba: Omnipax, 2013.

[42] MITCHELL, M. *An introduction to Genetic Algorithms.* Cambridge: MIT-Press, 1996.

[43] MARTELLO, S.; TOTH, P. *Knapsack problems:* Algorithms and Computer Implementation. New York: John Wiley, 1990.

[44] MATSUMOTO, M.; NISHIMURA, T. "Mersene twister: A623- dimensionally equidistributed uniform peseudotandom number generator." *ACM Transactions on Modeling and Computer Simulation*, v. 8, n. 1, pp.3–30, Jan. 1998.

[45] PIMENTEL, W. *Um Estudo de um Problema de Programação em dois Níveis: O Problema de Estratégia de Preço sob Incerteza em Mercado de Energia.* Tese Doutorado em Engenharia de Sistemas e Computação, Universidade Federal do Rio de Janeiro, Rio de Janeiro, Brasil, 2014.

[46] RESENDE, M. G. C. "Introdução aos algoritmos genéticos de chaves aleatórias viciadas." In: *XLV Simpósio Brasileiro de Pesquisa Operacional*, pp. 3680–3691, 2013.

[47] SILVA, R. M. S. A.; RESENDE, M. G. C. "Algoritmo genético de chaves aleatórias viciadas para o problema de otimização global com restrições de caixa sujeitas a restrições não lineares." In: *XLV Simpósio Brasileiro de Pesquisa Operacional*, pp. 16–19, 2013.

[48] SALKIN, H. M.; KLUYVER, C. A. De. "The knapsack problems: a survey." *Naval Research Logistic Quarterly*, v. 22, n.1, pp. 127–144, Mar. 1975.

[49] SILVA, F. B. S. *Algoritmos genéticos para otimização de estruturas reticuladas baseadas em modelo adaptativos e langrageano aumentados*. Dissertação de Mestrado em Modelagem Computacional, Universidade Federal de Juiz de Fora, Juiz de Fora, Brasil, 2011.

[50] SPEARS, W. M., JONG, K. A. D. "On the virtues of parameterized uniform crossover." In: *Proceedings of the Fourth International Conference on Genetic Algorithms*, pp. 230–236, 1991.

[51] SZETO, K. Y.; LO, M. H. "An application of adaptative genetic algorithm in financial knapsack" problem. In: *Innovations in Applied Artificial Intelligence*, v. 3029, pp. 1220–1228, 2004.

[52] TAHA, H. A. *Pesquisa Operacional*. São Paulo: Pearson, 2008.

[53] TASKIRAN, G. K. *An improved genetic algorithm for knapsack problems.* Dissertation Master of Science in Engineering, Wright State University, Ohio, USA, 2010.

[54] RESENDE, M. G. C.; TOSO, R. F. “A C++ aplication programming interface for biased random-key genetic algorithms.” *Optimization Methods and Software*, v. 30, n. 1, pp. 81–93, Jan. 2015.

[55] WRIGHT, A. H. “Genetic algorithms for real parameter optimization.” In: *Foundation of Genetic Algorithms*, pp. 205–218, 1991.

[56] MICHALEWICZ, Z. *Genetic Algorithms+Data Structures=Evolution Programs. 3 ed.* New York: Springer-Verlag, 1992.

[57] ZANG, B.; HUA, Z. S. “A unified method for a class of convex separable non linear knapsack problems.” *European Journal of Operational Research*, v. 191, n.1, pp. 1–6, Nov. 2008.