Metaheurísticas para Problemas de Otimização Combinatória: Dependência de Pacotes e Mochila com Penalidades

Fellipe Sanha

5 de dezembro de 2025

Resumo

Este trabalho apresenta a implementação e análise de metaheurísticas para dois problemas de otimização combinatória: o Problema da Dependência de Pacotes e o Problema da Mochila com Penalidades (KPF). São exploradas abordagens de Programação Linear Inteira (ILP) e o algoritmo BRKGA (Biased Random-Key Genetic Algorithm), com comparação de desempenho em instâncias de diferentes tamanhos.

Sumário

1 Introdução

Neste trabalho abordaremos a descrição de variações de um NP difíceis, similar ao Problema da Mochila(Knapsack Problem - KP), estratégias viáveis para resolver este problema em diferentes recortes temporais, e comparamos os resultados obtidos frente aos resultados ótimos conhecidos na literatura.

2 Problema da Mochila com Penalidades (Knapsack Problem with Forfeits - KPF)

O problema abordado é definido sobre um conjunto I de itens, com cada item i é associado a um peso w_i e um benefício $p_i \in \mathbb{N}$, consiste em encontrar um subconjunto $I' \subseteq I$, sujeito a:

1. A soma dos pesos dos itens escolhidos não ultrapassarem a capacidade $H \in \mathbb{N}$
2. Para um conjunto F de pares de itens f, associa-se uma penalidade d_f ao um par de itens $f = \{i, j\} \subseteq I$. Caso a solução avaliada contenha este par, ela deve ter a penalidade decrementada de seu benefício total.

Desta maneira, o *custo* podemos formular as restrições do problema como

$$\sum_{i=1}^{n} w_i x_i \leq H \tag{1}$$

$$x_i + x_j - v_f \leq 1, \quad \forall f = \{i, j\} \in F \tag{2}$$

$$x_i \in \{0, 1\}, \quad \forall i \tag{3}$$

$$v_f \in [0, 1], \quad \forall f \in F \tag{4}$$

E a função de objetivo como:

$$\max \sum_{i=1}^{n} p_i x_i - \sum_{f \in F} d_f v_f \tag{5}$$

Esta formulação foi apresentada pela primeira vez por (CERULLI et al., 2020), onde foi definido pela primeira vez o Problema da Mochila com Penalidades.

3 Soluções Propostas

Neste capítulo as soluções propostas para o problema serão exploradas, onde os detalhes de implementação e as estratégias escolhidas para tentar elevar os desempenhos serão comentadas.

3.1 Programação Linear Inteira (ILP)

A formulação ILP segue o modelo de Cerulli et al. (CERULLI et al., 2020), utilizando as equações de modelagem da seção acima. A motivação por trás do uso desta solução foram de poder testar soluções fora do escopo de metaheurísticas, e de poder se aproveitar de uma solução que pode garantir otimalidade para o problema.

Outro fator determinante foi a facilidade e confiabilidade da implementação, fazendo uso do framework JuMP(LUBIN et al., 2023), de programação matemática para a linguagem de programação Julia. A implementação foi feita utilizando o solver open source HiGHS(HUANGFU; HALL, 2018), que já vem integrado por padrão junto à biblioteca do JuMP.

A fim de acelerar a convergência do solver, utilizamos uma técnica de *warm start* (YILDIRIM; WRIGHT, 2002), onde uma solução com algum nível de qualidade é fornecido ao solver, permitindo-o ignorar todas as soluções de avaliação inferior mais facilmente.

A solução inicial utilizada no warm start foi gerada usando um algoritmo guloso randomizado, onde a solução melhora avaliada era escolhida entre alguns samples gerados com diferentes valores de α.

3.2 BRKGA (Biased Random-Key Genetic Algorithm)

O Algoritmo Genético de Chaves Enviesadas Aleatórias (Biased Random-Key Genetic Algorithm - BRKGA) é um algoritmo baseado no Algoritmo Genético de Chaves Aleatórias, com uma diferenciação na maneira com que os *pais* são escolhidos em uma dada população (GONÇALVES; RESENDE, 2011)

O algoritmo consiste em gerar uma população de soluções aleatória, *decodificar* esta população, isto é, gerar a representação da solução equivalente, avaliar todos os indivíduos e, similar ao processo de *sobrevivência dos mais aptos*, gerar uma nova população, onde os melhores candidatos têm mais influência.

Um diferencial da implementação desenvolvida foi o fato de que as populações iniciais era geradas através de um GRASP, com um α configurável nos parâmetros. Isso fez com que as soluções iniciais já tivessem uma qualidade maior, mas sem perder o elemento da aleatoriedade, tão importante em metaheurísticas.

A implementação do algoritmo é proprietária, desenvolvida para uma disciplina sobre metaheurísticas na Universidade Federal Fluminense - UFF, pensada inicialmente para o Problema da Mochila com uniões de conjuntos (Set-Union Knapsack Problem - SUKP), mas desenvolvido de maneira genérica para ser utilizado em qualquer problema que fosse descrito seguindo a interface sugerida pelo programa.

Podemos ver nas tabelas 1 e 2

Nome	Descrição
`ProblemContext`	Representa informações de contexto do problema, como capacidade da mochila, pesos e benefícios de itens, etc.
`Solution`	Representa uma solução para o problema
`EncodingStrategy`	Representa uma estratégia de encoding
`evaluate`	Função que recebe o contexto do problema e uma solução, e retorna um número representando a qualidade da solução

Tabela 1: Detalhes sobre a interface genérica da implementação do BRKGA

Nome	Descrição
`encode`	Função que recebe um `ProblemContext` e um `EncodingStrategy` que gera uma população nova e transforma sua representação usual em uma representação de chaves
`decode`	que recebe um `ProblemContext`, uma combinação de chaves representada por um `Array`, e um `EncodingStrategy` e retorna a representação usual da combinação de chaves informada

Tabela 2: Detalhes sobre a interface específica do BRKGA

Na tabela 3 podemos ver as configurações dos parâmetros utilizados pelo BRKGA. Estes parâmetros foram calibrados de modo que o maior fator de parada era o tempo de execução. Também podemos ver, nos parâmetros abaixo, a variável α, usada no algoritmo GRASP de geração de novos indivíduos.

Parâmetro	Valor
Tamanho da população	1000
Viés de crossover	0.5
α (GRASP)	0.7
Iterações máximas	50000

Tabela 3: Parâmetros do BRKGA

4 Comparação de resultados

Nesta seção veremos os desempenhos das diferentes estratégias desenvolvidas. Comparamos, também, os resultados com os ótimos conhecidos da literatura (MOURA; NORONHA; BOGUE, 2021). Todos os algoritmos foram executados em uma máquina 11th Gen Intel Core i7-1165G7 de décima primeira geração, com 4 núcleos e 8 processadores lógicos

4.1 Instâncias de Teste

Foram escolhidas vinte instâncias no total, de quatro tamanhos diferentes, 500 itens, 700 itens, 800 itens, e 1000 itens , visando avaliar o desempenho dos algoritmos desen-

volvidos desde instâncias relativamente pequenas(500 itens) até tamanhos considerados grandes(1000 itens).

4.2 Comparação com ótimos globais

Tamanho	Instância	Ótimo	60s		120s		180s	
			ILP	BRKGA	ILP	BRKGA	ILP	BRKGA
500	1	2626	2145	**2244**	**2336**	2244	**2336**	2244
	2	2660	**2308**	2238	**2324**	2238	**2419**	2242
	3	2516	**2207**	2152	**2299**	2163	**2310**	2198
	4	2556	**2233**	2170	**2254**	2170	**2445**	2185
	5	2625	**2297**	2199	**2353**	2199	**2409**	2218
700	1	3589	**3156**	3059	**3218**	3059	**3299**	3059
	2	3679	2763	**2857**	2775	**2857**	**2913**	2857
	3	3664	**3128**	3124	**3189**	3124	**3206**	3124
	4	3647	**3182**	3085	**3267**	3085	**3267**	3085
	5	3596	**3188**	3082	**3156**	3082	**3128**	3082
800	1	4184	**3400**	3398	3400	**3427**	**3520**	3427
	2	4065	3300	**3371**	3299	**3371**	3299	**3371**
	3	4104	**3475**	3316	**3475**	3388	**3475**	3388
	4	4056	3218	**3349**	3252	**3349**	3320	**3349**
	5	4086	3366	**3389**	3366	**3389**	3386	**3389**
1000	1	4940	3852	**4028**	3916	**4077**	3928	**4077**
	2	4969	4034	**4105**	4034	**4105**	4034	**4105**
	3	5177	4040	**4230**	4211	**4230**	4273	**4230**
	4	5143	4091	**4235**	4137	**4302**	4212	**4302**
	5	5136	4094	**4143**	4019	**4145**	4040	**4145**

Tabela 4: Resultados comparativos por tamanho de instância e tempo limite

4.3 Gaps médios por tamanho de instância

Tamanho	60s		120s		180s	
	ILP	BRKGA	ILP	BRKGA	ILP	BRKGA
500	**13.79**	15.24	**10.88**	15.15	**8.17**	14.58
700	**15.13**	16.31	**14.1**	16.31	**12.97**	16.31
800	18.23	**17.91**	18.07	**17.42**	**17.06**	17.42
1000	19.81	**18.23**	20.22	**17.76**	19.82	**17.76**

Tabela 5: GAP comparativo por tamanho de instância

5 Conclusão

Neste trabalho foram apresentadas e comparadas duas abordagens para resolver o Problema da Mochila com Penalidades (KPF): Programação Linear Inteira (ILP) utilizando o solver HiGHS, e o algoritmo metaheurístico BRKGA.

Os resultados experimentais demonstraram que o desempenho relativo das abordagens varia conforme o tamanho da instância e o tempo disponível para execução. Para instâncias menores (500 e 700 itens), a abordagem ILP apresentou gaps médios inferiores, beneficiando-se de que o tempo necessário para convergir em instâncias menores também é menor para soluções de melhor qualidade. Já para instâncias maiores (800 e 1000 itens), o BRKGA demonstrou superioridade, mantendo gaps mais consistentes mesmo com o aumento da complexidade do problema.

A técnica de warm start aplicada ao solver ILP, utilizando soluções geradas por um algoritmo guloso randomizado, mostrou-se eficaz para acelerar a convergência inicial. Da mesma forma, a estratégia de inicialização da população do BRKGA através do GRASP contribuiu para a qualidade das soluções encontradas.

Como trabalhos futuros, sugere-se a exploração de:

- Melhorar as estratégias de crossover e os critérios de avaliação da geração da solução inicial, possibilitando assim uma variabilidade ainda maior entre as soluções inicial geradas por mutação
- Calibração automática e eficiente dos parâmetros do BRKGA através de técnicas como irace ou SMAC, ou uma calibração paramétrica, que por exemplo varie o tamanho do α utilizado no GRASP varie com o tamanho da instância
- Implementação de operadores de busca local para refinamento das soluções do BRKGA
- Avaliação em instâncias de maior porte para verificar a escalabilidade das abordagens

Referências

CERULLI, Raffaele et al. The Knapsack Problem with Forfeits. In: BAÏOU, Mourad et al. (Ed.). **Combinatorial Optimization**. Cham: Springer International Publishing, 2020. P. 263–272. DOI: 10.1007/978-3-030-53262-8_22.

GONÇALVES, José Fernando; RESENDE, Mauricio GC. Biased random-key genetic algorithms for combinatorial optimization. **Journal of Heuristics**, Springer, v. 17, n. 5, p. 487–525, 2011.

HUANGFU, Q.; HALL, J. A. J. Parallelizing the dual revised simplex method. **Mathematical Programming Computation**, v. 10, n. 1, p. 119–142, 2018. DOI: 10.1007/s12532-017-0130-5. Disponível em: <https://doi.org/10.1007/s12532-017-0130-5>.

LUBIN, Miles et al. JuMP 1.0: Recent improvements to a modeling language for mathematical optimization. **Mathematical Programming Computation**, v. 15, p. 581–589, 2023. DOI: 10.1007/s12532-023-00239-3.

MOURA, Ana Flávia Ciríaco; NORONHA, Thiago; BOGUE, Eduardo Theodoro. Uma heurística ILS para o Problema da Mochila com Penalidades. In: ANAIS do LIII Simpósio Brasileiro de Pesquisa Operacional. João Pessoa: Galoá, 2021. Disponível em: <https://proceedings.science/sbpo/sbpo-2021/trabalhos/uma-heuristica-ils-para-o-problema-da-mochila-com-penalidades?lang=pt-br>. Acesso em: 4 dez. 2025.

YILDIRIM, E Alper; WRIGHT, Stephen J. Warm-start strategies in interior-point methods for linear programming. **SIAM Journal on Optimization**, SIAM, v. 12, n. 3, p. 782–810, 2002.