Metaheurística Híbrida HRH Para o Problema da Mochila

Daniel C. Freire
Universidade Federal de Minas Gerais
Belo Horizonte, MG, 31.270-901

1

Resumo. *O problema da mochila é um problema clássico de otimização discreta, que possui muitos paralelos ao mundo real, o tornando de grande interesse para estudo. Nessa pesquisa foi implementada uma metaheurística hibrida no modelo HRH para o problema da mochila 0-1 unidimensional. A metaheurística proposta consiste em realizar a execução de um algoritmo genético para selecionar soluções em regiões de alta performance, seguido de uma busca tabu, para explorar essas regiões e encontrar o ótimo local. A apresentação dessa pesquisa pode ser encontrada em* `https://www.youtube.com/watch?v=KVwEzizobKQ`

1. Introdução

Otimização discreta é um dos ramos de pesquisa operacional, que lida com o espaço de solução discreto. Um dos problemas mais importantes em otimização discreta é o problema da mochila (KP). Nesse problema temos uma mochila, que possui uma certa capacidade, e queremos selecionar itens para colocar na mochila de forma a maximizar o valor total de itens na mochila sem ultrapassar sua capacidade.

Existem diversas variações no problema da mochila, como o problema da mochila n-dimensional, o problema da mochila com múltiplas mochilas, etc. Nesse projeto propomos explorar o problema da mochila 0-1 unidimensional (KP01), no qual temos uma mochila que possui um peso máximo que ela pode carregar, e n itens, cada um com seu valor e peso, os quais devemos decidir se entram na mochila ou não.

O KP01 pode ser formulado pela programação inteira abaixo. Nesse modelo, X_i é a variável de decisão binária, que é 1 se o item i for selecionado e 0 caso contrário. p_i e w_i são o valor e o peso do item respectivamente e W é a capacidade da mochila.

$$\max \sum_{i=1}^{n} p_i \cdot X_i$$

sujeito a

$$\sum_{i=1}^{n} w_i \cdot X_i \leq W$$

$$X_i \in \{0, 1\}$$

KP01 é NP-Difícil, ou seja não existe algoritmo eficiente para resolvê-lo, a não ser que $P = NP$. Existem, porém, algoritmos pseudo-polinomiais que o resolvem, como o algoritmo de programação dinâmica.

Nessa pesquisa, KP01 foi escolhido por dois motivos. Primeiro por ser um problema fundamental em computação, com muitas aplicações práticas, como controle de inventário, modelagem financeira, controle de trafego em sistemas de telecomunicação, etc. Segundo, por ser um problema NP-Difícil, que é uma classe de problemas na qual é muito benéfico utilizar metaheurísticas, para produzir boas soluções em tempo viável.

Nessa pesquisa foi implementada uma metaheurística hibrida de modelo HRH (high-level relay hybrid), no qual primeiro procuramos regiões de alta-performance utilizando um algoritmo genético, e em seguida, procuramos nessas regiões pelo ótimo local utilizando busca tabu. Espera-se que a combinação de algoritmos genéticos com busca tabu dessa maneira possibilite que boas soluções sejam encontradas.

Na seção 2 iremos falar sobre os trabalhos feitos estudando metaheurísticas para KP01. Na seção 3 a nova metaheurística é proposta, e em seguida na seção 4 são realizados os experimentos. Finalmente, na última seção, está a conclusão e sugestões para pesquisas futuras.

2. Trabalhos Relacionados

Recentemente muitas metaheurísticas foram criadas para o problema da mochila 0-1. Nessa seção iremos falar sobre algumas delas.

Sapre et al. [Sapre et al. 2019] propôs uma versão do algoritmo de inteligência coorte juntamente com regras baseadas em viabilidade para solucionar casos pequenos de KP01. Escolhendo os candidatos de maneira educada eles foram capazes de obter a solução ótima para a maioria dos casos explorados.

Gómez-Herrera et al. [Gómez-Herrera et al. 2018] descreveu duas novas heurísticas e uma hiper-heurística para problema da mochila 0-1 n-dimensional. A hiper-heurística seleciona a heurística que vai utilizar baseado em características da instância, e dessa forma possibilita que a melhor heurística seja utilizada para cada instância.

Gao et al. [Gao et al. 2018] propôs uma algoritmo de matilha de lobos inspirado em física quântica para o problema da mochila 0-1 n-dimensional. Esse método possui duas operações importantes, rotação quântica e colapso quântico. O primeiro ajuda a população a mover em direção ao ótimo global, e o segundo ajuda a impedir que indivíduos fiquem presos em ótimos locais.

Abdel-Basset et al. [Abdel-Basset et al. 2019] propôs uma metaheurística de otimização de baleia para o problema da mochila 0-1. O método, baseado na natureza, simula o comportamento social de baleias jubarte. Essa metaheurística é eficiente para resolver problemas continuos e foi adaptada para o problema da mochila. Uma função de penalização foi adicionada para que a fitness das soluções viáveis prevaleçam sobre as soluções inviáveis. Além disso, soluções inviáveis são viabilizadas por um operador de reparo em dois estágios. Essa metaheurística foi demonstrada em ser eficiente, efetivo e robusto para resolver problemas da mochila 0-1 difíceis.

Na seção seguinte iremos detalhar a metodologia utilizada nessa pesquisa.

3. Metodologia

3.1. Controle

Foram utilizados seis algoritmos para o grupo controle dos experimentos. Três são componentes da metaheurística hibrida: O algoritmo genético, a busca tabu, e o algoritmo guloso randomizado. Além disso esses com esses três algoritmos foram implementados uma busca tabu com solução inicial aleatória, e uma população gulosa aleatória. Finalmente, com o intuito de demonstrar a importância de uma metaheurística para o problema, comparamos os tempos de execução com os do algoritmo de programação dinâmica, que produz uma solução exata, em tempo pseudo-polinomial.

Algorithm 1 Dynamic Programming KP

Input: w - An array containing the weights of the items
p - An array containing the value of the items
W - The capacity of the bag
Output: The maximum value of the items in the bag
$n \leftarrow |w|$
array $m[0..n, 0..W]$ initialized to 0
for i from 1 to n **do**
  for j from 0 to W **do**
    if $w[i] > j$ **then**
      $m[i, j] \leftarrow m[i-1, j]$
    else
      $m[i, j] \leftarrow \max(m[i-1, j], m[i-1, j-w[i]] + p[i])$
    end if
  end for
end for
return m[n, W]

3.2. Heurística proposta

A metaheurística proposta é uma metaheurística hibrida no esquema HRH. Ela é composta por duas metaheurísticas, um algoritmo genético e uma busca tabu. O algoritmo HRH está descrito no algoritmo 2.

Em KP_HRH começamos produzindo soluções distintas pelo algoritmo genético (Algoritmo 3), o objetivo disso é encontrar regiões de alta performance no espaço de soluções. Em seguida usamos a busca tabu (Algoritmo 6) para explorar tais regiões com o objetivo de encontrar o ótimo local.

Algorithm 2 KP_HRH

Input: w - An array containing the weights of the items
p - An array containing the value of the items
W - The capacity of the bag
k_{ga} - The stopping condition for the genetic algorithm
k_{tabu} - The stopping condition for the tabu search
l - Number of solutions that the genetic algorithm should return
α - How random the item selection is for the initial population
p_size - The population size
mutation_rate - The rate of mutation
Output: Best solution found
$S \leftarrow$ KP_GA$(w, p, W, k_{ga}, l, \alpha, p_size, mutation_rate)$
best_sol $\leftarrow 0$
for each s_i in S **do**
sol $\leftarrow$ KP_TABU(w, p, W, s_i, k_{tabu})
if sol $>$ best_sol **then**
best_sol $\leftarrow$ sol
end if
end for
return m[n, W]

Algorithm 3 KP_GA

Input: w - An array containing the weights of the items
p - An array containing the value of the items
W - The capacity of the bag
k - The stopping condition
l - Number of solutions that to return
α - How random the item selection is for the initial population
p_size - The population size
mutation_rate - The rate of mutation
Output: Top l solutions
Initialize the population $P(t)$ with p_size elements using KP_GREEDY(w, p, W, alpha)
Evaluate fitness of $P(t)$
while improvement has been made in the last k iterations **do** ▷ *x*
Select solutions from $P(t)$ proportionate to fitness
Recombine solutions
Mutate solutions by mutation_rate
$P(t+1) \leftarrow$ P(t) with worst solutions replaced by the best solutions of the offspring
Evaluate fitness of $P(t+1)$
$t \leftarrow t+1$
end while
return Top l distinct solutions in $P(t)$

Algorithm 4 KP_GREEDY

Input: w - An array containing the weights of the items
p - An array containing the value of the items
W - The capacity of the bag
α - The parameter that determines how random the item selection is
Output: Random greedy solution
Initialize the list D containing the cost density of each item
while The bag is not full **do** ▷ *x*
lrc_cutoff $\leftarrow (\max D) - alpha * (\max D - \min D)$
selected_item $\leftarrow$ Random item with cost density greater than or equal to lrc_cutoff
Add the selected_item to the bag if it has the capacity to do so
Remove selected_item from D
end while
return the selected items

Algorithm 5 KP_RANDOM_START_TABU

Input: w - An array containing the weights of the items
p - An array containing the value of the items
W - The capacity of the bag
α - The parameter that determines how random the item selection is
k - The stopping condition
s - The initial solution set size
t - Size of the tabu list
Output: Best solution found
Initialize the set P of size s with KP_GREEDY(w, p, W, alpha)
Sol $\leftarrow \{\}$
for each initial_solution in P **do**
Sol $\leftarrow$ Sol $\cup$ {KP_TABU(p, w, W, k, initial_solution, t)}
end for
return $\max$ Sol

Algorithm 6 KP_TABU

Input: w - An array containing the weights of the items
p - An array containing the value of the items
W - The capacity of the bag
k - The stopping condition
s - Initial solution
t - Size of the tabu list
Output: Best solution found
Initialize the tabu list of size t
while improvement has been made in the last k iterations **do** ▷ *x*
Evaluate the neighborhood of s for viability and presence in the tabu list
$s \leftarrow$ best candidate solution in the neighborhood
Update tabu list
end while
return s

4. Experimentos e Resultados

Os experimentos foram realizados sobre as instâncias disponibilizadas em `http://hjemmesider.diku.dk/~pisinger/codes.html` descritas no artigo "Where are the hard knapsack problems?" [Pisinger 2005]. O ajuste de parâmetros foi feito sobre um subconjunto dessas instâncias, ajustando cada parâmetro individualmente e observando qual produz as melhores soluções, e estão listados na tabela 1.

Parameter	Value
k_{ga}	10
k_{tabu}	70
l	50
α	0.9
p_size	100
mutation_rate	0.5
t	50

Tabela 1. Parâmetros utilizados nos experimentos.

Essas instâncias foram categorizadas em 13 tipos de problema da mochila: Sem correlação (UNCOR), correlação fraca (WCOR), correlação forte (SCOR), correlação forte inversa (ISCOR), quase fortemente correlacionado (ASCOR), problema da soma dos subconjuntos (SUBSUM), pesos similares (SIMW), intervalo não correlacionado(2,10) (UNCORS) , intervalo com correlação fraca(2,10) (WCORS), intervalo fortemente correlacionado(2,10) (SCORS), mstr(3R/10, 2R/10, 6) (MSTR), pceil(3)(PCEIL), circle(2/3) (CIRCLE). Para detalhes sobre cada um desses tipos de instância, consulte o artigo [Pisinger 2005].

4.1. Resultados

Primeiro foi avaliado o tempo de execução para o KP_DP, como KP_DP é $O(nW)$, onde n é o número de itens, e W é a capacidade da mochila, foi traçado um gráfico do tempo de execução versus nW (Fig. 1). Além disso, foi aproximada uma reta aos pontos obtidos. A reta pode ser utilizada para estimar o tempo de execução para instâncias grandes para KP01. Com a reta obtida, foi estimado que para obter a solução exata da instância com o maior nW, KP_DP iria executar por aproximadamente 28 horas.

Cada método foi executado sobre cada uma das mais de 30 mil instâncias, e foram medidos o tempo de execução e o resultado de cada heurística. Os resultados serão apresentados e discutidos a seguir.

Na figura 2 foi traçado um gráfico mostrando o tempo de execução versus n, para cada heurística . Podemos observar do gráfico traçado que todas as heurísticas performam bem, sendo a HRH proposta a mais lenta, e a heurística gulosa a mais rápida. Notamos que todas as heurísticas implementadas são muito mais rápidas que KP_DP, KP_HRH demora 40 segundos para encontrar uma solução para instâncias em que KP_DP demoraria mais de 25 horas.

A tabela 2 mostra o erro de aproximação percentual médio para cada heurística e cada tipo de instância. Nela observamos que a busca tabu obteve a melhor precisão em

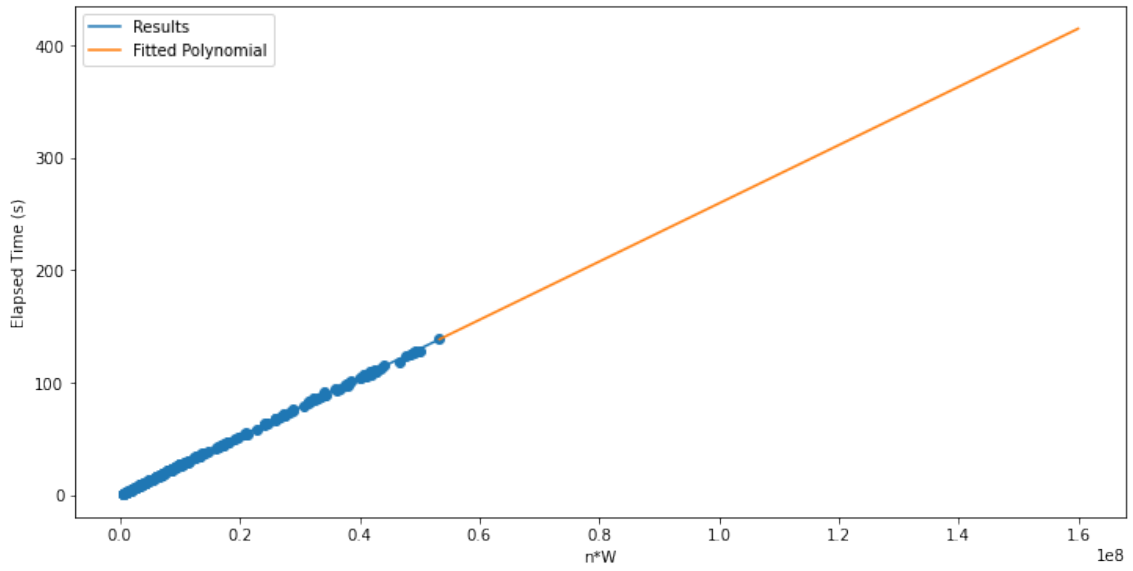

Figura 1. Tempo de execução de KP_DP versus nW. Os pontos em azul são os resultados experimentais, a reta laranja é a reta ajustada para os resultados experimentais.

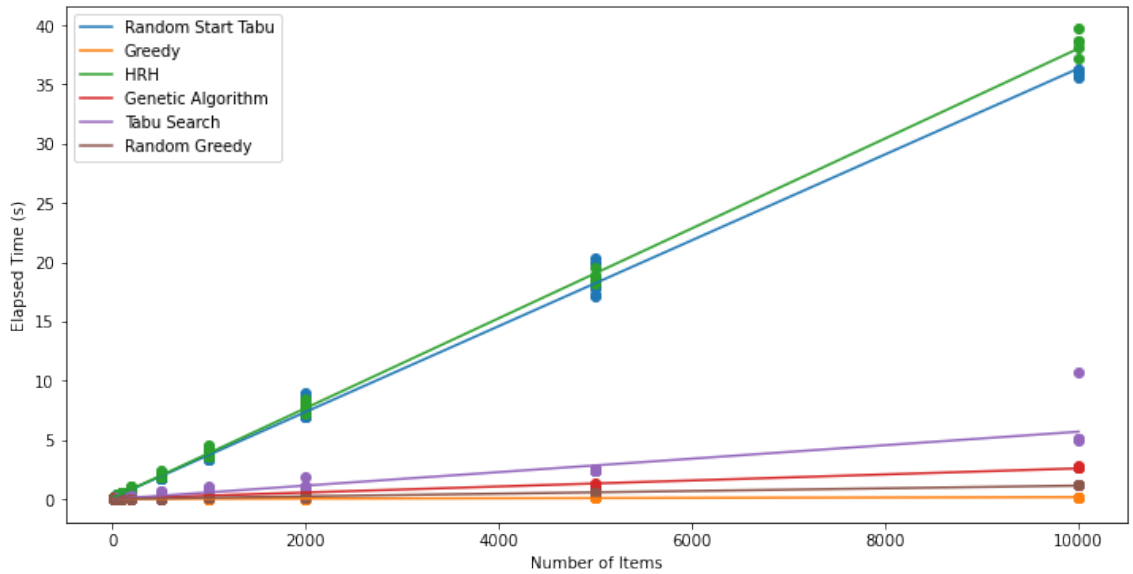

Figura 2. Tempo de execução de cada heurística implementada.

quase todas as instâncias, apenas para problemas de soma de subconjuntos (SUBSUM), que são instâncias onde o peso e o valor são proporcionais, que HRH obteve uma precisão melhor. Nos demais casos a heurística gulosa obteve resultados melhores que HRH também, perdendo apenas nas instâncias do tipo PCEIL, e UNCOR. Portanto, em geral, a heurística que obteve a melhor precisão foi a busca tabu, seguido da heurística gulosa, seguido de HRH.

A figura 3 apresenta o erro percentual de cada heurística para todas as instâncias, limitadas por $nW < 10^{13}$. Para valores pequenos de nW o erro de todas as heurísticas

implementadas é alto. Como a precisão HRH é altamente dependente do tipo de instância, é difícil dizer para quais valores de nW o método passa a ter uma boa precisão em geral. Em comparação, a busca tabu para valores de $nW > 10^{10}$ possui erro percentual menor que 3%, enquanto para os mesmos valores de nW há instâncias para qual HRH obteve erro de mais de 20%. Porém, se considerarmos um subconjunto dos tipos de instância, observamos uma melhora significativa na precisão de HRH.

	Rand Tabu	Greedy	HRH	GA	Tabu	Rand Greedy
UNCOR	0.300708	0.136695	0.110173	0.300017	0.067299	0.561236
WCOR	3.242019	0.210425	2.585577	3.232292	0.159128	3.782059
SCOR	3.149615	1.029683	2.476493	2.738927	0.548970	3.225537
ISCOR	3.812786	0.081114	3.318492	3.738212	0.032542	4.192252
ASCOR	3.162528	1.028030	2.492596	2.754301	0.568247	3.235294
SUBSUM	0.000082	0.175618	0.000076	0.003176	0.017757	0.004765
SIMW	0.008837	0.000006	0.003320	6.243186	0.000000	9.616048
UNCORS	1.389667	0.436869	1.076429	3.682819	0.029181	5.875631
WCORS	1.360307	0.654462	1.139805	2.070331	0.111652	2.568888
SCORS	2.189002	0.622021	1.820297	2.329991	0.052657	2.897536
MSTR	2.804147	1.438990	1.836710	2.168995	0.617751	2.973333
PCEIL	0.152674	0.828682	0.149483	0.185526	0.045900	0.203631
CIRCLE	2.252607	1.099117	1.274390	1.689788	0.470072	2.499344

Tabela 2. Erro de aproximação percentual médio de cada heurística para cada tipo de instância.

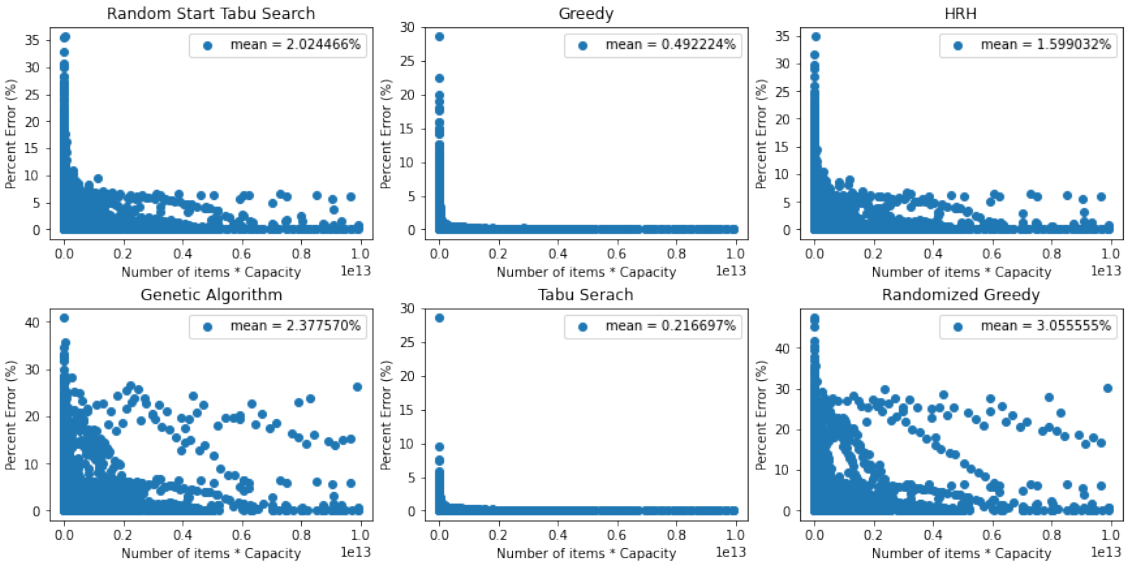

Figura 3. Erro percentual de cada heurística implementada para cada instância onde $nW < 10^{13}$.

Na figura 4 mostramos o erro percentual de cada heurística, para instâncias do tipo SUBSUM, SIMW, PCEIL e com $10^{10} < nW < 10^{13}$. Nesse caso HRH performa significativamente melhor que qualquer outro método, tanto em termos do erro médio, quanto no erro máximo.

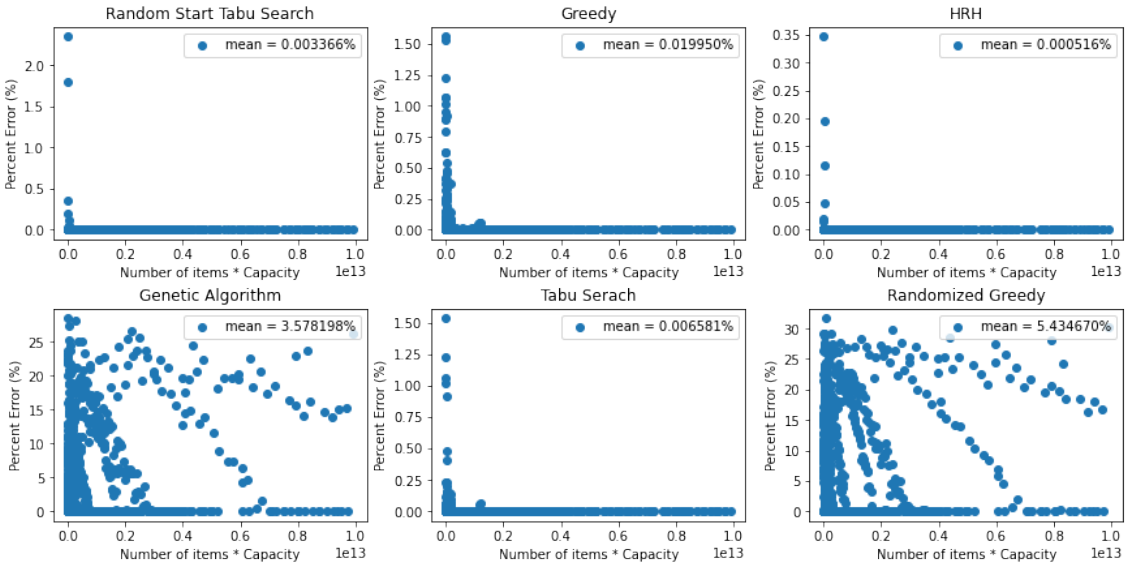

Figura 4. Erro percentual de cada heurística implementada para instâncias do tipo SUBSUM, SIMW, PCEIL. Com $10^{10} < nW < 10^{13}$.

5. Conclusão

Observamos nessa pesquisa que para a maior parte das instâncias o método proposto não é a melhor opção, tanto em tempo de execução, quanto em qualidade dos resultados. Porém, ele pode ser adequado dependendo do tipo de instância e do tamanho de n e W. Em geral, a heurística mais precisa para o problema, dentre as exploradas, é a busca tabu, seguido da heurística gulosa, e ambas são pelo menos uma ordem de magnitude mais rápidas que a metaheurística proposta. Porém, para valores pequenos de n e W sempre faz mais sentido usar o algoritmo exato de programação dinâmica para resolver o problema.

Para trabalhos futuros, seria interessante estudar mais tipos de instância para as quais a metaheurística proposta obtém bons resultados. Além disso, estudar diferentes metaheurísticas populacionais para alimentar a busca tabu no esquema HRH poderia ser útil.

Referências

Abdel-Basset, M., El-Shahat, D., and Sangaiah, A. K. (2019). A modified nature inspired meta-heuristic whale optimization algorithm for solving 0–1 knapsack problem. *International Journal of Machine Learning and Cybernetics*, 10(3):495–514.

Gao, Y., Zhang, F., Zhao, Y., and Li, C. (2018). Quantum-Inspired Wolf Pack Algorithm to Solve the 0-1 Knapsack Problem. *Mathematical Problems in Engineering*, 2018:1–10.

Gómez-Herrera, F., Ramirez-Valenzuela, R. A., Ortiz-Bayliss, J. C., Amaya, I., and Terashima-Marín, H. (2018). A Quartile-Based Hyper-heuristic for Solving the 0/1 Knapsack Problem. In Castro, F., Miranda-Jiménez, S., and González-Mendoza, M., editors, *Advances in Soft Computing*, volume 10632, pages 118–128. Springer International Publishing, Cham.

Pisinger, D. (2005). Where are the hard knapsack problems? *Comput. Oper. Res.*, 32(9):2271–2284.

Sapre, M. S., Patel, H., Vaishnani, K., Thaker, R., and Shastri, A. S. (2019). Solution to Small Size 0–1 Knapsack Problem Using Cohort Intelligence with Educated Approach. In Kulkarni, A. J., Singh, P. K., Satapathy, S. C., Husseinzadeh Kashan, A., and Tai, K., editors, *Socio-cultural Inspired Metaheuristics*, volume 828, pages 137–149. Springer Singapore, Singapore.