

UMA HEURÍSTICA POPULACIONAL HÍBRIDA PARA PROBLEMAS DE SEQUENCIAMENTO DE TAREFAS COM DATAS DE LIBERAÇÃO E RESTRIÇÕES DE INVENTÁRIO

Frederico Guerra, Rafael Morais, Ana Carolina Moura, Bruno Bruck, Anand Subramanian

Universidade Federal da Paraíba

Centro de Informática, Universidade Federal da Paraíba (UFPB) – R. dos Escoteiros, s/n
– Mangabeira, João Pessoa – PB, CEP 58058-600

frederico.souza.guerra@gmail.com, rafaelsobralm@gmail.com,
ana.marques3@academico.ufpb.br, bruno.bruck@ci.ufpb.br, anand@ci.ufpb.br

RESUMO

Problemas de sequenciamento de tarefas são vastamente abordados na literatura e com inúmeras aplicações na indústria. Dentre as variantes dessa classe de problemas, este trabalho aborda o problema de minimização do tempo de conclusão da última tarefa de uma sequência (*makespan*), considerando as restrições de inventário e datas de liberação. Para resolução desse problema, é proposta uma abordagem heurística híbrida baseada em busca populacional. O método proposto inclui procedimentos de intensificação e diversificação, em que o primeiro realiza uma busca local com avaliação eficiente de movimentos, enquanto o segundo consiste em uma estratégia inspirada em *ruin-and-recreate*, além da implementação de um mecanismo de controle de diversidade da população. Experimentos computacionais em um conjunto de 960 instâncias demonstram a competitividade do algoritmo proposto frente ao algoritmo *guess-and-check* (GC) encontrado na literatura. O método desenvolvido obteve soluções melhores em 21 instâncias, mesma solução em 927 casos e foi superado pelo GC em apenas 12 instâncias.

PALAVRAS CHAVE. Sequenciamento, Inventário, Busca Local. Meta-heurísticas, Otimização Combinatória.

ABSTRACT

Scheduling problems are commonly addressed in the literature and have several applications in industry. Among its numerous variants, this work focuses on the problem of minimizing the completion time of the last task in a sequence (*makespan*), considering inventory and release dates constraints. To solve this problem, a hybrid heuristic approach based on population-based search is proposed. The proposed method includes intensification and diversification procedures, where the former conducts a local search with an efficient move evaluation, while the latter comprise a strategy inspired by *ruin-and-recreate* strategy, along with the implementation of a population diversity control mechanism. Computational experiments on a set of 960 instances demonstrate the competitiveness of the proposed algorithm compared to the *guess-and-check* (GC) algorithm found in the literature. The HyPRR algorithm obtained better solutions in 21 instances, the same solution in 927 instances, and was outperformed by the GC in 12 instances.

KEYWORDS. Scheduling, Inventory, Local search. Metaheuristics, Combinatorial Optimization.

1. Introdução

Problemas de escalonamento de tarefas desempenham um papel importante no processo de tomada de decisão em muitos setores industriais e de produção, assim como na maioria dos ambientes que envolvem sistemas de processamento. Também são importantes em configurações de transporte, logística e distribuição de serviços [Pinedo, 2008]. Tipicamente, a teoria de escalonamento lida com a tarefa de processar um conjunto finito de trabalhos por um sistema com recursos limitados. Além disso, é uma área relevante da pesquisa operacional com aplicações práticas em diversos setores, como no transporte e na distribuição de recursos em cadeias produtivas, controle de fluxo em aeroportos, gestão de mão de obra em manufatura e no gerenciamento de tarefas em computação em nuvem [Bachtler et al., 2020].

Os recursos, tarefas e objetivos podem variar significativamente dependendo do contexto e das características de cada problema nas organizações. Os recursos podem ser máquinas fabris, pistas de pouso ou decolagem em aeroportos, trabalhadores da construção civil e unidades de processamento em computadores. As tarefas podem ser operações em uma linha de produção, demanda de pousos e decolagens em aeroportos, etapas de um projeto de engenharia ou ainda execuções necessárias em um software. Cada tarefa pode ter diferentes ordens de prioridade, data de liberação, tempo de processamento e data limite para finalização. Os recursos, por sua vez, são limitados e comumente abordados na literatura como parte de um inventário com capacidade de alocação limitada e não-negativa [Pinedo, 2008]. Davari et al. [2020] definiram as restrições de inventário como uma generalização das limitações de recursos não-renováveis. Tais recursos são consumidos a medida que as tarefas são executadas como, por exemplo, energia, materiais de consumo, equipamentos, trabalhadores ou recursos financeiros. Já os objetivos são abordados na literatura como minimização do número de tarefas finalizadas com atraso, soma de todos os atrasos ou tempo total de processamento de todas as tarefas. Além disso, para cada uma dessas variantes, as tarefas podem ter o mesmo ou diferentes pesos no objetivo final.

Nesse contexto, este trabalho aborda o problema de escalonamento de tarefas em uma única máquina com datas de liberação, restrição de inventário e com o objetivo de minimizar o tempo de conclusão da última tarefa (*makespan*). Como demonstrado por Davari et al. [2020], esse problema pertence à classe $\mathcal{NP}$ -difícil e, seguindo a notação de três campos de Graham et al. [1979], é denotado por $1|r_j, inv|C_{max}$. Para resolvê-lo, foi utilizada uma abordagem heurística híbrida [Teixeira et al., 2023] baseada em procedimentos de diversificação, através do operador *ruin-and-recreate* (R&R), e intensificação utilizando o algoritmo de descida em vizinhança variável com escolha aleatória [Subramanian et al., 2010], do inglês *randomized variable neighborhood descent* (RVND), para explorar as diferentes estruturas de vizinhança na busca local. Experimentos computacionais realizados em um conjunto de 960 instâncias, proposto por Davari et al. [2020], demonstram a competitividade da abordagem desenvolvida comparada ao algoritmo estado da arte para esse problema.

O restante do trabalho está organizado como segue. A Seção 2 apresenta os trabalhos relacionados ao problema abordado. A Seção 3 detalha a definição do problema. A Seção 4 descreve o algoritmo proposto. A Seção 5 apresenta os experimentos computacionais realizados e os resultados obtidos. A Seção 6 finaliza o trabalho com as considerações finais.

2. Trabalhos relacionados

O primeiro trabalho de sequenciamento envolvendo restrições de inventário encontrado na literatura com aplicação prática no gerenciamento de projetos foi o de Neumann

e Schwindt [2003]. Os autores estudaram um problema de sequenciamento de processos na indústria química e propuseram um algoritmo *branch-and-bound* (B&B). A partir dos experimentos reportados, o algoritmo foi capaz de resolver instâncias de até 100 atividades em menos de um minuto.

Bartels e Zimmermann [2009] abordaram o problema de escalonamento de testes em projetos de pesquisa e desenvolvimento (P&D) na indústria automotiva contemplando restrições de inventário, tempos mínimo e máximo entre duas tarefas subsequentes, bem como recursos renováveis e cumulativos. Os autores formularam e implementaram um modelo MILP para o problema em instâncias menores de 20 testes, 4 variantes de veículos e, motivados pelo trabalho de Neumann e Zimmermann [2000], desenvolveram uma heurística baseada em regras de prioridade para o escalonamento em instâncias maiores de 600 testes e 25 variantes de veículos.

Briskorn et al. [2010] provaram que o problema de escalonamento de tarefas com diferentes pesos em máquina única, restrições de inventário e minimização do tempo total de conclusão, denotado por $1|inv|\sum \omega_j C_j$, é fortemente $\mathcal{NP}$ -difícil. Eles consideraram como aplicação prática um terminal de cargas e descargas de caminhões. Briskorn et al. [2013] geraram 8640 instâncias de tamanhos $n \in \{5, 10, 15, 20, 25\}$ para o mesmo problema e propuseram dois algoritmos exatos baseados nas abordagens *branch-and-bound* e programação dinâmica (PD). Ao final, o B&B foi o único que encontrou mais soluções ótimas para instâncias de tamanho até 20. Entretanto, nenhum algoritmo foi capaz de resolver todas as instâncias a partir de 20 tarefas.

Nesse mesmo contexto, Briskorn e Leung [2013] desenvolveram um algoritmo B&B para o problema $1|inv|L_{max}$, também introduzido por Briskorn et al. [2010]. Com base nas fundamentações teóricas apresentadas e na regra de *earliest-due-date* (EDD), foi possível reduzir o espaço de busca e adotar uma estratégia mais eficiente de reordenação de tarefas em uma sequência. Posteriormente, foi proposta uma abordagem exata MILP e uma heurística baseada na metheurística *variable neighborhood search* (VNS), proposta por Mladenović e Hansen [1997], para o mesmo problema [Ranjbar e Saber, 2021].

Davari et al. [2020] apresentaram uma descrição formal do problema $1|r_j, inv|C_{max}$ e propuseram duas formulações de programação linear inteira mista: a primeira baseada em unidades discretas de tempo, enquanto a segunda com base na sequência de tarefas. Adicionalmente, eles propuseram um algoritmo exato B&B e uma abordagem baseada no algoritmo *guess-and-check* (GC) para resolver o problema de forma exaustiva. Apesar dos autores terem reportado a superioridade do GC no tempo e número de instâncias resolvidas em relação ao B&B, o GC conseguiu resolver apenas 80 instâncias do total de 96 a partir de 60 tarefas em 1000 segundos. Essa disparidade é ainda maior em instâncias de tamanho 100, em que o GC não conseguiu resolver 36 das 96 instâncias. Nesse contexto, há a oportunidade de desenvolver outras abordagens heurísticas mais eficientes em relação ao tempo e qualidade das soluções encontradas para instâncias do problema a medida que o número de tarefas aumenta ($n \geq 50$).

3. Descrição do problema

Seja $J = \{1, \dots, n\}$ um conjunto de tarefas a serem processadas, cada atividade $j \in J$ tem um tempo de processamento $p_j \in \mathbb{R}_{\geq 0}$, data de liberação $r_j \in \mathbb{R}_{\geq 0}$ e uma modificação de inventário $\delta_j \in \mathbb{R}$. O conjunto J pode ser particionado nos subconjuntos de tarefas J^- e J^+ que consomem e adicionam recursos no inventário, respectivamente. Dessa forma, $\delta_j \leq 0$ para $j \in J^-$ e $\delta_j \geq 0$ para $j \in J^+$. As tarefas precisam ser executadas em um ambiente de única máquina e cada tarefa deve ser processada uma única vez e somente

uma por vez. Além disso, só podem ser iniciadas após a sua respectiva data de liberação r_j e ao término da atividade antecessora $j - 1$. O inventário inicial e sua capacidade são definidos por I_0 e I_C , respectivamente, em que $I_0 \geq 0$ e $I_C > 0$.

Dada uma sequência $\pi = (\pi_1, \dots, \pi_n)$, ela é uma solução viável se $0 \leq I_0 + \sum_{k=1}^s \delta_{\pi_k} \leq I_C$ para cada $s = 1, \dots, n$. Seja Ω o conjunto de todas as soluções viáveis e $C_j(\pi)$ o tempo total de conclusão da tarefa j . Se todas as tarefas da sequência π iniciarem no menor tempo possível, o *makespan* da sequência π é denotado por $C_{max}(\pi)$ ou $C_{\pi_n}(\pi)$.

De forma resumida, o problema de sequenciamento com datas de liberação e restrição de inventário consiste em definir uma sequência de tarefas em J , cujo *makespan* é minimizado e as seguintes restrições devem ser satisfeitas:

- (i) cada atividade deve ser executada uma única vez;
- (ii) uma máquina só pode executar uma atividade por vez;
- (iii) o início do processamento de uma tarefa só pode se dar após a sua data de liberação;
- (iv) a modificação de recursos no inventário ao executar uma atividade não pode exceder a sua capacidade ou torná-lo negativo.

4. Algoritmo proposto

Esta seção apresenta o algoritmo utilizado para resolução do problema $1|r_j, inv|C_{max}$. O algoritmo populacional híbrido contém um operador de destruição e reconstrução R&R para diversificação das soluções. Simultaneamente, é apresentado um procedimento de busca local baseado na técnica RVND, composto por duas estruturas de vizinhança.

O Algoritmo 1 apresenta o pseudocódigo da abordagem utilizada e denominada como *hybrid population-based ruin-and-recreate* (HyPRR; foneticamente, “hyper”). O início do método consiste na inicialização de uma população P com μ indivíduos gerados aleatoriamente. Cada indivíduo representa uma permutação de tarefas. Cada indivíduo π é submetido à busca local (linhas 3 - 4) através do procedimento RVND.

Enquanto o número máximo de iterações consecutivas sem melhoria no *makespan* (It_{NI}) não é atingido, um indivíduo π é selecionado por torneio binário (linha 6) e submetido ao operador R&R [Ruiz e Stützle, 2007] para remoção e posterior reinserção de d tarefas (linha 7) utilizando o *makespan* como critério de avaliação da reinserção. Em seguida, o novo indivíduo π' é submetido à busca local para potencializar a eficiência do algoritmo na busca de soluções (linha 8) e o novo indivíduo π'' é inserido na população P (linha 9). Nessa etapa, se o tamanho da população atingir $\mu + \lambda$, o procedimento de gerenciamento da população descarta os λ indivíduos excedentes (linhas 10-11). O procedimento acaba quando o número de iterações consecutivas sem melhora (it) no *makespan* da melhor solução de P atinge It_{NI} (linhas 12-15). Finalmente o melhor indivíduo de P é retornado (linha 16).

4.1. *Ruin & recreate*

O operador R&R utilizado no HyPRR (linha 7) não tem como objetivo obter uma solução melhor, no entanto visa contribuir com a diversidade das soluções obtidas pelo algoritmo ao perturbar as soluções correntes para escapar de ótimos locais. O Algoritmo 2 apresenta o procedimento R&R. Inicialmente, d tarefas são removidas da sequência π e inseridas em uma lista temporária LC (linhas 2-3). Em seguida, enquanto LC não está

Algoritmo 1 Algoritmo HyPRR

```

1: procedimento HyPRR( $\mu, \lambda, \mu^{elite}, \mu^{close}, It_{NI}, d$ )
2:   Gere uma população  $P$  de  $\mu$  indivíduos aleatoriamente com busca local
3:   para cada  $\pi \in P$  faça
4:      $\pi \leftarrow RVND(\pi)$ 
5:   enquanto  $it < It_{NI}$  faça
6:     Selecione um indivíduo  $\pi$  de  $P$  por torneio binário
7:      $\pi' \leftarrow R\&R(\pi, d)$ 
8:      $\pi'' \leftarrow RVND(\pi')$ 
9:     Insira  $\pi''$  na população  $P$ 
10:    se  $|P| = \mu + \lambda$  então
11:       $P \leftarrow GerenciamentoPopulação(\lambda, \mu^{elite}, \mu^{close})$ ;
12:    se  $f(\pi'') < f(best(P))$  então
13:       $it \leftarrow 0$ ;
14:    senão
15:       $it \leftarrow it + 1$ ;
16:  retorne melhor solução da população
  
```

vazia, uma tarefa j é selecionada aleatoriamente de LC (linha 5), reinserida em π na posição que gere o menor *makespan* (linha 6) e removida da lista LC (linha 7).

Algoritmo 2 Algoritmo ruin-and-recreate

```

1: procedimento RUIN-AND-RECREATE( $\pi, d$ )
2:   Remova aleatoriamente  $d$  tarefas de  $\pi$ 
3:   Insira as  $d$  tarefas em uma lista LC
4:   enquanto  $LC \neq \emptyset$  faça
5:     Selecione aleatoriamente uma tarefa  $j \in LC$ 
6:      $\pi \leftarrow reinsira(j, \pi)$ 
7:      $LC \leftarrow LC \setminus \{j\}$ 
8:   retorne  $\pi$ 
  
```

4.2. Busca Local

A busca local é baseada na estratégia RVND [Subramanian et al., 2010]. A partir de uma lista NL inicializada com todas as estruturas de movimentos de vizinhança, uma delas é aleatoriamente selecionada e utilizada. Todos os tipos de movimentos são considerados na lista NL na busca de uma nova solução com menor *makespan* até que todas as estruturas sejam testadas e não resulte em nenhuma melhoria.

4.2.1. Estruturas de Vizinhança

São consideradas as seguintes estruturas de vizinhança no algoritmo proposto:

- *swap*: duas tarefas são permutadas em uma sequência.
- *reinsertion*: uma tarefa é removida da sequência e reinserida em outra posição.
- *l-block insertion*: um bloco de l tarefas adjacentes é removido e reinserido em outra posição para gerar uma nova sequência.

Seja s uma sequência de tamanho n que representa uma solução para o problema. O número de possíveis movimentos em cada uma das vizinhanças supracitadas é da ordem de $\mathcal{O}(n^2)$, enquanto a avaliação do *makespan* para cada movimento tem complexidade $\mathcal{O}(n)$. Dessa forma, em princípio, a complexidade total de enumeração e avaliação de todos os possíveis movimentos de cada vizinhança é $\mathcal{O}(n^3)$.

No entanto, este trabalho utiliza uma implementação de avaliação eficiente baseada na metodologia proposta por Vidal et al. [2012] que possibilita a avaliação do *makespan* e inventário em tempo amortizado $\mathcal{O}(1)$ utilizando uma estrutura de dados auxiliar e a concatenação de subsequências implementada por Bulhões et al. [2018] e utilizada por Morais et al. [2024]. Dessa forma, a complexidade total da busca local é reduzida para $\mathcal{O}(n^2)$.

Nessa estratégia de armazenamento de informações em uma estrutura de dados auxiliar, para cada subsequência $\pi = (\pi_1, \pi_2, \dots, \pi_n)$, são armazenadas as seguintes informações:

- $E(\pi)$ = menor tempo de início de processamento sem tempo ocioso;
- $D(\pi)$ = somatório dos tempos de processamento;
- $Q_s(\pi)$ = nível de inventário acumulado ao processar π ;
- $Q_{min}(\pi)$ e $Q_{max}(\pi)$ = níveis mínimo e máximo de inventário durante o processamento de π ;

Seja $\pi = (j)$ uma subsequência unitária composta pela atividade $j \in J$, o elemento correspondente na matriz de sequências tem as seguintes informações: $E(\pi) = r_j$, $D(\pi) = p_j$, $Q_s = \delta_j$, $Q_{min} = \min(0, \delta_j)$ e $Q_{max} = \max(0, \delta_j)$.

A partir de n subsequências unitárias de s , qualquer subsequência π , $|\pi| > 1$ pode ser definida a partir da concatenação de outras duas subsequências π^1 e π^2 . Essa concatenação é apresentada no Algoritmo 3. O *makespan* de qualquer subsequência pode ser definido como $C_{max} = E(\pi) + D(\pi) + M(\max(0, -Q_{min}(\pi)) + \max(0, Q_{max}(\pi) - I_c))$, em que M é fator de penalização da violação das restrições de inventário. Já o nível de inventário acumulado da concatenação de duas subsequências $Q_s(\pi^1 \oplus \pi^2)$ corresponde à soma dos níveis de inventário acumulado após o processamento de cada subsequência, denotados por $Q_s(\pi^1)$ e $Q_s(\pi^2)$.

Algoritmo 3 Concatenação $\pi^1 \oplus \pi^2$

```

1: procedimento CONCATENACAO-SUBSEQUENCIAS( $\pi^1, \pi^2$ )
2:    $Q_s(\pi^1 \oplus \pi^2) \leftarrow Q_s(\pi^1) + Q_s(\pi^2)$ ;
3:    $Q_{min}(\pi^1 \oplus \pi^2) \leftarrow \min(Q_{min}(\pi^1), Q_s(\pi^1) + Q_{min}(\pi^2))$ ;
4:    $Q_{max}(\pi^1 \oplus \pi^2) \leftarrow \max(Q_{max}(\pi^1), Q_s(\pi^1) + Q_{max}(\pi^2))$ ;
5:    $O \leftarrow 0$ ;
6:   se  $C(\pi^1) < E(\pi^2)$  então
7:      $O \leftarrow E(\pi^2) - C(\pi^1)$ ;
8:    $E(\pi^1 \oplus \pi^2) \leftarrow E(\pi^1) + O$ ;
9:    $D(\pi^1 \oplus \pi^2) \leftarrow D(\pi^1) + D(\pi^2)$ ;
10:  retorne  $\pi^1 \oplus \pi^2$ 
  
```

A concatenação de duas subsequências $\pi^1 \oplus \pi^2$ pode acarretar em duas possibilidades de tempo ocioso I entre a data de liberação $E(\pi^2)$ e o início da execução da

subsequência π^2 . O primeiro caso ocorre quando o tempo de conclusão da primeira subsequência $C(\pi^1)$ ocorre após a data de liberação da segunda subsequência $E(\pi^2)$. Assim sendo, o tempo ocioso de π^2 resulta apenas da conclusão de π^1 . Já no segundo caso ocorre o inverso ($E(\pi^2) > C(\pi^1)$) e o tempo ocioso é a diferença entre a conclusão da primeira $C(\pi^1)$ e a data de liberação da segunda subsequência $E(\pi^2)$, representado na Linha 7 do Algoritmo 3.

4.3. Gerenciamento da população

No Algoritmo 1, quando o tamanho da população P atinge $\mu + \lambda$, é realizado o procedimento de gerenciamento da população [Mecler et al., 2021a] para remoção de λ indivíduos. Esse procedimento utiliza a função de aptidão para ordenar os indivíduos com base nos valores do *makespan* e da contribuição para diversidade da população. O valor da função de aptidão é calculado a partir da Equação 1.

$$f(\pi) = f_{obj}(\pi) + (1 - \frac{\mu^{elite}}{|P|}) \cdot f_{div}(\pi) \quad (1)$$

O componente da função objetivo ($f_{obj}(\pi)$) é calculada ordenando os indivíduos em ordem crescente com base no *makespan* e dividindo a posição de cada indivíduo nessa lista ordenada pelo tamanho total da população. O componente de diversidade ($f_{div}(\pi)$) é calculado de maneira semelhante, mas os indivíduos são ordenados em ordem decrescente de acordo com sua contribuição para a diversidade. Essa contribuição é medida como a distância média para os vizinhos mais próximos de cada indivíduo, μ^{close} , com a distância sendo incrementada por uma unidade para cada vizinho diferente, seja à direita ou à esquerda. O parâmetro μ^{elite} especifica o número de indivíduos elite a serem preservados na população.

Essa função de aptidão também é utilizada no torneio binário que seleciona o indivíduo a ser submetido para o procedimento R&R. O indivíduo, entre os dois selecionados aleatoriamente, que tiver o maior valor $f(\pi)$ é o vencedor do torneio.

5. Resultados computacionais

O algoritmo proposto foi implementado utilizando a linguagem de programação C++ e os experimentos executados em um computador Intel®Core™ i7-12700 com 4.90 GHz, 16 GB de memória RAM e sistema operacional Ubuntu 20.04 LTS 64 bits. Foi utilizado um conjunto de 960 instâncias com tamanhos $n \in \{10, 20, 30, 40, 50, 60, 70, 80, 90, 100\}$ proposto por Davari et al. [2020].

Todos os valores de tempo de processamento (p_i), data de liberação (r_i) e modificação de inventário (δ_i) foram gerados seguindo uma distribuição uniforme inteira. Os valores de p_i ($1 \leq i \leq n$) seguiram o intervalo $[1, \alpha]$, em que $\alpha \in \{10, 100\}$. Já r_i , por sua vez, foram geradas no intervalo $[0, \tau \sum_{i \in J} p_i]$ com $\tau \in \{0, 5, 1, 1.5, 2\}$ e $|\delta_i|$ do intervalo $[1, 10]$. A capacidade de inventário I_C foi selecionada aleatoriamente entre $[10\eta, 20\eta]$, em que $\eta = \{1, 3, 5\}$.

Para avaliar o desempenho do algoritmo HyPRR, foram realizados testes de calibração em um subconjunto de 15 instâncias. No processo de escolha das instâncias para calibração, foram selecionadas as que obtiveram previamente os maiores *gaps* utilizando os parâmetros populacionais reportados por Mecler et al. [2021b]. Posteriormente, o algoritmo calibrado foi executado em todas as instâncias e comparado com o algoritmo *guess-and-check* [Davari et al., 2020]. Quanto à avaliação de qualidade das soluções, foi calculada a diferença percentual entre os resultados obtidos pelo algoritmo HyPRR e pelo GC. *Gaps* positivos indicam que o *makespan* obtido pelo HyPRR foi maior que o obtido pelo GC.

5.1. Estruturas de vizinhança e operador R&R

Uma das estruturas de vizinhança utilizada no algoritmo para busca local (*l-block insertion*) e o operador baseado no algoritmo *ruin-and-recreate* foram calibrados a fim de obter a combinação com os melhores resultados de *gap* médio, número de soluções inviáveis e tempo médio de execução.

De acordo com a Tabela 1, nota-se que a combinação $d = 5$ e $l = 5$ apresentou os melhores resultados, considerando o *gap* médio, número de inviabilidades e tempo médio. Os resultados para $d = 2$, $d = 3$ e $d = 4$ não são detalhados na Tabela 1 visto que apresentaram os maiores números de soluções inviáveis.

Tabela 1: *Gap* médio, número de soluções inviáveis e tempo médio em segundos por tamanho de bloco l e d tarefas removidas no R&R.

d	l	<i>Gap</i> (%)	Inviáveis	Tempo(s)	d	l	<i>Gap</i> (%)	Inviáveis	Tempo(s)
5	2	0,732	2	2,51	7	2	0,778	0	3,79
5	3	0,638	1	3,09	7	3	0,700	0	4,03
5	4	0,731	2	3,28	7	4	0,707	0	4,87
5	5	0,599	1	4,01	7	5	0,700	0	5,50
5	6	0,637	3	4,40	7	6	0,781	0	5,52
5	7	0,599	3	5,00	7	7	0,768	1	6,09
5	8	0,768	1	5,04	7	8	0,895	1	6,46
5	9	0,663	1	5,54	7	9	0,730	0	7,41
6	2	0,688	2	3,26	8	2	0,917	0	4,27
6	3	0,711	0	3,79	8	3	0,937	0	4,64
6	4	0,676	1	4,10	8	4	0,836	0	5,49
6	5	0,649	0	4,92	8	5	0,812	0	5,99
6	6	0,663	0	5,12	8	6	0,765	0	6,69
6	7	0,695	0	5,50	8	7	0,805	0	7,36
6	8	0,675	3	5,83	8	8	0,834	0	7,95
6	9	0,595	1	6,57	8	9	0,813	0	8,44

5.2. Parâmetros Populacionais

O gerenciamento da população seguiu os mesmos critérios utilizados por Teixeira et al. [2023], no entanto os seguintes parâmetros populacionais foram calibrados: tamanho da população inicial (μ), número máximo de novos indivíduos (λ), número de indivíduos elite (μ^{elite}) e o número de indivíduos mais próximos (μ^{close}). Os valores testados foram: $\mu = \{2, 4, 6, 8, 10, 20, 30\}$, $\lambda = \{1 \cdot \mu, 1,5 \cdot \mu, 2 \cdot \mu\}$, $\mu^{elite} = 0,5 \cdot \mu$ e $\mu^{close} = \lceil 0,2 \cdot \mu \rceil$.

A Tabela 2 apresenta o *gap* médio percentual, tempo médio em segundos e quantidade de soluções inviáveis obtidos por cada combinação dos parâmetros populacionais. O *gap* médio foi calculado comparando a média do *makespan* após 10 execuções em cada instância com o respectivo *makespan* obtido pelo GC.

Observa-se que o algoritmo com o menor tamanho de população inicial ($\mu = 2$) foi o único que não conseguiu encontrar soluções viáveis em todas as instâncias testadas. O conjunto de parâmetros com o melhor *makespan* foi a combinação $\mu = 8, \lambda = 12, \mu^{elite} = 4, \mu^{close} = 2$.

5.3. Impacto do R&R e do gerenciamento da população

A fim de avaliar a efetividade do algoritmo HyPRR, foi realizado um experimento em todo o conjunto de instâncias comparando o operador *ruin-and-recreate* com os operadores *Partially Mapped Crossover* (PMX) [Goldberg e Lingle, 2014], *Cycle Crossover* (CX) [Grefenstette, 2013] e *Order Crossover* (OX) [Davis et al., 1985] no algoritmo HGS.

Tabela 2: Variando os parâmetros populacionais ($d = 5$).

μ	λ	μ^{elite}	μ^{close}	$Gap_{médio}(\%)$	Tempo(s)	Inviáveis
2	2	1	1	1,820	4,88	2
2	3	1	1	1,750	4,70	2
2	4	1	1	1,921	5,10	2
4	4	2	2	0,549	0,75	0
4	6	2	1	0,544	0,76	0
4	8	2	2	0,531	0,79	0
6	6	3	2	0,484	0,64	0
6	9	3	2	0,482	0,62	0
6	12	3	2	0,493	0,64	0
8	8	4	2	0,559	0,76	0
8	12	4	2	0,465	0,60	0
8	16	4	2	0,609	0,82	0
10	10	5	2	0,587	0,75	0
10	15	5	2	0,557	0,77	0
10	20	5	2	0,536	0,69	0
20	20	10	4	0,755	0,93	0
20	30	10	4	0,791	0,95	0
20	40	10	4	0,789	1,04	0
30	30	15	6	0,896	1,08	0
30	45	15	6	1,005	1,14	0
30	60	15	6	0,987	1,18	0

Além disso, também foi avaliado o desempenho do HyPRR sem controle de diversidade e o algoritmo com indivíduo único.

De acordo com os resultados apresentados na Tabela 3, nota-se que as duas estratégias mais simples com indivíduo único e sem controle de diversidade são boas soluções para instâncias de tamanho até $n = 40$. A partir de instâncias maiores que 50 tarefas, o operador *ruin-and-recreate* [2] se destaca como o melhor operador frente aos demais avaliados.

Tabela 3: Gap médio e tempo médio de execução em segundos de diferentes operadores por diferentes tamanhos de instâncias.

Instâncias	HyPRR		HGS-OX		HGS-PMX		HGS-CX		S/ Cont. Div.		Ind. Único	
	Gap	Tempo	Gap	Tempo	Gap	Tempo	Gap	Tempo	Gap	Tempo	Gap	Tempo
$n = 10$	0,000	0,01	0,013	0,01	0,013	0,01	0,037	0,01	0,000	0,01	0,000	0,00
$n = 20$	0,000	0,02	0,013	0,03	0,029	0,02	0,064	0,02	0,000	0,03	0,000	0,02
$n = 30$	0,001	0,08	0,023	0,12	0,022	0,08	0,056	0,06	0,012	0,09	0,000	0,08
$n = 40$	0,025	0,22	0,080	0,33	0,033	0,21	0,165	0,17	0,024	0,22	0,047	0,20
$n = 50$	0,018	0,47	0,066	0,78	0,029	0,45	0,185	0,36	0,021	0,47	0,021	0,43
$n = 60$	0,033	0,98	0,136	1,67	0,045	0,99	0,242	0,73	0,039	0,97	0,047	0,92
$n = 70$	0,039	1,46	0,143	2,75	0,058	1,55	0,220	1,24	0,048	1,45	0,050	1,36
$n = 80$	0,020	2,27	0,069	4,46	0,030	2,40	0,178	1,89	0,023	2,22	0,026	2,09
$n = 90$	0,035	3,54	0,112	7,16	0,035	3,72	0,311	2,98	0,046	3,48	0,051	3,20
$n = 100$	0,055	5,26	0,180	10,61	0,069	5,80	0,332	4,62	0,076	5,13	0,083	4,47

5.4. Comparação com algoritmos da literatura

Os resultados obtidos pelo HyPRR no conjunto de 960 instâncias foram comparados ao algoritmo *guess-and-check* [Davari et al., 2020]. Ao todo, o HyPRR obteve soluções melhores em 21 instâncias, 927 empates e apenas 12 soluções com qualidade inferior ao algoritmo GC. Considerando o gap (%) médio, a partir de instâncias de tamanho $n = 50$ o HyPRR obteve vantagem marginal frente ao GC, como pode ser visto na Tabela 4.

Tabela 4: Gap médio (%), vitórias, empates e derrotas na qualidade de soluções do HyPRR comparado ao GC.

Instâncias	$Gap_{médio}(\%)$	Vitórias	Empates	Derrotas
$n = 10$	0,000	0	96	0
$n = 20$	0,000	0	96	0
$n = 30$	0,000	0	96	0
$n = 40$	0,001	1	94	1
$n = 50$	-0,011	1	95	0
$n = 60$	-0,003	2	93	1
$n = 70$	-0,003	4	91	1
$n = 80$	-0,005	3	92	1
$n = 90$	-0,016	5	88	3
$n = 100$	-0,004	5	86	5

Os tempos de execução dos dois algoritmos são apresentados na Tabela 5. Enquanto o GC foi executado em um computador com processador Intel®Core™ i7-3720QM de 2.6 GHz e 8 GB de RAM, o HyPRR foi testado em um computador Intel®Core™ i7-12700 com 4.90 GHz, 16GB de memória RAM. A diferença na capacidade de processamento entre as duas máquinas foi compensada no cálculo de tempo em segundos do GC, seguindo o comparativo em CPU Benchmarks (<https://www.cpubenchmark.net/compare/4669vs895/Intel-i7-12700-vs-Intel-i7-3720QM>) e dividindo o tempo reportado do GC por 2,096.

A Tabela 5 mostra os tempos de execução em segundos dos dois algoritmos somente nas instâncias resolvidas pelo GC antes do critério de parada de 1000 segundos do GC a fim de dimensionar corretamente a diferença de tempo entre os dois algoritmos. Nota-se que a partir de $n \geq 30$ o HyPRR torna-se significativamente mais rápido em relação ao GC.

Tabela 5: Tempo médio em segundos de execução por grupo de tamanho do HyPRR e GC. (Somente instâncias resolvidas pelo GC antes do critério de parada)

Instâncias	Tempo médio HyPRR (s)	Tempo médio GC (s)
$n = 10$	0,00	0,00
$n = 20$	0,02	0,04
$n = 30$	0,08	6,01
$n = 40$	0,20	8,22
$n = 50$	0,42	13,29
$n = 60$	0,83	20,77
$n = 70$	1,22	22,26
$n = 80$	1,91	13,08
$n = 90$	2,78	20,46
$n = 100$	4,31	21,39

6. Conclusão

Este trabalho aborda o problema de sequenciamento em única máquina com tempo de liberação e restrições de inventário ($1|r_j, inv|C_{max}$), propondo uma abordagem heurística híbrida denominada *hybrid population-based ruin-and-recreate* (HyPRR) baseado no algoritmo *hybrid genetic search* (HGS) combinado ao operador R&R e RVND na busca de melhores soluções. Foi utilizado um grupo de 960 instâncias propostas por Davari et al. [2020].

Foram apresentados os resultados de calibração dos parâmetros populacionais, tamanho dos blocos na reinserção e operador R&R, bem como um comparativo do R&R com diferentes operadores de *crossover* vastamente abordados na literatura. Ao fim, o resultado obtido pelo algoritmo HyPRR superou o *guess-and-check* na qualidade das soluções em instâncias maiores ($n \geq 50$), na quantidade de instâncias em que o melhor *makespan* conhecido foi encontrado e no tempo de execução.

Para trabalhos futuros, sugere-se a avaliação do algoritmo apresentado em instâncias maiores ($n \geq 100$), desenvolvimento de métodos que possam melhorar a qualidade das soluções encontradas e do tempo de execução. Outra oportunidade futura é a avaliação de melhores estratégias na utilização do operador R&R.

Referências

- Bachtler, O., Krumke, S. O., e Le, H. M. (2020). Robust single machine makespan scheduling with release date uncertainty. *Operations Research Letters*, 48(6):816–819.
- Bartels, J.-H. e Zimmermann, J. (2009). Scheduling tests in automotive r&d projects. *European Journal of Operational Research*, 193(3):805–819.
- Briskorn, D. e Leung, J. Y.-T. (2013). Minimizing maximum lateness of jobs in inventory constrained scheduling. *Journal of the Operational Research Society*, 64(12):1851–1864.
- Briskorn, D., Choi, B.-C., Lee, K., Leung, J., e Pinedo, M. (2010). Complexity of single machine scheduling subject to nonnegative inventory constraints. *European Journal of Operational Research*, 207(2):605–619.
- Briskorn, D., Jaehn, F., e Pesch, E. (2013). Exact algorithms for inventory constrained scheduling on a single machine. *Journal of scheduling*, 16:105–115.
- Bulhões, T., Subramanian, A., Erdoğan, G., e Laporte, G. (2018). The static bike relocation problem with multiple vehicles and visits. *European Journal of Operational Research*, 264(2):508–523.
- Davari, M., Ranjbar, M., De Causmaecker, P., e Leus, R. (2020). Minimizing makespan on a single machine with release dates and inventory constraints. *European Journal of Operational Research*, 286(1):115–128.
- Davis, L. et al. (1985). Applying adaptive algorithms to epistatic domains. In *IJCAI*, volume 85, p. 162–164. Citeseer.
- Goldberg, D. E. e Lingle, R. (2014). Alleles, loci, and the traveling salesman problem. In *Proceedings of the first international conference on genetic algorithms and their applications*, p. 154–159. Psychology Press.
- Graham, R. L., Lawler, E. L., Lenstra, J. K., e Kan, A. R. (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. In *Annals of discrete mathematics*, volume 5, p. 287–326. Elsevier.
- Grefenstette, J. J. (2013). *Genetic algorithms and their applications: proceedings of the second international conference on genetic algorithms*. Psychology Press.

- Mecler, J., Subramanian, A., e Vidal, T. (2021a). A simple and effective hybrid genetic search for the job sequencing and tool switching problem. *Computers & Operations Research*, 127:105153.
- Mecler, J., Subramanian, A., e Vidal, T. (2021b). A simple and effective hybrid genetic search for the job sequencing and tool switching problem. *Computers & Operations Research*, 127:105153.
- Mladenović, N. e Hansen, P. (1997). Variable neighborhood search. *Computers & Operations Research*, 24(11):1097–1100.
- Morais, R., Bulhões, T., e Subramanian, A. (2024). Exact and heuristic algorithms for minimizing the makespan on a single machine scheduling problem with sequence-dependent setup times and release dates. *European Journal of Operational Research*, 315(2):442–453.
- Neumann, K. e Zimmermann, J. (2000). Procedures for resource leveling and net present value problems in project scheduling with general temporal and resource constraints. *European Journal of Operational Research*, 127(2):425–443.
- Neumann, K. e Schwindt, C. (2003). Project scheduling with inventory constraints. *Mathematical Methods of Operations Research*, 56:513–533.
- Pinedo, M. L. (2008). *Scheduling: Theory, Algorithms, and Systems*. Springer Publishing Company, Incorporated, 3rd edition.
- Ranjbar, M. e Saber, R. G. (2021). A variable neighborhood search algorithm for transshipment scheduling of multi products at a single station. *Applied Soft Computing*, 98: 106736.
- Ruiz, R. e Stützle, T. (2007). A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, 177(3):2033–2049.
- Subramanian, A., Drummond, L. M., Bentes, C., Ochi, L. S., e Farias, R. (2010). A parallel heuristic for the vehicle routing problem with simultaneous pickup and delivery. *Computers & Operations Research*, 37(11):1899–1911.
- Teixeira, E., Subramanian, A., e Kramer, H. H. (2023). Um algoritmo populacional para o problema de flow shop com bloqueio e minimização do makespan. In *Anais do VIII Encontro de Teoria da Computação*, p. 180–184. SBC.
- Vidal, T., Crainic, T. G., Gendreau, M., Lahrichi, N., e Rei, W. (2012). A hybrid genetic algorithm for multidepot and periodic vehicle routing problems. *Operations Research*, 60 (3):611–624.