UNIVERSIDADE FEDERAL DA PARAÍBA
CENTRO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

FREDERICO DE SOUZA GUERRA

UMA ABORDAGEM HEURÍSTICA HÍBRIDA PARA O PROBLEMA DE SEQUENCIAMENTO DE TAREFAS COM DATAS DE LIBERAÇÃO E RESTRIÇÕES DE INVENTÁRIO

João Pessoa
2024

FREDERICO DE SOUZA GUERRA

UMA ABORDAGEM HEURÍSTICA HÍBRIDA PARA O PROBLEMA DE SEQUENCIAMENTO DE TAREFAS COM DATAS DE LIBERAÇÃO E RESTRIÇÕES DE INVENTÁRIO

Dissertação submetida ao Programa de Pós-Graduação em Informática do Centro de Informática da Universidade Federal da Paraíba, como requisito parcial para obtenção do Grau de Mestre em Informática.

Orientador: Prof. Dr. Anand Subramanian

Co-orientador: Prof. Dr. Bruno Petrato Bruck

João Pessoa

2024

Catalogação na publicação
Seção de Catalogação e Classificação

G934a Guerra, Frederico de Souza.
Uma abordagem heurística híbrida para o problema de sequenciamento de tarefas com datas de liberação e restrições de inventário / Frederico de Souza Guerra. - João Pessoa, 2024.
103 f. : il.

Orientação: Anand Subramanian.
Coorientação: Bruno Petrato Bruck.
Dissertação (Mestrado) - UFPB/CI.

1. Informática. 2. Sequenciamento de tarefas. 3. Inventário. 4. Meta-heurísticas. 5. Busca populacional. 6. Busca local. I. Subramanian, Anand. II. Bruck, Bruno Petrato. III. Título.

UFPB/BC CDU 004(043)

Elaborado por Larissa Silva Oliveira de Mesquita - CRB-15/746

UNIVERSIDADE FEDERAL DA PARAÍBA
CENTRO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

Ata da Sessão Pública de Defesa de Dissertação de Mestrado de Frederico de Souza Guerra, candidato ao título de Mestre em Informática na área de Sistemas de Computação, realizada em 30 de agosto de 2024.

Aos trinta dias do mês de agosto do ano de dois mil e vinte e quatro, às treze horas, no Centro de Informática da Universidade Federal da Paraíba, reuniram-se os membros da Banca Examinadora constituída para julgar o Trabalho Final do discente Frederico de Souza Guerra, vinculado a esta Universidade sob a matrícula nº 20221004457, candidato ao grau de Mestre em Informática, na área de *"Sistemas de Computação"*, na linha de pesquisa *"Computação Distribuída"*, do Programa de Pós-Graduação em Informática. A comissão examinadora foi composta pelos professores: Anand Subramanian, Orientador e Presidente da banca; Bruno Petrato Bruck, Examinador Interno; Luciano Carlos Azevedo da Costa, Examinador Externo ao Programa; Yuri Laio Teixeira Veras Silva, Externo à Instituição. Dando início aos trabalhos, o Presidente da Banca cumprimentou os presentes, comunicou a finalidade da reunião e passou a palavra ao candidato para que ele fizesse a exposição oral do trabalho de dissertação intitulado "**Uma Abordagem Heurística Híbrida para o Problema de Sequenciamento de Tarefas com Datas de Liberação e Restrições de Inventário**". Concluída a exposição, o candidato foi arguido pela Banca Examinadora que emitiu o seguinte parecer: "***aprovado***". Do ocorrido, eu, Gilberto Farias de Sousa Filho, coordenador do Programa de Pós-Graduação em Informática, lavrei a presente ata que vai assinada por mim e pelos membros da Banca Examinadora. João Pessoa, 30 de agosto de 2024.

gov.br
Documento assinado digitalmente
GILBERTO FARIAS DE SOUSA FILHO
Data: 21/11/2024 10:35:47-0300
Verifique em https://validar.iti.gov.br

Gilberto Farias de Sousa Filho
Coordenador do Programa de Pós-Graduação em Informática

gov.br
Documento assinado digitalmente
ANAND SUBRAMANIAN
Data: 10/11/2024 17:28:45-0300
Verifique em https://validar.iti.gov.br

Prof. Dr. Anand Subramanian
Orientador (PPGI-UFPB)

gov.br
Documento assinado digitalmente
BRUNO PETRATO BRUCK
Data: 05/09/2024 14:09:39-0300
Verifique em https://validar.iti.gov.br

Prof. Dr. Bruno Petrato Bruck
Examinador Interno (PPGI-UFPB)

gov.br
Documento assinado digitalmente
LUCIANO CARLOS AZEVEDO DA COSTA
Data: 05/09/2024 08:43:13-0300
Verifique em https://validar.iti.gov.br

Prof. Dr. Luciano Carlos Azevedo da Costa
Examinador Externo ao Programa (UFPB)

gov.br
Documento assinado digitalmente
YURI LAIO TEIXEIRA VERAS SILVA
Data: 05/09/2024 08:37:42-0300
Verifique em https://validar.iti.gov.br

Prof. Dr. Yuri Laio Teixeira Veras Silva
Examinador Externo à Instituição (UFCG)

RESUMO

Problemas de sequenciamento de tarefas são comumente 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 resolver esse problema, é proposta uma abordagem heurística híbrida simples e eficaz baseada em *ruin-and-recreate* (HyPRR). Além do procedimento *ruin-and-recreate*, o algoritmo incorpora um esquema de diversificação para o gerenciamento da população e uma fase de intensificação por meio de busca local. Para melhorar o desempenho, é introduzida uma estratégia de avaliação de movimentos capaz de calcular eficientemente tanto o *makespan* quanto avaliar a inviabilidade em relação às restrições de inventário em tempo constante amortizado. Experimentos computacionais em um conjunto de 960 instâncias destacam a competitividade do algoritmo proposto quando comparado ao *guess-and-check* (GC) encontrado na literatura. O método desenvolvido encontrou a solução ótima em 99% das instâncias com ótimos conhecidos, e encontrou soluções melhores em 21 instâncias.

Palavras-chaves: Sequenciamento. Inventário. Meta-heurísticas; Busca populacional; Busca local.

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 makespan on a single-machine scheduling problem with release dates and inventory constraints. To solve this problem, a simple yet effective hybrid population-based ruin-and-recreate (HyPRR) heuristic is proposed. In addition to a simple ruin-and-recreate procedure, the algorithm incorporates a diversification scheme for population management and an intensification phase through local search. To enhance performance, we introduced a move evaluation strategy capable of efficiently computing both the makespan and assessing infeasibility with respect to inventory constraints in amortized constant time. Computational experiments on a set of 960 instances highlight the competitiveness of the proposed algorithm when compared to the guess-and-check (GC) found in the literature. The developed method found the optimal solution in 99% of the instances with known optima, and produced new best solutions for 21 instances.

Keywords: Scheduling; Inventory; Metaheuristics; Population search; Local search;

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

Lista de Abreviaturas e Siglas

It_{NI}	critério de parada
AG	Algoritmo Genético
ATCS	*apparent tardiness cost with setup*
B&B	*branch-and-bound*
CX	*cycle crossover*
GBF	*greedy-based fixing algorithm*
GC	*guess-and-check*
HGS	*hybrid genetic search*
HyPRR	*hybrid population-based ruin-and-recreate*
MILP	*mixed integer linear programming*
MIP	*mixed integer programming*
OC	Otimização Combinatória
OX	*order crossover*
P&D	Pesquisa e Desenvolvimento
PD	Programação Dinâmica
PMX	*partially mapped crossover*

PO	Pesquisa Operacional
PSDLI	Problema de Sequenciamento de tarefas com Datas de Liberação e Inventário
PSO	*particle swarm optimization*
R&R	*ruin-and-recreate*
RPC	*relative percent change*
RVND	*randomized variable neighborhood descent*
SA	*simulated annealing*
TA	*threshold-accepting*
TS	*tabu search*
TSP	*travelling salesman problem*
VNS	*variable neighborhood search*

Capítulo 1

Introdução

1.1 Definição do tema

O problema de escalonamento, do inglês *scheduling*, desempenha um papel importante no processo de tomada de decisão em muitos setores industriais e de produção. Além disso, é um problema relevante de Otimização Combinatória (OC) nas áreas de Ciência da Computação e Pesquisa Operacional (PO) com aplicações práticas em diversos setores, como na indústria automotiva (Bartels e Zimmermann, 2009), descomissionamento de usinas nucleares (Bartels et al., 2011), controle de fluxo em terminais logísticos (Briskorn et al., 2010) e aeroportos (Han et al., 2020), gerenciamento de projetos (Neumann e Schwindt, 2003), ou ainda na orquestração de tarefas em computação em nuvem (Houssein et al., 2021).

De forma geral, dado um conjunto de tarefas a serem processadas em um ambiente de produção, o problema de *scheduling* consiste em obter uma sequência que representa a ordem em que cada tarefa deve ser processada, de modo a otimizar um objetivo específico inerente ao problema. O ambiente de produção pode conter uma única ou múltiplas máquinas. No primeiro, apenas uma máquina é utilizada no processamento das tarefas, enquanto no último, múltiplas máquinas podem estar em série ou em paralelo (Pinedo, 2022).

O problema de *scheduling* em ambiente de múltiplas máquinas também é denomi-

nado como escalonamento de atividades. Já o ambiente de máquina única é um caso particular do problema de escalonamento e denominado como sequenciamento. Na prática, esses ambientes de máquinas podem ser unidades de processamento em computadores (Houssein et al., 2021), terminais de embarque e desembarque em aeroportos (Pinol e Beasley, 2006) ou unidades de cargas e descargas em terminais logísticos (Fonseca et al., 2019). As tarefas, por sua vez, podem ser processos a serem ordenados no gerenciamento de projetos (Neumann e Schwindt, 2003; Bartels et al., 2011), caminhões de cargas em terminais logísticos (Fonseca et al., 2019; Briskorn et al., 2010) ou ainda aeronaves no problema de escalonamento em aeroportos (Pinol e Beasley, 2006).

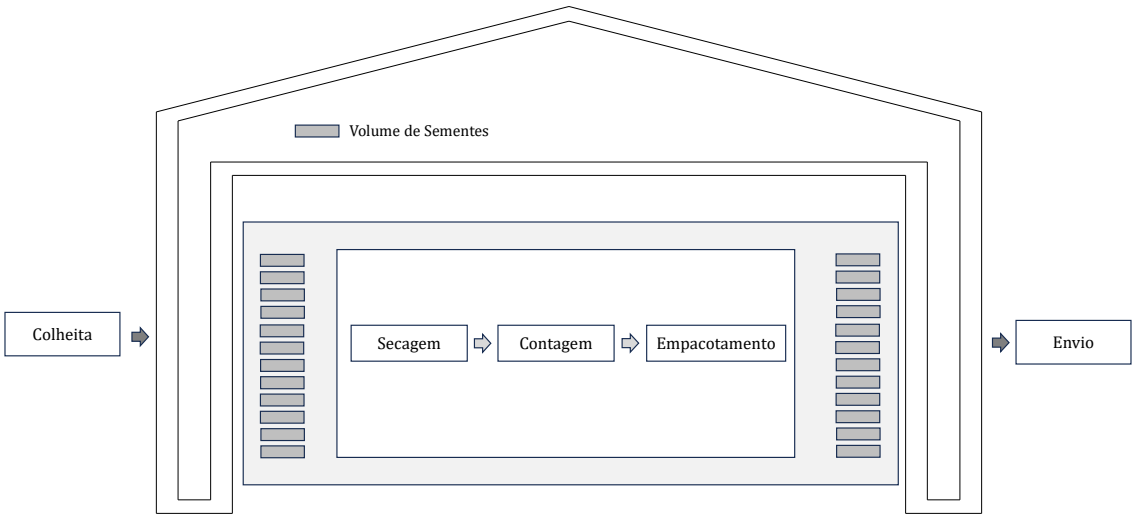
Para se obter esse sequenciamento de tarefas, pode-se considerar ainda as características ou restrições inerentes ao problema como, por exemplo, as datas de liberação para o início de cada tarefa (Avdeenko et al., 2017; Bazgosha et al., 2017), tempos de preparação de máquina entre o processamento de tarefas subsequentes (Cappadonna et al., 2013; Torabi et al., 2013), recursos limitados que são transformados durante o processamento, ou ainda inventários usados para armazenar recursos (Davari et al., 2020).

Na indústria, os inventários podem ser terminais de cargas e descargas em galpões logísticos, como abordado por Yu e Egbelu (2008), onde as unidades de distribuição são recebidas e despachadas por caminhões. Outro exemplo prático de inventário, exemplificado na Figura 1.1, pode ser encontrado em um laboratório de processamento de sementes em uma grande empresa agrícola.

Nesse exemplo, o laboratório corresponde ao ambiente de máquina única em que as tarefas são os pacotes de sementes oriundas do processo de colheita para serem processadas e armazenadas em câmaras frias antes do envio para outras localidades de plantio. Os volumes de sementes têm diferentes datas de liberação, tempos de preparação específicos e datas de entrega. Apesar da necessidade de maximização da quantidade de sementes processadas, a capacidade das câmaras frias, ou inventário, não pode ser ultrapassada em nenhum momento durante essa operação. Além disso, os recursos de armazenamento necessários no inventário estão relacionados aos diferentes volumes de sementes em cada pacote.

Figura 1.1: Exemplo de fluxo de processamento de sementes em uma indústria agrícola com restrição de inventário

.



Fonte: Elaboração Própria.

Segundo Pinedo (2022), os recursos são parte de um inventário com capacidade de alocação limitada e não-negativa, ou seja, um ambiente de armazenamento cujo volume não pode ser negativo ou ultrapassar a sua capacidade à medida que as tarefas são processadas e o nível de recursos no inventário varia durante o processamento. Davari et al. (2020) definiram as restrições de inventário como uma generalização das limitações de recursos não-renováveis. Esses recursos, de forma prática, podem ser unidades de armazenamento em terminais de distribuição (Briskorn et al., 2010; Ghorbanzadeh et al., 2019) ou passageiros no escalonamento de aeronaves (Pinol e Beasley, 2006).

Outro fator importante para se obter o sequenciamento de tarefas em um problema de *scheduling*, é o objetivo específico de otimização. Na literatura, há diversos objetivos abordados, dentre eles a minimização do número de tarefas finalizadas com atraso (Torabi et al., 2013), a soma de todos os atrasos (Briskorn et al., 2013; Briskorn e Pesch, 2013) e o tempo total de processamento de todas as tarefas (Ghorbanzadeh

et al., 2019; Davari et al., 2020). Além disso, para cada uma dessas variantes, as tarefas podem ter o mesmo ou diferentes pesos no objetivo final (Briskorn e Pesch, 2013; Torabi et al., 2013; Ranjbar e Saber, 2021).

Nesse sentido, este trabalho aborda o problema de minimização do tempo total de processamento (*makespan*) no sequenciamento de tarefas em um ambiente de máquina única (sequenciamento) com restrições de datas de liberação e inventário. Em outras palavras, o problema consiste em obter uma sequência de processamento das atividades de modo a minimizar o tempo de conclusão da última tarefa, em que cada tarefa só pode iniciar o processamento após a respectiva data de liberação e ao processar uma tarefa são consumidos ou adicionados recursos em um inventário de capacidade limitada.

No entanto, a obtenção de soluções ótimas para problemas de sequenciamento de tarefas com restrições de data de liberação e inventário revela-se altamente desafiadora à medida que o número de tarefas aumenta, devido às inúmeras possibilidades de ordenação (Davari et al., 2020). A resolução desse tipo de problema de OC pode ser realizada por meio de algoritmos exatos, que garantem a otimalidade para qualquer instância do problema. Contudo, esses algoritmos podem demandar um tempo computacional inviável para instâncias maiores (Briskorn et al., 2010).

Aliado a isso, o rápido avanço tecnológico nas indústrias torna os problemas práticos de *scheduling* cada vez maiores e mais complexos de se resolver, inviabilizando o uso de algoritmos exatos para obtenção de soluções ótimas em tempo hábil. Como alternativa, podem ser utilizadas abordagens heurísticas que, apesar de não garantirem a otimalidade, são capazes de explorar o espaço de busca de forma inteligente para obtenção de soluções sub-ótimas em um tempo aceitável.

Em geral, as heurísticas são definidas conforme as características específicas e especializadas em cada problema. As abordagens heurísticas podem ser construtivas ou de busca local. Na primeira, os algoritmos heurísticos visam construir uma solução viável para o problema a partir de uma sequência de regras. As heurísticas de busca local, por outro lado, partem de uma solução já criada e visam melhorar a solução corrente a partir de movimentos para explorar o espaço de vizinhanças.

Para problemas reconhecidamente difíceis de se resolver, como o problema de minimização do *makespan* no sequenciamento de tarefas com datas de liberação e res-

trições de inventário, as heurísticas não são suficientes para encontrar soluções tão ótimas ou viáveis quanto o possível. Nessas situações, é necessária a utilização de meta-heurísticas, que são procedimentos iterativos de alto nível que utilizam heurísticas subordinadas para explorar o espaço de soluções de forma mais robusta (Blum e Roli, 2003). As meta-heurísticas podem ainda ser classificadas de acordo com o seu princípio de funcionamento: busca populacional, local ou híbrida (Talbi, 2002).

As meta-heurísticas híbridas representam uma linha de pesquisa relevante na atualidade (Blum e Roli, 2008) e o seu uso é apropriado para problemas complexos que requerem mecanismos de intensificação para busca local de boas soluções na vizinhança do espaço amostral, bem como mecanismos de diversificação que auxiliem o algoritmo a escapar de regiões já exploradas para busca de boas soluções em um caráter global.

Nesse contexto, este trabalho propõe uma abordagem híbrida com procedimentos de intensificação e diversificação, baseados em busca local e populacional, para minimização do *makespan* no PSDLI, utilizando instâncias da literatura e comparando os resultados obtidos com o trabalho encontrado na literatura.

1.2 Justificativa

Com o avanço da tecnologia e o aumento significativo da digitalização nas indústrias nos últimos anos, cada vez mais torna-se necessário o desenvolvimento de ferramentas que auxiliem na tomada de decisão de forma assertiva nas empresas. Aliado a isso, em diversos setores industriais, há a necessidade de sistemas de produção mais ágeis e com utilização eficiente dos recursos disponíveis. Nesse âmbito, o processo *scheduling* desempenha um papel fundamental ao determinar sequências de tarefas que reduzam a ociosidade e maximizem a utilização eficiente dos recursos.

No entanto, esse processo de sequenciamento pode ser de difícil resolução sob uma perspectiva técnica do próprio problema, bem como do ponto de vista da implementação. A complexidade desses fatores aliam-se ainda a outros desafios, quando o problema de sequenciamento está inserido em um ambiente industrial, devido à interdependência com outros processos, reduções repentinas na disponibilidade de recursos ou ainda na capacidade de inventário para armazenamento desses recursos. Nesse cenário, as to-

madas de decisões podem ser extremamente ineficientes, se não forem suportadas por uma abordagem matemática apropriada.

De forma prática, o sequenciamento pode ser observado em diferentes problemas na indústria. Briskorn et al. (2010) abordou o problema de escalonamento em uma única máquina com restrições de inventário a partir de um problema de agendamento de caminhões em um terminal de cargas. Cada caminhão entrega ou recolhe um item específico que, naturalmente, não pode ser inferior à quantidade a ser recolhida. O ambiente de máquina única nesse problema corresponde ao único portão de cargas, enquanto os caminhões representam tarefas cujo tempo de liberação corresponde à sua disponibilidade para entrada.

Avdeenko et al. (2017) modelaram um planejamento de atividades de perfuração de poços para mineração na indústria de petróleo e gás como um sistema de *scheduling*. A perfuração de poços necessita de uma sequência otimizada para a sua construção e de outras instalações, considerando a disponibilidade dos recursos necessários, alto custo das operações e com o objetivo de minimizar o tempo total de execução dos poços na área de petróleo e gás. Outros problemas na indústria também podem ser modelados como um problema de *scheduling*, como na indústria automotiva (Bartels e Zimmermann, 2009), no descomissionamento de usinas nucleares (Bartels et al., 2011) ou ainda no agendamento de operações em centros de distribuição com restrições de inventário (Davari et al., 2020).

Além da vasta aplicabilidade prática demonstrada, o problema de minimização do *makespan* no sequenciamento com datas de liberação e restrição de inventário é comprovadamente $\mathcal{NP}$-difícil (Davari et al., 2020), ou seja, não é possível garantir que a melhor solução pode ser encontrada em tempo polinomial. Davari et al. (2020) tentaram resolver o problema com abordagens exatas através dos algoritmos *branch-and-bound* (B&B) e *guess-and-check* (GC), no entanto, o algoritmo GC, que obteve os melhores resultados, só foi capaz de resolver todas as instâncias com até 30 tarefas de forma ótima. Portanto, faz-se necessária a utilização de abordagens heurísticas a fim de encontrar soluções eficientes em instâncias de tamanhos realistas para o problema.

1.3 Objetivos

1.3.1 Objetivo geral

Diante do exposto, o presente trabalho visa propor uma abordagem heurística híbrida (Teixeira et al., 2023) para minimização do *makespan* no problema PSDLI 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 busca local.

1.3.2 Objetivos específicos

Os objetivos específicos do presente trabalho estão listados a seguir.

- Realizar uma revisão da literatura dos modelos e abordagens para resolução do problema.
- Implementar e avaliar a efetividade de diferentes operadores de busca local na melhoria das soluções obtidas.
- Implementar e avaliar a eficácia do operador R&R em comparação com outros operadores de mutação, considerando a qualidade das soluções.
- Resolver o problema de minimização do *makespan* no sequenciamento de tarefas com datas de liberação e restrição de inventário.
- Comparar os resultados obtidos com os de outros algoritmos que constituem o estado da arte para o problema considerado.

1.4 Estrutura do trabalho

O trabalho está organizado da seguinte maneira: o Capítulo 2 pontua algumas notações referentes ao problema de sequenciamento de tarefas e à revisão da literatura relacionada. No Capítulo 3 é apresentada a abordagem proposta para resolução do

problema. Em seguida, no Capítulo 4, são apresentados os resultados obtidos dos experimentos computacionais realizados. Por fim, no Capítulo 5, são elencadas as considerações finais do trabalho.

Capítulo 2

Fundamentação Teórica e Revisão da Literatura

2.1 Notações em *scheduling*

O problema de *scheduling* é um problema clássico na área de PO com diversos trabalhos na literatura. Essa diversidade de trabalhos, com diferentes particularidades relacionadas a diferentes máquinas, restrições e objetivos, acarreta muitas variantes do problema. Nesse contexto, Graham et al. (1979) fizeram uma contribuição importante propondo a notação de três campos ($\alpha|\beta|\gamma$) para representar de forma simplificada cada variante. A seguir, são demonstradas algumas das principais configurações para cada campo e apresentadas anteriormente por Pinedo (2022) e Graham et al. (1979).

O campo α representa a configuração de máquinas em que as tarefas são processadas e contém somente um parâmetro de entrada. Em seguida, são apresentadas as definições dos principais termos encontrados na literatura:

- ***Single Machine*** (1): todas as tarefas devem ser processadas em apenas uma máquina.
- ***Identical machines in parallel*** (P_m): há m máquinas idênticas em paralelo para processar as n tarefas, cujos tempos de processamento são os mesmos em

todas as máquinas.

- ***Machines in parallel with different speeds*** (Q_m): existem m máquinas em paralelo com diferentes velocidades de processamento. A velocidade de uma máquina i é denotada como v_i e o tempo de processamento p_{ij} que uma tarefa j leva na máquina i é igual a p_j/v_i. O ambiente de máquinas P_m, definido anteriormente, é um caso particular do ambiente Q_m em que todas as máquinas têm a mesma velocidade de processamento para cada tarefa, por exemplo, $v_i = 1$ para toda máquina i e $p_{ij} = p_j$.

- ***Unrelated machines in parallel*** (R_m): esse ambiente é uma generalização da configuração Q_m em que a velocidade de processamento da máquina i varia para cada tarefa j, ou seja, existem m máquinas em paralelo e uma máquina i pode processar a tarefa j a uma velocidade v_{ij}. O tempo de processamento p_{ij} da tarefa j na máquina i é igual a p_j/v_{ij}.

- ***Flow shop*** (F_m): nesse ambiente existem m máquinas em série, cada tarefa deve ser processada em cada uma das m máquinas e todas as tarefas devem seguir a mesma ordem de processamento. Além disso, após a finalização de processamento em uma máquina, a tarefa deve entrar em uma fila de espera para ser processada na máquina seguinte.

- ***Job shop*** (J_m): nessa configuração cada tarefa tem a sua própria sequência de máquinas para seguir durante o processamento. Há a versão de *job shop* em que cada tarefa deve ser processada uma única vez em cada máquina e o caso em que as tarefas podem ser processadas mais de uma vez em uma mesma máquina. Nesse último caso há uma denotação *rcrc* no campo β especificando que há recirculação nas tarefas.

- ***Open shop*** (O_m): existem m máquinas e cada tarefa deve ser processada uma única vez em cada máquina. Porém, em alguns casos o tempo de processamento da tarefa j na máquina i pode ser zero. Não há restrições quanto à ordem de processamento e o escalonamento pode ser diferente para cada tarefa.

O campo β representa as restrições inerentes às características do escalonamento e pode conter mais de um parâmetro de entrada. Algumas das principais notações encontradas na literatura são:

- ***Release dates*** (r_j): cada tarefa j só pode ser iniciada após a sua respectiva data de liberação r_j. Se essa notação r_j não aparecer no campo β, indica que as tarefas podem iniciar o seu processamento a qualquer momento. Em alguns problemas pode haver a especificação da data de finalização d_j de cada tarefa, no entanto já está inerente à função objetivo denotada no campo γ e, por isso, não aparece no campo de restrições β.

- ***Preemptions*** ($prmp$): essa restrição implica que o processamento de uma tarefa, após ser iniciado em uma máquina, pode ser interrompido para dar lugar a outra tarefa. O tempo já processado da tarefa interrompida não é perdido, ou seja, é compensado do tempo total e a tarefa seguirá o processamento posteriormente em qualquer máquina com o tempo restante no momento da interrupção.

- ***Inventory level*** (inv): o processamento de uma tarefa j consome ou adiciona uma quantidade de recursos δ_j em um inventário inv de capacidade limitada. Essa alteração não pode resultar em um nível de inventário abaixo de zero ou acima da capacidade máxima em nenhum momento durante o processamento.

- ***Sequence dependent setup time*** (s_{jk}): a restrição s_{jk} representa o tempo de preparação da máquina entre os processamentos das tarefas subsequentes j e k. O termo s_{0j} indica o tempo de preparação antes da primeira tarefa, enquanto s_{i0} o tempo de limpeza após o término da última tarefa na sequência. Se o termo s_{jk} não aparece no campo β, indica que não há tempo de preparação de máquina dependente da sequência e são todos iguais a 0.

O campo γ denota a função objetivo do problema que deve ser minimizada. Alguns dos principais objetivos encontrados na literatura estão descritos a seguir:

- ***Makespan*** (C_{max}): o *makespan* representa o tempo total de processamento, definido como $max(C_1, \ldots, C_n)$, e é equivalente ao tempo de conclusão da última

tarefa a ser processada. A minimização do *makespan* resulta em uma utilização eficiente da(s) máquina(s).

- ***Maximum Lateness*** (L_{max}): denota o pior caso de violação das datas limites de conclusão das tarefas.
- ***Total weighted completion time*** ($\sum \omega_j C_j$): representa a soma ponderada dos tempos de conclusão das n tarefas. Cada tarefa j tem um peso ω_j associado e é um indicativo dos custos operacionais ou logísticos associado.
- ***Total weighted tardiness*** ($\sum \omega_j T_j$): indica a soma dos atrasos ponderados das tarefas, considerando os respectivos pesos associados ω_j.
- ***Weighted number of tardy jobs*** ($\sum \omega_j U_j$): similar à função anterior, no entanto, considera o número de tarefas atrasadas.

Seguindo a notação de três campos e as definições apresentadas por Graham et al. (1979) e Pinedo (2022), o problema de minimização do *makespan* no sequenciamento de tarefas com datas de liberação e restrições de inventário, também abreviado anteriormente como PSDLI, é denotado como $1|r_j, inv|C_{max}$. O valor 1 no campo α indica que o ambiente de processamento é de máquina única. As notações r_j e inv no campo β caracterizam as restrições de data de liberação e inventário, respectivamente. Já o objetivo C_{max}, representado no campo γ, é de minimização do *makespan*.

A seguir, serão abordados os trabalhos encontrados na literatura acerca dos problemas envolvendo o escalonamento de tarefas com restrições de inventário.

2.2 Literatura relacionada

Uma característica importante do PSDLI, presente em diversas aplicações práticas do mundo real, é a restrição de inventário. Neumann e Schwindt (2003) demonstraram que essas restrições de inventário são uma generalização de recursos renováveis e não-renováveis.

Na literatura, essas restrições têm sido amplamente aplicadas a contextos reais, como no gerenciamento de projetos, terminais logísticos de carga e descarga de cami-

nhões, perfuração de poços de óleo e gás, distribuição de energia, indústria química, usinas nucleares e processos de manufatura. Em geral, todas as tarefas estão disponíveis antes do início do processamento e, à medida que são processadas, consomem ou adicionam mais recursos ao inventário. A quantidade de recursos modificados depende de cada tarefa e o nível de inventário não pode ser negativo em nenhum instante durante o processamento.

Carlier e Kan (1982) abordaram o problema de *scheduling* com restrições de recursos não-renováveis consumidos no processamento de cada tarefa. Desde então, inúmeros trabalhos na literatura propuseram abordagens para diferentes variantes desse problema com restrições de recursos e inventário. No entanto, poucos estudos tratam do problema de escalonamento com restrições de inventário em um ambiente de máquina única. Por esse motivo, a revisão da literatura considera não apenas os problemas de sequenciamento ($\alpha = 1$), mas também trabalhos que envolvem ambientes com múltiplas máquinas idênticas (P_m) e com velocidades dependentes das tarefas (R_m) em paralelo.

Além do *makespan* (C_{max}), também são consideradas diferentes funções objetivo, como a minimização do atraso no processamento de tarefas com diferentes pesos ($\sum \omega_j T_j$), mesma prioridade de processamento ($\sum T_j$), máximo atraso (L_{max}) e número de tarefas atrasadas com pesos diferentes ($\sum \omega_j U_j$). Esses trabalhos são abordados nesta Seção e estão listados na Tabela 2.1.

O primeiro trabalho de sequenciamento de tarefas com restrições de inventário encontrado na literatura aplicado ao gerenciamento de projetos, foi realizado por Neumann e Schwindt (2003). Nesse estudo, os autores investigaram um problema de sequenciamento de processos na indústria química e propuseram um algoritmo B&B. Os experimentos reportados demonstraram que o algoritmo foi capaz de resolver instâncias com até 100 tarefas em menos de um minuto.

Bartels e Zimmermann (2009) abordaram o problema de escalonamento de tarefas em um ambiente de testes em projetos de Pesquisa e Desenvolvimento (P&D) na indústria automotiva, contemplando restrições de inventário, datas mínima e máxima entre duas tarefas subsequentes, bem como recursos renováveis e acumulativos. Eles formularam e implementaram um modelo *mixed integer linear programming* (MILP) para o

problema em instâncias com até 20 tarefas e 4 variantes de veículos, solucionando o problema em tempo inferior a 10 segundos. Além disso, motivados pelo trabalho de Neumann e Zimmermann (2000), desenvolveram uma heurística baseada em regras de prioridade para o escalonamento em instâncias com até 600 tarefas e 25 variantes de veículos.

Como *benchmark*, Bartels e Zimmermann (2009) consideraram as soluções manuais aplicadas ao problema até aquele momento. Em termos de qualidade de solução, a abordagem heurística proposta foi superior à manual em 26%. No entanto, os autores sugeriram investigar técnicas de busca local para aprimorar ainda mais as soluções obtidas pela heurística baseada em regras de prioridade e avaliar a aplicabilidade desses métodos em problemas relacionados a outros tipos de projetos de P&D, como na indústria aeronáutica.

Posteriormente, Bartels e Zimmermann (2015) desenvolveram uma variante da heurística baseada em regras de prioridade e um Algoritmo Genético (AG) para o mesmo problema de escalonamento de testes de veículos em uma unidade automotiva de P&D. Baseado nos resultados de experimentos computacionais, os autores reportaram que a heurística com base em regras de prioridades obteve melhores resultados do que o AG em instâncias com até 100 tarefas.

Além da indústria automotiva, outra aplicação prática encontrada na literatura é no descomissionamento de usinas nucleares, abordado por Bartels et al. (2011). De acordo com os autores, o custo de desmontagem de um reator nuclear pode ultrapassar um bilhão de euros, além de ser um projeto com alta complexidade de execução, em que o desenvolvimento de uma abordagem matemática apropriada para o escalonamento eficiente de tarefas nesse problema tem um impacto financeiro e operacional muito significativo. Nesse contexto, foi proposto um modelo exato de enumeração, baseado em relaxação Lagrangiana, capaz de resolver o problema de minimização do custo total através do sequenciamento de até 50 tarefas com restrições de inventário e janelas de tempo de atraso.

Briskorn et al. (2010), por sua vez, abordaram o problema prático de sequenciamento de caminhões com diferentes prioridades em um terminal de cargas e descargas de modo a minimizar o *makespan*. O ambiente de máquina única corresponde a um

terminal com um único portão em que os caminhões são as tarefas que retiram ou adicionam unidades de itens no inventário. Como restrição, nenhum caminhão pode ser carregado com um volume maior do que o disponível no inventário, nem descarregar mais itens do que o disponível para atingir a capacidade máxima do terminal. A principal contribuição do trabalho de Briskorn et al. (2010) foi a prova de que esse problema, denotado por $1|inv|\sum \omega_j C_j$, é fortemente $\mathcal{NP}$-difícil.

Posteriormente, Briskorn et al. (2013) geraram 8640 instâncias de tamanhos $n \in \{5, 10, 15, 20, 25\}$ para o mesmo problema $1|inv|\sum \omega_j C_j$. Os autores propuseram dois algoritmos exatos baseados nas abordagens B&B e Programação Dinâmica (PD) para o problema. Ao final, o B&B foi o único que encontrou mais soluções ótimas para instâncias de até 20 tarefas, no entanto, nenhum algoritmo foi capaz de resolver todas as instâncias a partir desse tamanho.

Briskorn e Leung (2013) consideraram o problema de minimização do tempo máximo de atraso no sequenciamento de tarefas em um ambiente de máquina única e restrições de inventário ($1|inv|L_{max}$), introduzido por Briskorn et al. (2010). A motivação do trabalho também foi um terminal de cargas como um ambiente de máquina única onde os caminhões despacham ou carregam itens para serem distribuídos. Cada caminhão pode modificar o inventário de acordo com o número de unidades comportadas. Com base nas fundamentações do problema prático e na regra *earliest-due-date*, os autores reduziram o espaço de busca e adotaram uma estratégia mais eficiente de reordenação das tarefas em uma sequência. Utilizando o algoritmo B&B, os autores conseguiram resolver instâncias com tamanhos de até 60 e 90 tarefas em menos de 60 e 1800 segundos, respectivamente.

No mesmo ano, Torabi et al. (2013) propuseram uma abordagem multiobjetivo a partir da heurística *particle swarm optimization* (PSO) para o problema de *scheduling* em um ambiente de máquinas paralelas com diferentes velocidades de processamento para cada tarefa ($\alpha = R_m$). Como restrições, as tarefas tinham datas de liberação, tempos de preparação dependentes da máquina e da sequência. Além disso, os tempos de processamento e conclusão das tarefas foram considerados como incertos. As funções objetivos de minimização consideradas pelos autores foram: (i) o tempo total ponderado de processamento, (ii) o tempo total ponderado de atraso e (iii) a diferença entre

o *makespan* total e o *makespan* em cada máquina. Os autores desenvolveram um algoritmo MOPSO (do inglês, *multi-objective particle swarm optimization*), e compararam com o algoritmo convencional CMOPSO (do inglês, *conventional multi-objective particle swarm optimization*). A diferença entre os dois algoritmos consistiu na modificação dos parâmetros de seleção social e cognitivo das partículas. Os autores reportaram que a abordagem proposta obteve melhores resultados comparada à versão convencional do algoritmo. Além disso, o algoritmo proposto foi capaz de gerar soluções mais diversas, ao direcionar as partículas para diferentes regiões do espaço de soluções eficientes ou ótimas de Pareto.

Ghorbanzadeh et al. (2019) desenvolveram uma heurística e dois algoritmos exatos baseados em PD e B&B para o problema $1|r_j, inv|C_{max}$, aplicado a um terminal de cargas com datas de liberação para carga e descarga e restrição de inventário no armazenamento. Após os experimentos, os autores relataram que a abordagem heurística apresentou um desempenho superior, encontrando a solução ótima na maioria dos casos.

Ranjbar e Saber (2021), ampliando a análise iniciada por Briskorn e Leung (2013), propuseram um modelo MILP e a meta-heurística *variable neighborhood search* (VNS) para o problema $1|inv|L_{max}$. Para avaliar os resultados do algoritmo VNS, os autores modificaram os algoritmos AG e PSO desenvolvidos por Bazgosha et al. (2017) para o problema $P_m|r_j, inv|C_{max}$. Considerando o conjunto de instâncias ($n \leq 30$) geradas pelos próprios autores, o VNS conseguiu encontrar a solução ótima em 95% dos casos em menos de 180 s, superando os algoritmos AG e PSO em qualidade e custo computacional. Como oportunidade de trabalhos futuros, os autores sugeriram o desenvolvimento de novas heurísticas para o mesmo problema e outras variantes. Além disso, o trabalho considera apenas instâncias com tamanho inferior a 50 tarefas, indicando a necessidade de desenvolver heurísticas para problemas com instâncias maiores e restrições de inventário.

Nos últimos anos também foram desenvolvidos outros trabalhos considerando restrições de inventário para o escalonamento de tarefas em ambientes com mais de uma máquina. Chen e Wu (2006) propuseram uma heurística híbrida com base nas metaheurísticas de busca local *threshold-accepting* (TA) e *tabu search* (TS) para o problema

de minimização do tempo total de atraso no escalonamento de tarefas no ambiente de máquinas paralelas com restrições de *setup* e inventário, denotado por $R_m|s_{ij}, inv|T_{max}$. Eles avaliaram a heurística proposta comparando-a com as abordagens *apparent tardiness cost with setup* (ATCS) e *simulated annealing* (SA), propostas por Lee e Pinedo (1997) e Tamaki et al. (1993), respectivamente. Eles identificaram que a heurística desenvolvida superou o algoritmo SA em qualidade e tempo, no entanto, obteve um custo computacional maior que o ATCS.

Cappadonna et al. (2013), por sua vez, propuseram uma heurística híbrida para o problema de processamento de tarefas com tempos de *setup* e restrição de inventário em máquinas paralelas independentes, denotado por $R_m|s_{ij}, inv|C_{max}$. Eles formularam um modelo MILP para obter soluções exatas do problema e implementaram um AG a fim de obter boas soluções aproximadas em instâncias maiores.

Avdeenko et al. (2017) modelaram um problema da indústria de perfuração de poços de óleo e gás como $R_m|r_j, inv|C_{max}$, propuseram uma formulação matemática e dois algoritmos. Um deles, com abordagem exata, conseguiu resolver problemas com até 5 máquinas paralelas e 40 tarefas. Já o outro algoritmo polinomial conseguiu resolver instâncias maiores, mas sem garantir o ótimo.

No mesmo ano, Fanjul-Peyro et al. (2017) modelaram o problema de minimização do *makespan* no escalonamento de tarefas em máquinas paralelas com restrições de recursos. Eles assumiram que cada máquina precisa de recursos para processar cada tarefa. Além disso, os recursos são renováveis, ou seja, ficam disponíveis novamente após o processamento de cada tarefa, e são discretos porque a sua quantidade é representada por um número inteiro positivo e só são necessários durante o processamento. Eles implementaram um algoritmo guloso denominado *greedy-based fixing algorithm* (GBF) cujos resultados foram melhores que os modelos desenvolvidos de *mixed integer programming* (MIP) em instâncias pequenas e médias. No entanto, para instâncias a partir de 50 tarefas e 15 máquinas, o GBF não obteve boas soluções.

Yepes-Borrero et al. (2020) e Soares e Carvalho (2022) também abordaram o problema de minimização do *makespan* em máquinas paralelas. No entanto, assim como Cappadonna et al. (2013), eles utilizaram o tempo de *setup* e os recursos limitados como restrições. Diferentemente dos demais trabalhos citados, Yepes-Borrero et al.

(2020) consideraram o consumo de recursos durante o *setup* ao invés do processamento das tarefas. Eles desenvolveram três meta-heurísticas combinando as estratégias de construção e busca local. Diferente da terceira, as duas primeiras meta-heurísticas não consideraram as restrições de inventário durante a fase construtiva e, apesar de não haver diferença significativa nos *gaps* encontrados para instâncias menores, a abordagem com restrições de inventário na fase construtiva obteve melhores resultados em termos de qualidade das soluções encontradas.

Como mencionado na Seção 1.2, um dos estudos que mais motivaram o presente trabalho foi o de Davari et al. (2020). Após apresentarem uma descrição formal do problema $1|r_j, inv|C_{max}$, os autores propuseram dois modelos MILP: o primeiro baseado em cada unidade discreta de tempo, enquanto o segundo com base na sequência de tarefas. Em seguida, eles propuseram um algoritmo exato B&B e uma abordagem GC para resolver o problema de forma exaustiva. Apesar dos autores terem reportado a superioridade do GC em termos do tempo de execução e número de instâncias resolvidas em relação ao B&B, o GC enfrentou dificuldades para resolver instâncias maiores, particularmente aquelas com tamanho $n \in \{90, 100\}$. Nesse contexto, há a oportunidade de desenvolver outras abordagens heurísticas mais eficientes em relação ao tempo e qualidade das soluções encontradas para instâncias maiores do problema ($n \geq 50$).

A Tabela 2.1 apresenta de forma resumida os trabalhos supracitados, ambientes de máquina, restrições, objetivos e abordagens utilizadas.

Tabela 2.1: Principais trabalhos relacionados com o problema de sequenciamento de tarefas com restrições de inventário.

Referência	α	β	γ	**Abordagem**
Chen e Wu (2006)	R_m	s_{ij}, inv	T_{max}	Heurística
Briskorn et al. (2009)	1	inv	L_{max}, $\sum \omega_j C_j$, $\sum \omega_j U_j$, $\sum \omega_j T_j$	Heurística GA
Briskorn et al. (2013)	1	inv	$\sum \omega_j C_j$	B&B, PD
Briskorn e Leung (2013)	1	inv	L_{max}	B&B, heurísticas
Briskorn e Pesch (2013)	1	inv	L_{max}, $\sum \omega_j C_j$, $\sum \omega_j U_j$, $\sum \omega_j T_j$	Heurísticas VND, VNS, GA
Cappadonna et al. (2013)	R_m	s_{ij}, inv	C_{max}	Heurística GA
Torabi et al. (2013)	R_m	r_j, s_{ij}, inv	$\sum \omega_j T_j$, $\sum \omega_j L_j$, $\sum \omega_j C_j$	Heurística MOPSO
Morsy e Pesch (2015)	1	inv	$\sum \omega_j C_j$	Heurística 2-*approximation*
Bazgosha et al. (2017)	P_m	r_j, inv	C_{max}	Heurísticas GA, PSO, CO
Avdeenko et al. (2017)	R_m	r_j, inv	C_{max}	Heurística
Fanjul-Peyro et al. (2017)	R_m	inv	C_{max}	Heurística
Ghorbanzadeh et al. (2019)	1	r_j, inv	C_{max}	B&B, heurística
Davari et al. (2020)	1	r_j, inv	C_{max}	Heurística GC
Yepes-Borrero et al. (2020)	R_m	s_{ij}, inv	C_{max}	Meta-heurística GRASP
Ranjbar e Saber (2021)	1	inv	$\sum \omega_j T_j$	Heurística VNS
Soares e Carvalho (2022)	P_m	s_{ij}, inv	C_{max}	Heurística híbrida

2.3 Problema de sequenciamento de tarefas com datas de liberação e restrições de inventário

Conforme explicitado anteriormente, o objetivo do problema é minimizar o *makespan* em um ambiente de máquina única com restrições de data de liberação e inventário. Dessa forma, deve-se encontrar uma sequência π de n tarefas a serem processadas em uma única máquina, em que cada tarefa deve ser iniciada somente após a sua data de

liberação. Além disso, o processamento de cada tarefa implica em uma alteração dos recursos disponíveis no inventário e a tarefa só pode ser iniciada se essa alteração não tornar o nível do inventário negativo ou ultrapassar a sua capacidade máxima.

2.3.1 Exemplo numérico

Seja J um conjunto que representa uma instância de 5 tarefas ($n = 5$), um inventário com ocupação inicial $I_0 = 10$ e capacidade $I_C = 12$, a obtenção de uma solução para o problema de sequenciamento dessas tarefas, de modo a minimizar o *makespan* e não ultrapassar a capacidade máxima I_C do inventário ou torná-lo negativo durante o processamento, é demonstrada a seguir.

A Tabela 2.2 sumariza os dados de datas de liberação (r_j), tempos de processamento (p_j) e modificações de inventário (δ_j) para as tarefas $j \in J$.

Tabela 2.2: Exemplo dos dados r_j, p_j e δ_j para o problema PSDLI com $n = 5$

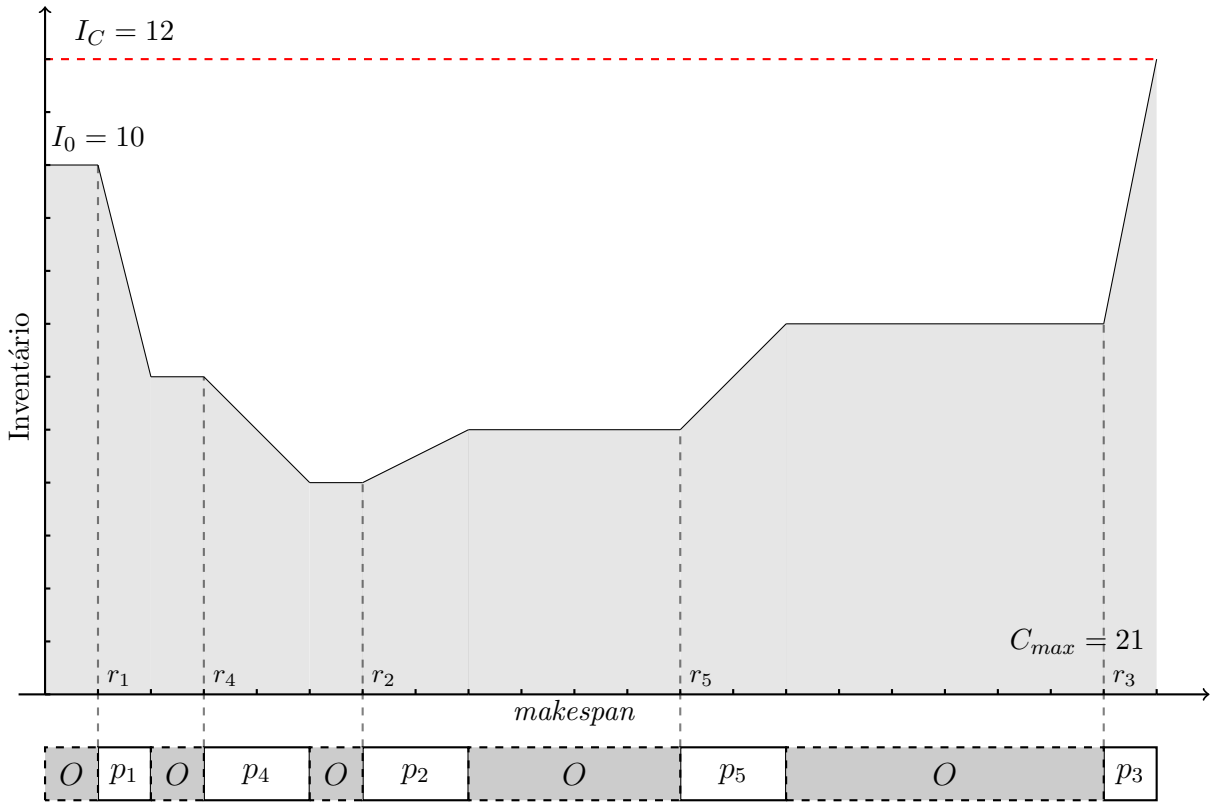
j	r_j	p_j	δ_j
1	1	1	-4
2	6	2	1
3	20	1	5
4	3	2	-2
5	12	2	2

Seja a sequência $\pi = (1, 4, 2, 5, 3)$ uma solução viável para esse problema, a Figura 2.1 ilustra o sequenciamento das tarefas e níveis de inventário durante o processamento.

Nota-se que r_1 representa a menor data de liberação, e por ser maior do que zero não há processamento antes desse momento. Esse tempo ocioso é representado por O. A partir do tempo de finalização da tarefa 1, observa-se que há mais tempos ociosos entre o término do processamento de uma tarefa e a data de liberação da tarefa sucessora. Considerando esses tempos ociosos e os tempos de processamento das tarefas, satisfazendo as restrições de data de liberação, o tempo total de conclusão C_{max} é igual a 21.

A Figura 2.1 destaca também as modificações de inventário inerentes à ordem de processamento das tarefas, considerando o nível de ocupação inicial $I_0 = 10$ e a capacidade máxima $I_C = 12$.

Figura 2.1: Exemplo da solução $s = (1, 4, 2, 5, 3)$



Fonte: Elaboração Própria.

A seguir, são apresentadas uma descrição formal e três formulações encontradas na literatura para o PSDLI.

2.3.2 Descrição do problema

Seja $J = \{1, \ldots, n\}$ um conjunto de tarefas a serem processadas, cada tarefa $j \in J$ tem um tempo de processamento $p_j \in \mathbb{R}^+$, e está associada a uma data de liberação $r_j \in \mathbb{R}^+$ e a 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, $\delta_j < 0$ para $j \in J^-$ e $\delta_j \geq 0$ para $j \in J^+$. As tarefas devem ser processadas em um ambiente de máquina única, 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 tarefa antecessora $j - 1$. O nível inicial do inventário e a sua capacidade são denotados por I_0 e I_C, respectivamente, em que $I_0 \geq 0$ e $I_C > 0$.

Dada uma sequência $\pi = (\pi_1, \ldots, \pi_n)$ que representa um conjunto de n sequências unitárias, ela é uma solução viável se $0 \leq I_0 + \sum_{k=1}^{s} \delta_{\pi_k} \leq I_C$ para cada $s = 1, \ldots, n$. 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 iniciarem no menor tempo possível com base na sequência π, o *makespan* da sequência π é denotado por $C_{max}(\pi)$ ou $C_{\pi_n}(\pi)$.

De forma resumida, o PSDLI consiste em definir uma sequência de tarefas em J, cujo tempo total até a finalização da última tarefa é minimizado e as seguintes restrições devem ser satisfeitas:

(i) cada tarefa deve ser processada uma única vez;

(ii) uma máquina só pode processar uma tarefa por vez;

(iii) a tarefa só pode ser iniciada após a sua data de liberação;

(iv) a modificação de recursos no inventário ao processar uma tarefa não pode exceder a sua capacidade ou torná-lo negativo.

2.3.3 Modelos da literatura

Nessa Seção são descritos os modelos encontrados na literatura para o problema $1|r_j, inv|C_{max}$ considerado no presente trabalho. A Tabela 2.3 apresenta de forma resumida cada uma das formulações e indexação utilizada. A seguir, cada uma dessas formulações é formalmente apresentada e descrita na forma compacta.

Tabela 2.3: Formulações da literatura para o PSDLI.

Formulação	**Trabalho**	**Indexação**
F1	Davari et al. (2020)	Sequência de tarefas
F2	Davari et al. (2020)	Unidades de tempo
F3	Ghorbanzadeh et al. (2019)	Unidades de tempo

Formulação F1 de Davari et al. (2020)

Davari et al. (2020) desenvolveram uma formulação matemática MIP baseada na sequência de tarefas para o PSDLI. A formulação compacta é apresentada conforme segue.

Conjuntos:

J: conjunto de tarefas a serem processadas.

Dados:

n: número de tarefas a serem processadas.

p_j: tempo de processamento da tarefa j.

r_j: data de liberação da tarefa j.

δ_j: modificação de recursos no inventário ao processar a tarefa j.

I_0: quantidade inicial de recursos no inventário.

I_C: capacidade do inventário.

Variáveis de decisão:

x_{js}: variável binária cujo valor é 1 se a tarefa j é processada na posição s, 0 no caso contrário.

y_s: nível de recursos no inventário ao processar a tarefa da posição s.

t_s: tempo de conclusão do processamento da tarefa na ordem s.

Objetivo:

$$\min t_n \tag{2.1}$$

Sujeito a:

$$\sum_{s=1}^{n} x_{js} = 1 \qquad j \in J \tag{2.2}$$

$$\sum_{j \in J} x_{js} = 1 \qquad s = 1, \ldots, n \tag{2.3}$$

$$t_s \geq \sum_{j \in J} x_{js}(r_j + p_j) \qquad s = 1, \ldots, n \tag{2.4}$$

$$t_s \geq t_{s-1} + \sum_{j \in J} x_{js} p_j \qquad s = 2, \ldots, n \tag{2.5}$$

$$y_1 = I_0 + \sum_{j=1}^{n} \delta_j x_{j1} \tag{2.6}$$

$$y_s = y_{s-1} + \sum_{j=1}^{n} \delta_j x_{js} \qquad s = 2, \ldots, n \tag{2.7}$$

$$0 \leq y_s \leq I_c \qquad s = 1, \ldots, n. \tag{2.8}$$

Nessa formulação, o objetivo é minimizar o tempo de conclusão t_n da última tarefa na sequência, ou simplesmente *makespan*. As restrições (2.2) garantem que somente uma tarefa deve ser processada na posição s, enquanto as restrições (2.3) asseguram que cada tarefa deve ser processada uma única vez. As restrições (2.4) e (2.5) determinam que cada tarefa não pode iniciar antes da sua data de liberação e só pode ocorrer após a conclusão da tarefa antecessora na sequência, respectivamente. As restrições (2.6), (2.7) e (2.8) asseguram que o nível de inventário após o processamento da primeira tarefa considera a capacidade inicial do inventário, o nível de inventário após a conclusão de cada tarefa subsequente depende do nível de inventário anterior ao seu processamento e que em nenhum momento esse nível ultrapasse a capacidade máxima do inventário ou se torne negativo.

Formulação F2 de Davari et al. (2020)

Na formulação F2, o problema é modelado em unidades discretas de tempo para representar os instantes em que as tarefas iniciam o processamento.

A variável de decisão x_{jt} assume o valor 1 se a tarefa j inicia o processamento no instante de tempo t, e zero no caso contrário. A variável de decisão y_t representa o nível de inventário no instante de tempo t. O intervalo de tempo $T = \{t_{\min}, t_{\min}+1, \ldots, t_{\max}\}$ é o conjunto de potenciais unidades de tempo em que ocorre o início de processamento de cada tarefa, onde:

- $t_{\min} = \min_{j \in J}\{r_j\}$
- $t_{\max} = \max_{j \in J}\{r_j\} + \sum_{j \in J} p_j - \min_{j \in J}\{p_j\}$

Para cada tarefa j, é definido outro conjunto menor $T_j = \{r_j, r_j + 1, \ldots, t^j_{max}\}$, onde: $t^j_{max} = \max\left\{ r_j; \max_{i \in J\setminus\{j\}} \{r_i\} + \sum_{i \in J\setminus\{j\}} p_i \right\}$.

Na forma compacta, a formulação F2 com base em unidades discretas de tempo é definida da seguinte forma.

Objetivo:

$$\min z \tag{2.9}$$

Sujeito a:

$$\sum_{t \in T_j} x_{jt} = 1 \qquad j \in J \tag{2.10}$$

$$\sum_{j \in J} \sum_{\tau = t - p_j}^{t-1} x_{j\tau} \leq 1 \qquad t \in T \tag{2.11}$$

$$z \geq \sum_{t \in T_j} t x_{jt} + p_j \qquad j \in J \tag{2.12}$$

$$y_{t_{min}} = I_0 + \sum_{j=1}^{n} \delta_j x_{jt_{min}} \tag{2.13}$$

$$y_t = y_{t-1} + \sum_{j=1}^{n} \delta_j x_{jt} \qquad t \in T\setminus\{t_{min}\} \tag{2.14}$$

$$0 \leq y_t \leq I_c \qquad t \in T. \tag{2.15}$$

Nessa formulação, o objetivo é minimizar o *makespan* z. As restrições (2.10) exigem que cada tarefa deva iniciar somente uma vez. As restrições (2.11) garantem que não há sobreposição no processamento das tarefas, enquanto o conjunto de restrições (2.12) determina o *makespan* para cada tarefa iniciada. Por fim, as restrições de inventário são garantidas através das restrições (2.13) - (2.15). Essa formulação representa corretamente o problema somente quando o tempo é discreto, ou seja, as datas de liberação e tempo de processamento são números inteiros.

Formulação F3 de Ghorbanzadeh et al. (2019)

Nessa formulação, os autores também modelaram o problema de acordo com unidades de tempo, no entanto, considerando o instante de finalização do processamento das tarefas. A variável de decisão x_{jt} assume valor 1 se o processamento da tarefa j é finalizado no instante t e zero no caso contrário.

De forma compacta, a formulação F3 é definida da seguinte forma.

Conjuntos:

$J = \{1, 2, \ldots, n\}$: conjunto de tarefas que consomem e adicionam recursos do inventário com índices i e j.

$T = \{s^*, \ldots, |T|\}$: conjunto que contém as unidades discretas de tempo t disponíveis para processamento das tarefas, onde $s^* = \min_{j \in J}\{r_j + p_j\}$.

Dados:

n: número de tarefas a serem processadas.

p_j: tempo de processamento da tarefa j.

r_j: data de liberação da tarefa j.

δ_j: modificação de recursos no inventário ao processar a tarefa j.

I_0: quantidade inicial de recursos no inventário.

I_C: capacidade do inventário.

$|T|$: limite superior para tempo de conclusão de todas as tarefas.

Variáveis de decisão:

z: valor do *makespan* da solução

x_{jt}: variável binária que assume valor 1 se a tarefa j completa o processamento no instante t, 0 no caso contrário.

y_t: nível de inventário no instante t.

Objetivo:

$$\min z \tag{2.16}$$

Sujeito a:

$$\sum_{t=s^*}^{|T|} x_{jt} = 1 \qquad j \in J \tag{2.17}$$

$$\sum_{\tau=t-p_j+1}^{t} \sum_{i \in J \setminus \{j\}} x_{i\tau} \leq M(1 - x_{jt}) \qquad j \in J, t \in T \tag{2.18}$$

$$\sum_{t=s^*}^{|T|} t x_{jt} - p_j \geq r_j \qquad j \in J \tag{2.19}$$

$$z \geq \sum_{t=s^*}^{|T|} t x_{jt} \qquad j \in J \tag{2.20}$$

$$y_{s^*} = I_0 \tag{2.21}$$

$$y_t = y_{t-1} + \sum_{j \in J} \delta_j x_{jt} \qquad t \in T \tag{2.22}$$

$$0 \leq y_t \leq I_c \qquad t \in T. \tag{2.23}$$

A Equação (2.16) minimiza o *makespan* da solução. As restrições (2.17) garantem que cada tarefa deve ser processada uma única vez. Utilizando um número M suficientemente grande, as restrições (2.18) asseguram que somente uma tarefa deve ser processada em cada instante de tempo. O conjunto de restrições (2.19) indica que o início do processamento de cada tarefa deve ocorrer somente após a respectiva data de liberação. As restrições impostas para o cálculo do *makespan* são definidas através da Equação (2.20). Por fim, as restrições (2.21)-(2.23) garantem que o nível de inventário seja positivo e menor do que a capacidade máxima em qualquer instante de tempo durante o processamento.

Capítulo 3

Algoritmo Proposto

Este capítulo descreve o algoritmo utilizado para a resolução do problema de sequenciamento de tarefas em um ambiente de máquina única com data de liberação e restrições de inventário, denotado como $1|r_j, inv|C_{max}$ na notação de três campos. Devido ao problema pertencer à classe $\mathcal{NP}$-difícil, é apresentado um algoritmo heurístico populacional híbrido que contém um operador de destruição e reconstrução, do inglês R&R, para a diversificação das soluções. Simultaneamente, é utilizado um procedimento de busca local baseado na heurística RVND, composto por três estruturas de vizinhança que intensificam a busca local por novas soluções e ajudam o algoritmo a escapar de ótimos locais.

3.1 Algoritmo HyPRR

A abordagem heurística híbrida proposta é denominada HyPRR (Teixeira et al., 2023), foneticamente *"hyper"*, e o pseudocódigo é apresentado no Algoritmo 1. O início do método consiste na geração de uma população P com μ indivíduos (Linha 2), que representam as sequências de permutação de tarefas como possíveis soluções para o problema.

Após a criação da população P, cada indivíduo $\pi \in P$ é submetido à busca local através do procedimento RVND e atualizado na população (Linhas 3-4). 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). Após essa 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).

Nas próximas seções deste capítulo, cada etapa do HyPRR é apresentada de forma detalhada.

Algoritmo 1 Algoritmo HyPRR

```
1:  procedimento HyPRR(μ, λ, μ^elite, μ^close, It_NI, d)
2:      Gere uma população P de μ indivíduos;
3:      para cada π ∈ P faça
4:          π ← RVND(π)
5:      enquanto it < It_NI faça
6:          Selecione um indivíduo π de P utilizando torneio binário;
7:          π' ← R&R(π, d);
8:          π'' ← RVND(π')
9:          Insira π'' na população P;
10:         se |P| = μ + λ então
11:             P ← GerenciamentoPopulação(λ,μ^elite,μ^close);
12:         se f(π'') < f(best(P)) então
13:             it ← 0;
14:         senão
15:             it ← it + 1;
16:     retorne best(P);
```

Na Seção 3.2, é abordada a etapa de geração de novos indivíduos através do operador R&R.

3.2 Geração de novos indivíduos

Após a criação da população inicial com indivíduos gerados aleatoriamente e submetidos ao procedimento de busca local, é necessária a aplicação de mecanismos de intensificação e diversificação para geração de novos indivíduos, de modo a evitar que o método fique preso em ótimos locais. Nesse sentido, dois indivíduos são pré-selecionados de forma aleatória para comparação dos valores de aptidão, ou *fitness* em inglês. A aptidão representa a qualidade da solução do indivíduo e o quanto contribui para a diversidade da população. A Seção 3.4 detalha como o valor de aptidão é calculado.

Após a pré-seleção dos dois indivíduos, a sequência π com maior aptidão é selecionada por torneio binário e mantida (Linha 5). Em seguida, este indivíduo π é submetido ao operador R&R para obtenção de um novo indivíduo π'.

Em contraste com os movimentos realizados no procedimento de busca local, a operação *ruin-and-recreate* não tem como objetivo gerar um novo indivíduo π' cujo *makespan* seja obrigatoriamente menor do que o indivíduo gerado, ou seja, $C_{max}(\pi') < C_{max}(\pi)$. O principal objetivo do operador R&R, proposto por Schrimpf et al. (2000) para o *travelling salesman problem* (TSP), consiste em gerar um novo indivíduo que represente uma solução de boa qualidade para ser submetido ao procedimento de busca local.

O pseudocódigo do operador R&R é detalhado no Algoritmo 2. Inicialmente, de forma aleatória, são selecionadas (Linha 2) e removidas d tarefas do indivíduo π, obtendo-se um novo indivíduo parcial π_D (Linha 3). Todas as d tarefas removidas são adicionadas a uma lista LC (Linha 4) para posterior reinserção. O algoritmo, iterativamente enquanto LC não estiver vazia, seleciona uma tarefa de forma aleatória da lista LC (Linha 6) e avalia qual a posição de reinserção da tarefa que resulta no menor *makespan* dentro da sequência parcial π_D (Linha 7).

Em seguida, a tarefa é removida da lista LC (Linha 8) e adicionada à sequência parcial π_D (Linha 9) na posição obtida anteriormente. Ao fim da iteração, quando

todas as tarefas removidas são reinseridas, o operador retorna uma nova sequência π' para ser submetida ao procedimento de busca local (Linha 11).

Algoritmo 2 Algoritmo ruin-and-recreate

```
1:  procedimento R&R(π, d)
2:      Selecione aleatoriamente d tarefas de π;
3:      Remova as d tarefas de π, gerando uma sequência parcial π_D;
4:      Insira as d tarefas em uma lista LC;
5:      enquanto LC ≠ ∅ faça
6:          Selecione aleatoriamente uma tarefa j ∈ LC;
7:          Obtenha a posição de inserção de j em π_D que resulte no menor makespan;
8:          LC ← LC − j;
9:          π_D ← π_D + j;
10:     π' ← π_D;
11:     return π';
```

Dado que a população inicial é gerada aleatoriamente, os indivíduos podem representar soluções inviáveis em relação às restrições de inventário. Essas violações são penalizadas no cálculo da função objetivo, multiplicando-se a quantidade de violações na sequência por um fator ρ. Portanto, ao obter a melhor posição de reinserção das tarefas na sequência π, o operador R&R utiliza essa penalização e, consequentemente, busca restaurar a viabilidade da solução.

Dessa forma, além do próprio indivíduo gerado π, outro fator muito importante para a efetividade do operador R&R é o grau de perturbação da solução inicial, representado pelo número de tarefas d removidas e reinseridas. Se o grau de perturbação é baixo, o método se torna ineficaz para contribuir com a diversidade de novas soluções e fuga de ótimos locais. Por outro lado, se o grau de perturbação for muito alto, o algoritmo não consegue convergir de forma eficiente na busca de boas soluções e o seu tempo de execução também aumenta. Por esse motivo, foram realizados experimentos de calibração do operador R&R e os resultados são apresentados no Capítulo 4.

Na Seção que segue, é detalhado o procedimento de busca local utilizado, e que o novo indivíduo π' é submetido após a operação R&R.

3.3 Busca local

O procedimento de busca local é baseado na estratégia RVND proposta por Subramanian et al. (2010). O Algoritmo 3 detalha o pseudocódigo do procedimento para explorar as vizinhanças a partir de uma dada sequência π e tamanho do bloco *l-block-insertion*. O Algoritmo inicia com a criação de uma lista NL, do inglês *neighborhood list*, com elementos que representam os diferentes movimentos de vizinhança. Em seguida, de forma iterativa, cada estrutura $N^{(\eta)}$ é selecionada aleatoriamente da lista (Linha 4) para ser aplicada na sequência π e obter uma nova sequência π' com menor o *makespan* dentre as estruturas vizinhas (Linha 5). Caso a solução da nova sequência π' seja melhor que π (Linha 6), então a solução atual é substituída por π' (Linha 7) e a lista NL volta ao estado inicial com todas as estruturas presentes (Linha 8). Caso contrário, a estrutura selecionada que não resultou em melhoria é removida da lista NL (Linha 10). Esse procedimento é repetido até que todas as estruturas são exploradas sem resultar em melhoria no *makespan*, então o algoritmo é finalizado e retorna a sequência π (Linha 11).

Algoritmo 3 Algoritmo RVND

```
1:  procedimento RVND(π, N^(*))
2:      Inicie uma lista NL com todas as estruturas de vizinhanças N^(*);
3:      enquanto NL ≠ ∅ faça
4:          Selecione aleatoriamente uma estrutura N^(η) ∈ NL;
5:          Encontre o melhor vizinho π' de π ∈ N^(η);
6:          se C(π') < C(π) então
7:              π ← π';
8:              Atualize a lista NL com todas as estruturas de vizinhança;
9:          senão
10:             Remove N^(η) da lista NL;
11:     return π;
```

Compreendida a lógica do procedimento RVND, em seguida são detalhadas as estruturas de vizinhança e a avaliação de movimentos utilizadas, fundamentais para o

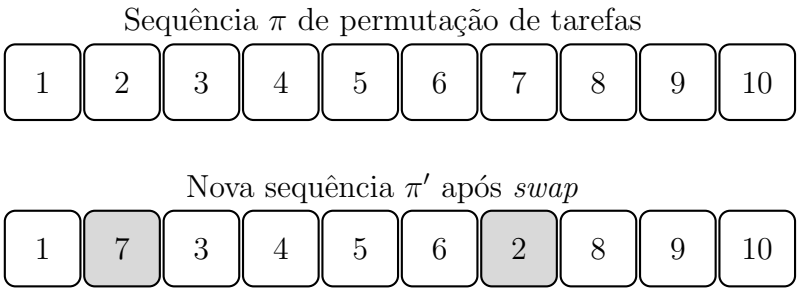
sucesso do algoritmo.

3.3.1 Estruturas de vizinhanças

Ao todo são exploradas 3 estruturas diferentes de vizinhança: *swap*, *insertion* e *l-block insertion*. Enquanto as estruturas *swap* e *insertion* resultam em apenas um tipo de movimento possível, a quantidade de vizinhanças que podem ser exploradas através da estrutura *l-block insertion* depende do tamanho do bloco. Essas estruturas de vizinhança são detalhadas a seguir.

- ***Swap***: duas tarefas da solução π são permutadas, gerando uma nova sequência π', como pode ser visto na Figura 3.1 em que as tarefas 2 e 7 trocam de posição.

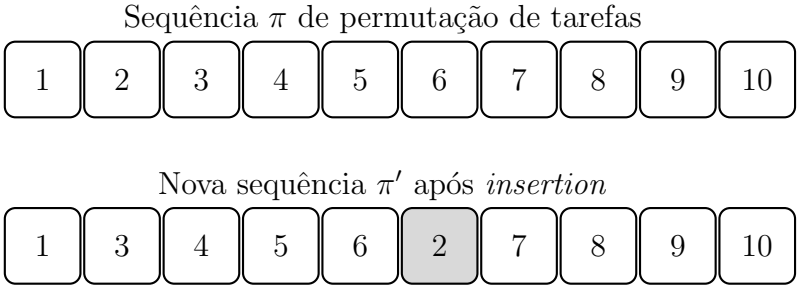
Figura 3.1: Estrutura de vizinhança *swap*



Fonte: Elaboração Própria.

- ***Insertion***: uma tarefa é retirada da sua posição original na sequência π e reinserida em outra posição, gerando uma nova sequência π'. Como exemplificado na Figura 3.2, a tarefa 2 foi removida e reinserida entre as tarefas 6 e 7.

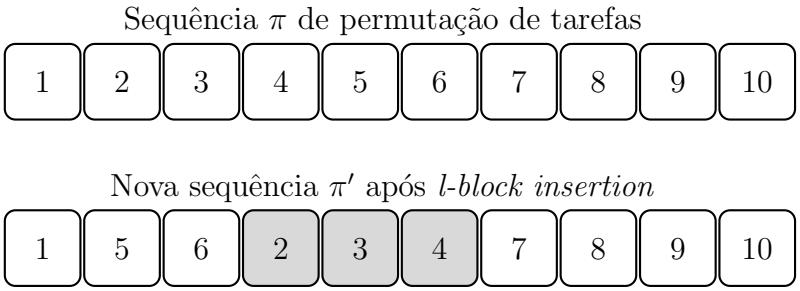
Figura 3.2: Estrutura de vizinhança *insertion*



Fonte: Elaboração Própria

- ***l-block insertion***: um bloco de l tarefas adjacentes é removido da sua posição original na solução π e reinserido em uma nova posição. Nota-se que cada tamanho de bloco está associado a uma vizinhança distinta, por exemplo, se $2 \leq l \leq 5$, então existem 4 vizinhanças diferentes correspondentes aos movimentos dos blocos de tamanhos 2, 3, 4 e 5. Além disso, é importante destacar que a estrutura *insertion* é um caso particular do *l-block insertion* quando o tamanho do bloco é igual a 1. No exemplo da Figura 3.3 em um bloco de tamanho $l = 3$, as tarefas adjacentes 2, 3 e 4 são removidas da sequência π e reinseridas após a tarefa 6.

Figura 3.3: Estrutura de vizinhança *l-block insertion*



Fonte: Elaboração Própria

Essas estruturas de vizinhança utilizadas no procedimento de busca local têm uma contribuição muito importante na capacidade do algoritmo HyPRR de intensificar as buscas no espaço de soluções, ao mesmo tempo que requer um custo computacional expressivo, em relação ao tempo total de execução do algoritmo, para avaliar os movimentos realizados de cada estrutura.

Por esse motivo, a Seção 3.3.2 descreve o método utilizado para avaliação eficiente de movimentos. Posteriormente, no Capítulo 4, são apresentados os experimentos computacionais realizados para avaliar a contribuição de cada estrutura de vizinhança na qualidade das soluções obtidas, bem como dos experimentos de calibração para definir o tamanho do bloco de reinserção.

3.3.2 Avaliação eficiente de movimentos

Dada uma sequência π com n tarefas que representa uma solução para o problema de *scheduling*, a escolha de uma subsequência e as possibilidades de movimentos para

explorar cada estrutura de vizinhança é de complexidade $\mathcal{O}(n^2)$. Adicionalmente, o cálculo do *makespan* na sequência π', resultante de cada estrutura de movimento, pode ser feito iterando toda a solução, o que gera uma complexidade adicional $\mathcal{O}(n)$. Dessa forma, a complexidade resultante para seleção de uma subsequência, estruturas de vizinhança e cálculo do tempo total de processamento 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). Dessa forma, a complexidade total do procedimento de busca local é reduzida para $\mathcal{O}(n^2)$.

Nessa estratégia de armazenamento de informações em uma estrutura de dados auxiliar, para cada subsequência $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$, 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;
- $Q_{min}(\pi)$ = nível mínimo de inventário durante o processamento de π;
- $Q_{max}(\pi)$ = nível 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 $Q_{max} = \max(0, \delta_j)$.

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 4. 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 (Linha 2). O nível mínimo de inventário da subsequência resultante é o menor valor entre $Q_{min}(\pi^1)$ e $Q_s(\pi^1) + Q_{min}(\pi^2)$ (Linha 3). De forma similar, é calculado o nível máximo de inventário da nova subsequência (Linha 4). O *makespan* de qualquer subsequência pode ser definido como $C(\pi) = E(\pi) + D(\pi)$. Caso o *makespan*

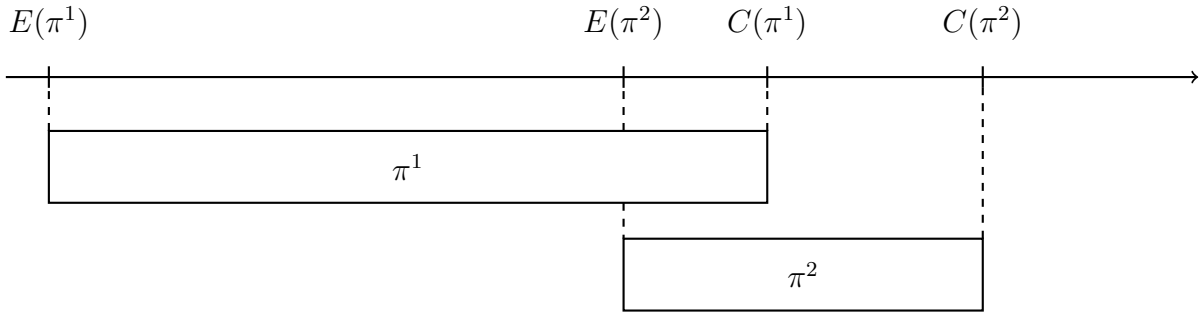
da subsequência π^1 seja menor que $E(\pi^2)$, a subsequência resultante $Q_s(\pi^1 \oplus \pi^2)$ tem um tempo ocioso $O = E(\pi^2) - C(\pi^1)$ (Linhas 5 - 7). Nesse caso, o valor de $E(\pi^1 \oplus \pi^2)$ é ajustado de acordo o tempo ocioso (Linha 8). Por fim, o valor de $D(\pi^1 \oplus \pi^2)$ é dado pela soma dos tempos de processamento das subsequências π^1 e π^2 (Linha 9).

Algoritmo 4 Concatenação $\pi^1 \oplus \pi^2$

1: **procedimento** CONCATENACAO-SUBSEQUENCIAS(π^1, π^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** $\pi^1 \oplus \pi^2$

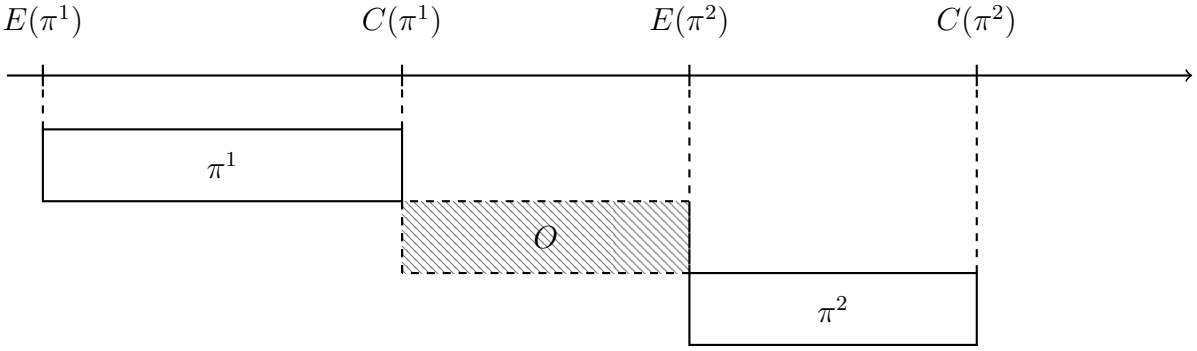
A concatenação de duas subsequências $\pi^1 \oplus \pi^2$ pode acarretar em duas possibilidades de tempo ocioso O entre a data de liberação $E(\pi^2)$ e o início da execução da subsequência π^2. O primeiro caso, como ilustrado na Figura 3.4, ocorre quando o tempo de conclusão da primeira subsequência $C(\pi^1)$ 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, ilustrado na Figura 3.5, 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 4.

Figura 3.4: Concatenação quando $C(\pi^1) > E(\pi^2)$



Fonte: Adaptado de Morais et al. (2024)

Figura 3.5: Concatenação das subsequências π^1 e π^2 quando $C(\pi^1) \leq E(\pi^2)$



Fonte: Adaptado de Morais et al. (2024)

3.4 Gerenciamento de diversidade

O gerenciamento de diversidade utilizado no HyPRR foi o mesmo utilizado no trabalho de Teixeira et al. (2023), e originalmente abordado por Vidal et al. (2012) em variantes do problema de roteamento de veículos.

Esse gerenciamento consiste na utilização de uma função de aptidão, ou *fitness*, como critério de manutenção dos indivíduos que mais contribuem com a população em termos de qualidade e diversidade de soluções. Além disso, a função de aptidão também é utilizada na avaliação dos indivíduos selecionados durante o torneio binário para seleção do indivíduo π, como especificado na Linha 5 do Algoritmo 1.

Enquanto o fator qualidade na função de aptidão é dado pelo valor do *makespan* de cada solução, o fator de diversidade dos indivíduos depende de dois parâmetros: o nú-

mero μ^{close} de indivíduos adjacentes mais próximos considerados no cálculo da distância para cada indivíduo e a quantidade μ^{elite} de indivíduos que devem ser preservados.

No cálculo da função de diversidade entre os indivíduos de uma população, são calculadas as distâncias entre os μ^{close} indivíduos mais próximos, considerando a similaridade dos vizinhos que um mesmo elemento compartilha nas duas soluções, e cada divergência na vizinhança de um elemento entre as duas soluções é adicionada um valor igual a 1 na distância entre os indivíduos. Ou seja, na comparação de duas sequências, é avaliado se cada uma das tarefas é antecedida e seguida pelas mesmas tarefas nas duas soluções.

Após o cálculo das distâncias entre os indivíduos mais próximos, eles são ordenados de forma decrescente para garantir que os indivíduos com maiores dissimilaridades em relação aos μ^{close} vizinhos tenham mais chances de permanência. Em seguida, é calculado o valor da função de diversidade $f_{div}(\pi)$, a partir da divisão da posição do indivíduo e do tamanho da população.

O valor da função objetivo $f_{obj}(\pi)$, por sua vez, considera o valor do *makespan* para ordenar os indivíduos de forma crescente. Após a ordenação crescente dos indivíduos pelo *makespan*, o valor da função $f_{obj}(\pi)$ é obtido a partir da divisão da posição do indivíduo e do tamanho da população. Dessa forma, os indivíduos que estão nas últimas posições após a ordenação têm maiores valores na função $f_{obj}(\pi)$.

Ao fim, a função de aptidão é calculada considerando a Equação (3.1). Além disso, se houver indivíduos iguais na mesma população, é adicionada uma penalização igual a 5 para cada ocorrência na função de aptidão.

$$f(\pi) = f_{obj}(\pi) + (1 - \frac{\mu^{elite}}{|P|}) \cdot f_{div}(\pi) \tag{3.1}$$

Como evidenciado no trabalho de Vidal et al. (2012), esse procedimento preserva a diversidade das soluções, ao mesmo tempo que garante a permanência dos melhores indivíduos μ^{elite} da população.

No Capítulo que segue são apresentados os experimentos computacionais realizados para calibração e avaliação do algoritmo desenvolvido, bem como os resultados desses experimentos.

Capítulo 4

Experimentos Computacionais

Este capítulo apresenta os resultados obtidos a partir dos experimentos de calibração dos parâmetros populacionais, das estruturas de vizinhança e do operador R&R empregados no algoritmo HyPRR, desenvolvido para o PSDLI. O algoritmo HyPRR foi implementado utilizando a linguagem C++ e os experimentos foram executados em um computador com processador Intel®Core™ i7-12700 de 4.90 GHz, 16 GB de memória RAM e sistema operacional Ubuntu 20.04 LTS 64 bits. Em todos os experimentos, foi adotado $\rho = 10000$ para garantir que as violações de viabilidade sejam fortemente penalizadas. O critério de parada adotado foi o mesmo utilizado por Mecler et al. (2021), em que $It_{NI} = \min(20n, 100)$ e n é o número de tarefas.

Para maximizar a eficiência do algoritmo, foram realizados experimentos de calibração dos parâmetros do HyPRR. Inicialmente, o algoritmo foi executado em todo o conjunto de 960 instâncias com os parâmetros populacionais já utilizados na literatura. Foram adotados os parâmetros populacionais reportados por Mecler et al. (2021) ($\mu = 20, \lambda = 40, \mu^{elite} = 10, \mu^{close} = 3$). Com relação ao tamanho de bloco do R&R, o valor adotado antes da calibração foi $d = 5$. Esse valor foi obtido com base em experimentos realizados por Ruiz e Stützle (2007), variando d entre 2 e 8 para um algoritmo *iterated greedy algorithm* desenvolvido para o $F_m|prmu|C_{max}$ e também utilizado por Teixeira et al. (2023) para o $F_m|block|C_{max}$. O tamanho de bloco de reinserção utilizado foi $l = 4$, assim como utilizado por Teixeira et al. (2023), e as três estruturas de

vizinhança descritas na Seção 3.3.1.

Com base nos resultados preliminares obtidos, foi realizado um processo de calibração em um subconjunto composto pelas 15 instâncias mais desafiadoras, ou seja, aquelas que apresentaram os maiores *gaps* no *makespan* no teste preliminar comparado aos resultados do GC reportados por Davari et al. (2020). Concluída a calibração, procedeu-se à execução do algoritmo HyPRR com os parâmetros calibrados em todo o conjunto de instâncias, e os resultados obtidos foram comparados aos do algoritmo GC, conforme apresentado na Seção 4.5. Em todos os experimentos realizados, o algoritmo foi executado 10 vezes em cada instância para cada combinação de parâmetros.

As próximas seções estão organizadas conforme segue: Na Seção 4.1, são descritas as instâncias utilizadas nos experimentos. Em seguida, na Seção 4.2, são apresentados os resultados de calibração das estruturas de vizinhança e do operador R&R. Na Seção 4.3, são detalhados os experimentos de calibração dos parâmetros populacionais. Na Seção 4.4, são apresentados os resultados de comparação do operador R&R com outros operadores encontrados na literatura. Finalmente, na Seção 4.5, são apresentados os resultados do HyPRR no conjunto total de instâncias, comparando-os com o algoritmo GC encontrado na literatura.

4.1 Instâncias e métrica de avaliação

Para o problema de sequenciamento de tarefas com datas de liberação e restrições de inventário, foram utilizadas as instâncias geradas por Davari et al. (2020). O conjunto é composto por 10 grupos de instâncias com diferentes tamanhos $n \in \{10, 20, 30, 40, 50, 60, 70, 80, 90, 100\}$.

Conforme descrito pelos autores, os tempos de processamento p_i $(1 \leq i \leq n)$ foram gerados a partir de uma distribuição uniforme inteira no intervalo $[1, \alpha]$, onde $\alpha \in \{10, 100\}$. As datas de liberação também foram definidas com base em uma distribuição uniforme inteira, com valores distribuídos no intervalo $[0, \tau \sum_{i \in J} p_i]$, em que $\tau \in \{0.5, 1, 1.5, 2\}$. De forma similar, os valores absolutos $|\delta_i|$ das modificações de inventário foram derivados de uma distribuição uniforme inteira no intervalo $[1, 10]$. A capacidade de inventário I_C foi aleatoriamente selecionada dentro do intervalo $[10\eta, 20\eta]$,

onde $\eta \in \{1, 3, 5\}$.

O valor do inventário inicial I_0, influencia a distribuição das tarefas entre os subconjuntos J^- e J^+, podendo tornar algumas instâncias inviáveis de serem solucionadas. Para mitigar essa possibilidade, os autores inicialmente distribuíram as tarefas entre os subconjuntos J^- e J^+ com uma probabilidade de 50% para cada um. Em seguida, caso $\sum_{j \in J} \delta_j > I_C$ ou $\sum_{j \in J} \delta_j < -I_C$, o processo é repetido até que a condição $-I_C \leq \sum_{j \in J} \delta_j \leq I_C$ seja atendida. Por fim, o valor do inventário inicial I_0 é determinado a partir do seguinte intervalo:

$$\left[\min\left\{ I_C; \max\left\{ 0; 0 - \sum_{j \in J} \delta_j \right\} \right\}, \max\left\{ 0; \min\left\{ I_C; I_C - \sum_{j \in J} \delta_j \right\} \right\} \right].$$

Quatro instâncias foram geradas para cada combinação de parâmetros (n, α, τ, η), totalizando 960 instâncias ($10 \times 2 \times 4 \times 3 \times 4 = 960$). Para avaliar o desempenho do algoritmo HyPRR, foram realizados testes de calibração em um subconjunto de 15 instâncias, listadas na Tabela 4.1.

Tabela 4.1: Instâncias utilizadas para calibração do algoritmo HyPRR

Número de tarefas (Tamanho)	Instância	*Makespan* GC
30	data30_79	297
40	data40_50	260
60	data60_25	333
60	data60_28	305
60	data60_32	365
70	data70_28	364
80	data80_4	395
80	data80_50	661
90	data90_2	476
90	data90_26	554
90	data90_33	438
90	data90_61	6460
100	data100_25	510
100	data100_34	540
100	data100_51	727

Para a avaliação da qualidade das soluções, utilizou-se a métrica *relative percent change* (RPC), que compara os resultados obtidos pelos algoritmos HyPRR e GC. *Gaps* positivos indicam que o *makespan* obtido pelo HyPRR foi maior do que o obtido pelo

GC, enquanto os *gaps* negativos indicam que o HyPRR superou o GC. Nos resultados apresentados nas seções seguintes, o RPC, referido também como *gap*, mede a diferença percentual entre o *makespan* obtido pelo algoritmo HyPRR e pelo algoritmo GC. O seu valor é calculado conforme a Equação (4.1).

$$RPC = \frac{C_{HyPRR} - C_{GC}}{C_{GC}} \times 100 \tag{4.1}$$

Na sequência, são apresentados os resultados dos experimentos de calibração dos parâmetros populacionais do algoritmo HyPRR.

4.2 Seleção de vizinhanças e operador R&R

Conforme descrito na Seção 3.3.1, a busca local do HyPRR utiliza três estruturas de vizinhança: *swap*, *insertion* e *block insertion*. No entanto, é necessário determinar o tamanho do bloco de inserção que, em combinação com o operador R&R, resulte em soluções de melhor qualidade. Para isso, foi realizado um experimento utilizando as 15 instâncias mais desafiadoras, conforme listado na Tabela 4.1, a fim de identificar a melhor combinação dos tamanhos do *block insertion* e do operador R&R. A avaliação considerou a média dos *gaps*, o número de soluções inviáveis e o tempo médio de execução do algoritmo, considerando 10 execuções para cada instância do conjunto.

A fim de evitar viés de outros parâmetros, foi executada uma versão simplificada do algoritmo na qual foi mantido apenas o melhor indivíduo na população em qualquer momento, desconsiderando o procedimento de gerenciamento da população. Dessa forma, os fatores inerentes ao gerenciamento da população não impactaram na calibração dos tamanhos de bloco do R&R (d) e *block insertion* (l).

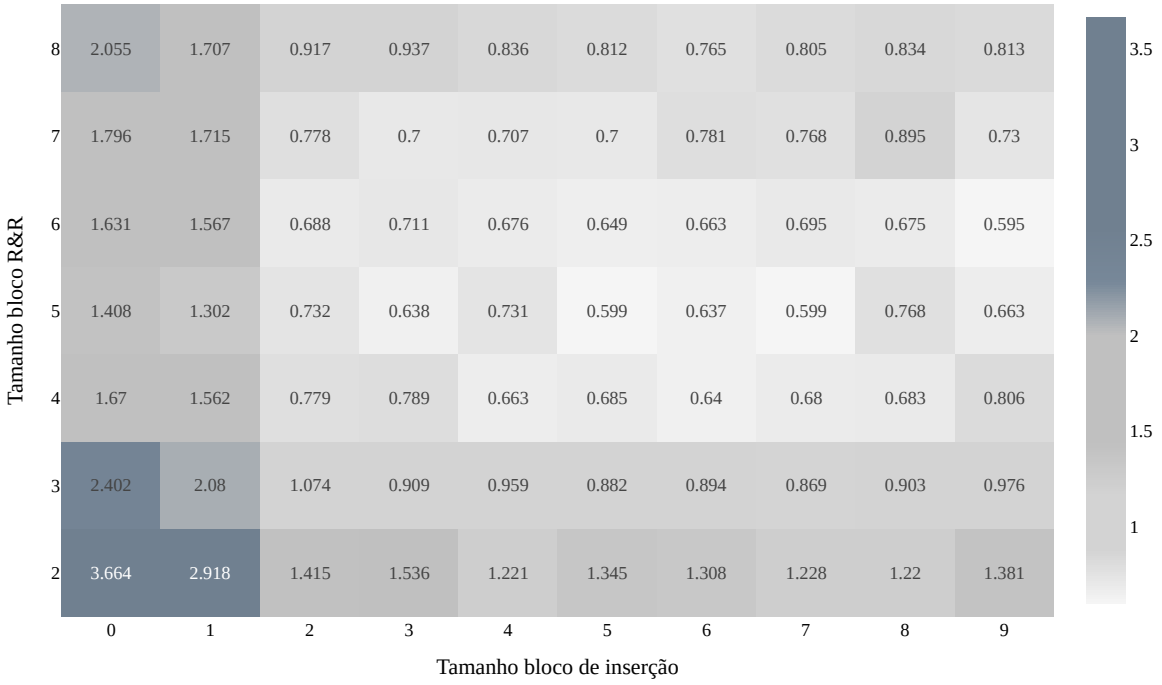
Ao todo, foram testadas 56 combinações de (d, l), em que $d \in \{2, 3, 4, 5, 6, 7, 8\}$ e $l \in \{2, 3, 4, 5, 6, 7, 8, 9\}$. Os resultados para cada combinação de tamanho de bloco R&R e *block insertion* são detalhados na Tabela 4.2 e ilustrados na Figura 4.1. Entre as combinações avaliadas, a configuração ($d = 5$, $l = 5$) apresentou os melhores resultados, considerando o *gap* médio, o número de soluções inviáveis e o tempo médio de execução. Além disso, verificou-se que blocos de R&R com tamanhos $d \leq 4$ resultaram nos maiores números de soluções inviáveis.

Tabela 4.2: *Gap* médio, número de soluções inviáveis e tempo médio em segundos por número de tarefas d removidas no R&R e tamanho de bloco l.

d	l	$gap(\%)$	Inviáveis	Tempo(s)	d	l	$Gap(\%)$	Inviáveis	Tempo(s)
2	2	1,42	16	1,23	5	6	0,64	3	4,40
2	3	1,54	11	1,61	5	7	0,60	3	5,00
2	4	1,22	14	1,81	5	8	0,77	1	5,04
2	5	1,35	9	2,03	5	9	0,66	1	5,54
2	6	1,31	10	2,41	6	2	0,69	2	3,26
2	7	1,23	9	2,67	6	3	0,71	0	3,79
2	8	1,22	6	3,02	6	4	0,68	1	4,10
2	9	1,38	14	3,30	6	5	0,65	0	4,92
3	2	1,07	8	1,65	6	6	0,66	0	5,12
3	3	0,91	3	1,96	6	7	0,70	0	5,50
3	4	0,96	7	2,35	6	8	0,68	3	5,83
3	5	0,88	4	2,69	6	9	0,60	1	6,57
3	6	0,89	6	2,88	7	2	0,78	0	3,79
3	7	0,87	4	3,32	7	3	0,70	0	4,03
3	8	0,90	7	3,52	7	4	0,71	0	4,87
3	9	0,98	6	3,97	7	5	0,70	0	5,50
4	2	0,78	4	2,19	7	6	0,78	0	5,52
4	3	0,79	4	2,58	7	7	0,77	1	6,09
4	4	0,66	3	3,04	7	8	0,90	1	6,46
4	5	0,69	2	3,45	7	9	0,73	0	7,41
4	6	0,64	4	3,82	8	2	0,92	0	4,27
4	7	0,68	2	4,08	8	3	0,94	0	4,64
4	8	0,68	4	4,30	8	4	0,84	0	5,49
4	9	0,80	0	4,88	8	5	0,81	0	5,99
5	2	0,73	2	2,51	8	6	0,77	0	6,69
5	3	0,64	1	3,09	8	7	0,81	0	7,36
5	4	0,73	2	3,28	8	8	0,83	0	7,95
5	**5**	**0,60**	**1**	**4,01**	8	9	0,81	0	8,44

Figura 4.1: *Heatmap* do *gap* médio obtido para cada tamanho de *block insertion* e R&R

.



Em seguida, na Seção 4.4, são apresentados os resultados do experimento que compara o desempenho do operador R&R com outros operadores amplamente discutidos na literatura.

4.3 Calibração de parâmetros populacionais

Conforme ilustrado no Algoritmo 1, existem quatro parâmetros populacionais, similares aos parâmetros do *hybrid genetic search* (HGS) (Mecler et al., 2021), que necessitam de calibração: o tamanho da população inicial (μ), o número máximo de novos indivíduos (λ), o número de indivíduos elite (μ^{elite}) e o número de indivíduos mais próximos usados no gerenciamento da diversidade (μ^{close}).

No trabalho de Mecler et al. (2021), os autores utilizaram a seguinte configuração de parâmetros: $\mu = 20$, $\lambda = 40$, $\mu^{elite} = 10$ e $\mu^{close} = 3$. Contudo, devido às particularidades do problema abordado neste trabalho, optou-se por recalibrar os parâmetros

populacionais do algoritmo, utilizando as 15 instâncias mais desafiadoras do conjunto. Os valores testados foram: $\mu = \{2, 4, 6, 8, 10, 20, 30\}$, $\lambda = \{1 \times \mu, 1,5 \times \mu, 2 \times \mu\}$, $\mu^{elite} = 0,5 \times \mu$ e $\mu^{close} = \lceil 0,2 \times \mu \rceil$.

No total, foram avaliadas 21 combinações de parâmetros populacionais. A Tabela 4.3 apresenta os resultados médios de *gap* percentual, tempos de execução médio em segundos e o número de soluções inviáveis obtidas para cada combinação de parâmetros. Para cada configuração, o HyPRR foi executado 10 vezes, e o RPC foi obtido para cada instância. Posteriormente, o *gap* médio (%) foi calculado com base nos resultados das 15 instâncias testadas.

Tabela 4.3: Variando os parâmetros populacionais do HyPRR ($d = 5, l = 5$).

μ	λ	μ^{elite}	μ^{close}	$Gap_{médio}(\%)$	Tempo(s)	Inviáveis
2	2	1	1	1,82	4,88	2
2	3	1	1	1,75	4,70	2
2	4	1	1	1,92	5,10	2
4	4	2	2	0,55	0,75	0
4	6	2	1	0,54	0,76	0
4	8	2	2	0,53	0,79	0
6	6	3	2	0,48	0,64	0
6	9	3	2	0,48	0,62	0
6	12	3	2	0,49	0,64	0
8	8	4	2	0,56	0,76	0
8	**12**	**4**	**2**	**0,47**	**0,60**	**0**
8	16	4	2	0,61	0,82	0
10	10	5	2	0,59	0,75	0
10	15	5	2	0,56	0,77	0
10	20	5	2	0,54	0,69	0
20	20	10	4	0,76	0,93	0
20	30	10	4	0,79	0,95	0
20	40	10	4	0,79	1,04	0
30	30	15	6	0,90	1,08	0
30	45	15	6	1,01	1,14	0
30	60	15	6	0,99	1,18	0

Verificou-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. Além disso, o conjunto de parâmetros que obteve os menores valores de *gap* médio e tempo de execução foi a combinação $\mu = 8$, $\lambda = 12$, $\mu^{elite} = 4$, $\mu^{close} = 2$. Observou-se também que, à medida que os parâmetros populacionais aumentam, tanto o *gap* quanto

o tempo de execução também tendem a crescer.

Com base nesses resultados, optou-se por adotar uma configuração de 8 indivíduos na população inicial, com 12 indivíduos excedentes, 4 indivíduos elite e os 2 indivíduos mais próximos para o gerenciamento da diversidade no algoritmo HyPRR aplicado ao PSDLI.

4.4 Comparação com operadores *crossover*

Para avaliar a efetividade do algoritmo HyPRR, foi conduzido um experimento considerando todo o conjunto de instâncias, comparando o operador R&R com outros operadores estabelecidos na literatura. Entre eles, destacam-se o *partially mapped crossover* (PMX), desenvolvido por Goldberg e Lingle (2014), o *cycle crossover* (CX), proposto por Grefenstette (2013) e o *order crossover* (OX), implementado por Davis et al. (1985) no algoritmo HGS. Adicionalmente, foi avaliado o desempenho do HyPRR sem controle de diversidade na população, bem como a sua performance em uma configuração com indivíduo único, ou seja, sem população.

Esses operadores *crossover* e as diferentes versões do algoritmo HyPRR foram implementados. Os resultados apresentados na Tabela 4.4 indicam que as duas estratégias mais simples, ou seja, o uso de um indivíduo único e a ausência de controle de diversidade, são eficazes para instâncias $n < 40$. No entanto, para instâncias com mais de 50 tarefas, o operador R&R sobressai-se em relação aos demais operadores avaliados, demonstrando sua superioridade em instâncias maiores.

4.5 Comparação com a literatura

Os resultados obtidos pelo HyPRR no conjunto de 960 instâncias foram comparados aos do algoritmo GC (Davari et al., 2020). O HyPRR conseguiu obter melhores soluções em 21 instâncias, apresentou 927 empates e produziu resultados inferiores em apenas 12 instâncias quando comparado ao GC. Considerando o *gap* percentual médio, o HyPRR demonstrou vantagem sobre o GC a partir das instâncias com tamanho $n = 50$, como ilustrado na Tabela 4.5.

Tabela 4.4: *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/ Div.		Ind. Único	
	Gap	Tempo	*Gap*	Tempo	*Gap*	Tempo	*Gap*	Tempo	*Gap*	Tempo	*Gap*	Tempo
$n = 10$	**0,00**	**0,01**	0,01	0,01	0,01	0,01	0,04	0,01	**0,00**	**0,01**	**0,00**	**0,00**
$n = 20$	**0,00**	**0,02**	0,01	0,03	0,03	0,02	0,06	0,02	**0,00**	**0,03**	**0,00**	**0,02**
$n = 30$	0,00	0,08	0,02	0,12	0,02	0,08	0,06	0,06	0,01	0,09	**0,00**	**0,08**
$n = 40$	0,03	0,22	0,08	0,33	0,03	0,21	0,17	0,17	**0,02**	**0,22**	0,05	0,20
$n = 50$	**0,02**	**0,47**	0,07	0,78	0,03	0,45	0,19	0,36	0,02	0,47	0,02	0,43
$n = 60$	**0,03**	**0,98**	0,14	1,67	0,05	0,99	0,24	0,73	0,04	0,97	0,05	0,92
$n = 70$	**0,04**	**1,46**	0,14	2,75	0,06	1,55	0,22	1,24	0,05	1,45	0,05	1,36
$n = 80$	**0,02**	**2,27**	0,07	4,46	0,03	2,40	0,18	1,89	0,02	2,22	0,03	2,09
$n = 90$	**0,04**	**3,54**	0,11	7,16	**0,04**	**3,72**	0,31	2,98	0,05	3,48	0,05	3,20
$n = 100$	**0,06**	**5,26**	0,18	10,61	0,07	5,80	0,33	4,62	0,08	5,13	0,08	4,47

Tabela 4.5: *Gap* médio (%), vitórias, empates e derrotas na qualidade de soluções do HyPRR comparado ao GC.

Instâncias	$Gap_{médio}(\%)$	Vitórias	Empates	Derrotas
$n = 10$	$0,00$	0	96	0
$n = 20$	$0,00$	0	96	0
$n = 30$	$0,00$	0	96	0
$n = 40$	$0,00$	1	94	1
$n = 50$	$-0,01$	1	95	0
$n = 60$	$0,00$	2	93	1
$n = 70$	$0,00$	4	91	1
$n = 80$	$-0,01$	3	92	1
$n = 90$	$-0,02$	5	88	3
$n = 100$	$0,00$	5	86	5

Os tempos de execução dos dois algoritmos são apresentados na Tabela 4.6. O GC foi executado em um computador com processador Intel®Core™; i7-3720QM de 2.6 GHz e 8 GB de RAM, enquanto o HyPRR foi testado em um Intel®Core™; i7-12700 com 4.90 GHz e 16 GB de RAM. A diferença na capacidade de processamento entre as máquinas foi compensada ajustando o tempo reportado do GC pela divisão por 2,096, de acordo com o comparativo em CPU Benchmarks (`https://www.cpubenchmark.net/compare/4669vs895/Intel-i7-12700-vs-Intel-i7-3720QM`).

A Tabela 4.6 apresenta as médias dos tempos de execução, em segundos, para ambos os algoritmos. Devido ao fato do tempo de execução do GC em diversas instâncias ter alcançado o critério de parada de 1000 segundos, optou-se por considerar somente as instâncias resolvidas pelo GC antes do critério de parada do algoritmo. Mesmo assim, observa-se que, a partir de $n \geq 30$, o HyPRR se torna drasticamente mais rápido do que o GC.

Tabela 4.6: 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

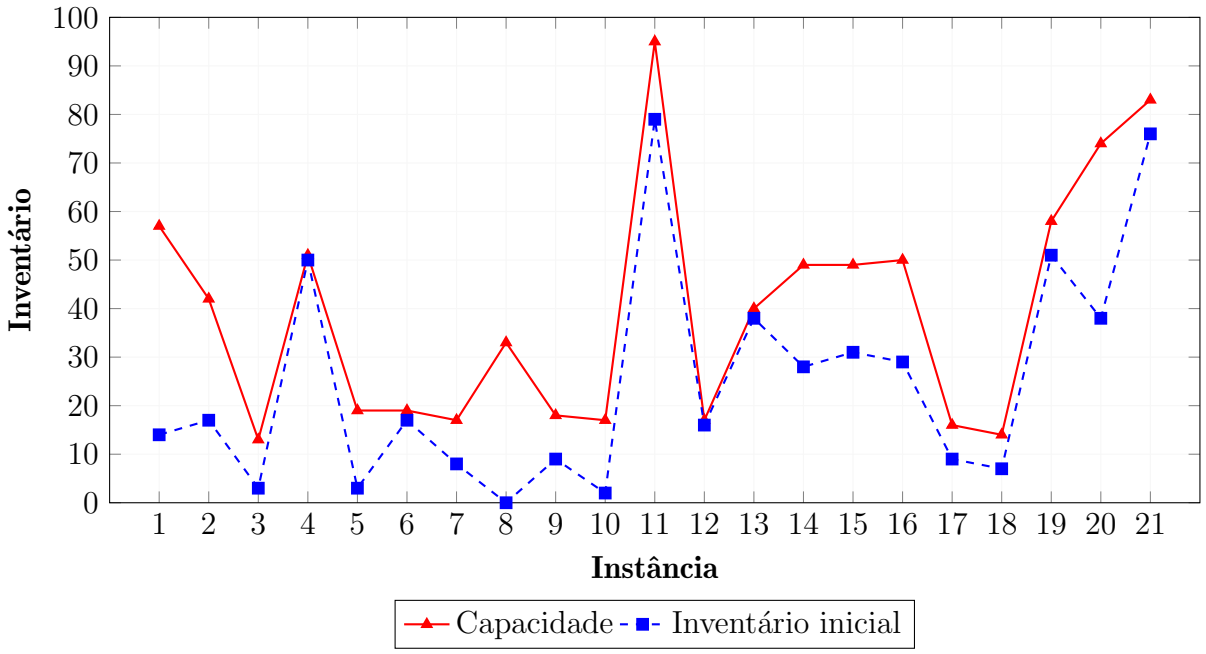
A Tabela 4.7 detalha os resultados para as instâncias em que o HyPRR e o GC divergiram no valor do *makespan* encontrado. Em muitos dos casos onde o HyPRR superou o GC, o tempo de execução do GC alcançou o critério de parada de 1000 segundos, enquanto o HyPRR conseguiu encontrar soluções melhores em menos tempo. Devido à necessidade de conversão dos tempos de execução do GC para comparação com o HyPRR, as instâncias que alcançaram o critério de parada (It_{NI}) estão com os tempos na ordem de 400 segundos.

Tabela 4.7: Resultados do *makespan*, tempo de execução e *gap* das instâncias com vitória ou derrota do HyPRR comparado ao GC

Instância	*Makespan*				Tempo(s)				*GAP*
	HyPRR		GC		HyPRR		GC		
	Best	*Avg.*	*Best*	*Avg.*	*Best*	*Avg.*	*Best*	*Avg.*	
1 - data40_68	**3481**	3481	3487	-	0,24	0,25	187,25	-	-0,17%
2 - data50_41	**2693**	2693	2722	-	0,54	0,56	477,71	-	-1,07%
3 - data60_13	**3483**	3483	3491	-	0,73	0,82	3,73	-	-0,23%
4 - data60_43	**3271**	3271	3282	-	1,08	1,11	477,94	-	-0,34%
5 - data70_15	**3737**	3737	3744	-	1,10	1,21	482,73	-	-0,19%
6 - data70_38	**3772**	3772,7	3794	-	2,15	2,59	478,09	-	-0,58%
7 - data70_40	**3760**	3761,2	3763	-	2,32	3,06	484,09	-	-0,08%
8 - data70_43	**4251**	4251	4261	-	1,79	2,03	478,58	-	-0,23%
9 - data80_26	**377**	379,3	378	-	2,33	3,75	478,63	-	-0,26%
10 - data80_39	**4118**	4124,6	4120	-	4,44	6,50	482,57	-	-0,05%
11 - data80_46	**4316**	4316	4336	-	2,02	2,13	478,43	-	-0,46%
12 - data90_40	**5520**	5520	5583	-	8,19	10,61	479,15	-	-1,13%
13 - data90_41	**4563**	4563	4570	-	4,55	4,91	483,23	-	-0,15%
14 - data90_42	**4969**	4969	4994	-	3,69	3,86	480,50	-	-0,50%
15 - data90_43	**4625**	4625	4634	-	3,21	3,32	478,74	-	-0,19%
16 - data90_48	**3881**	3881	3903	-	3,73	4,12	478,67	-	-0,56%
17 - data100_13	**4617**	4617	4625	-	4,56	5,06	479,48	-	-0,17%
18 - data100_39	**5980**	5981,6	5989	-	14,29	17,38	477,65	-	-0,15%
19 - data100_44	**4823**	4823	4878	-	5,30	5,69	479,74	-	-1,13%
20 - data100_45	**4300**	4300	4309	-	5,37	5,65	480,69	-	-0,21%
21 - data100_47	**5437**	5437	5445	-	6,21	6,64	482,73	-	-0,15%
22 - data40_52	324	324	**323**	-	0,35	0,42	0,05	-	0,31%
23 - data60_25	334	340	**333**	-	1,73	2,92	0,04	-	0,30%
24 - data70_28	367	371,7	**364**	-	3,12	4,45	70,79	-	0,82%
25 - data80_4	396	396	**395**	-	2,23	2,82	2,17	-	0,25%
26 - data90_25	475	478	**473**	-	5,95	10,05	481,57	-	0,42%
27 - data90_26	556	558,6	**554**	-	6,25	10,23	478,69	-	0,36%
28 - data90_28	500	505	**499**	-	5,76	10,92	52,62	-	0,20%
29 - data100_25	512	520	**510**	-	8,73	14,71	0,05	-	0,39%
30 - data100_26	501	502,4	**500**	-	8,15	12,61	49,80	-	0,20%
31 - data100_27	505	511,3	**503**	-	8,35	14,88	477,34	-	0,40%
32 - data100_28	589	593	**587**	-	9,34	17,92	0,79	-	0,34%
33 - data100_51	728	733,3	**727**	-	9,34	12,70	477,99	-	0,14%

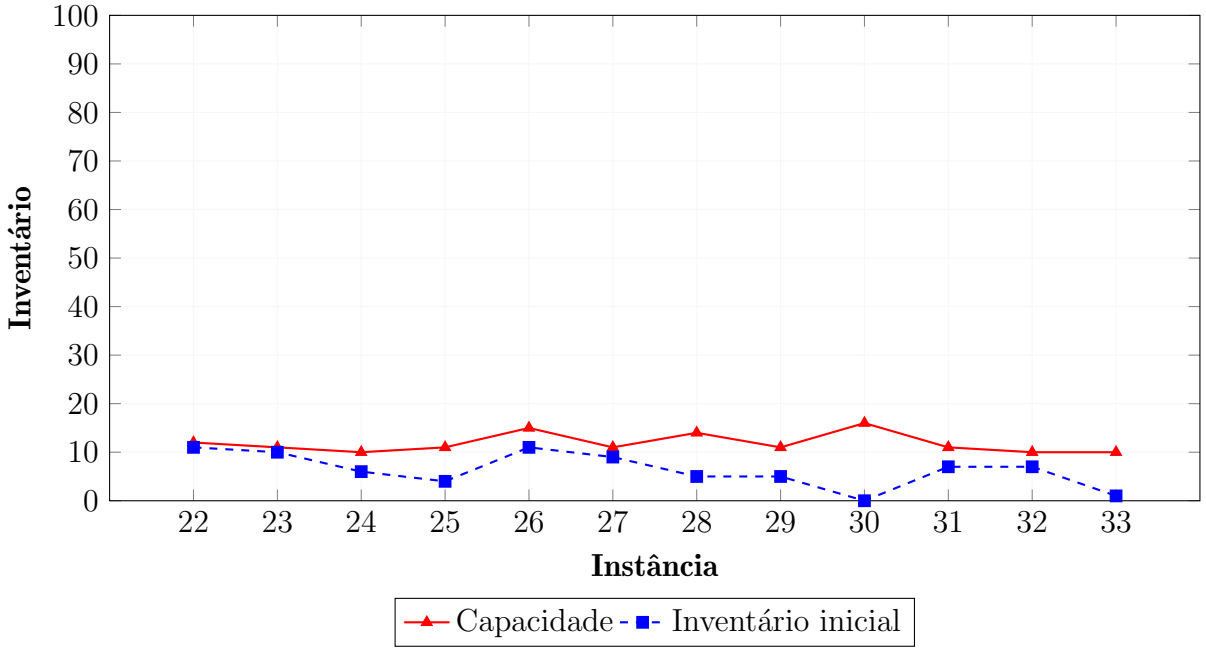
Embora o HyPRR tenha obtido resultados inferiores ao GC em 12 instâncias, as Figuras 4.2 e 4.3 mostram que há diferenças notáveis nas características de inventário entre os grupos de instâncias onde o HyPRR superou e foi superado pelo GC. O grupo de instâncias de 1 a 21, ilustrado na Figura 4.2, apresenta as capacidades de inventário variando entre 20 e 95.

Figura 4.2: Dados de Inventário inicial e capacidade no grupo de instâncias em que HyPRR superou o GC



Já no grupo de instâncias em que o HyPRR foi superado pelo GC, conforme ilustrado na Figura 4.3, todas as instâncias possuem capacidade de inventário inferior a 20. Essa observação sugere que há oportunidades para incorporar outras técnicas de perturbação de soluções, capazes de lidar melhor com situações de alta restrição de capacidade de inventário. Apesar disso, destaca-se a capacidade do HyPRR em obter soluções iguais ou superiores ao GC em um tempo muito menor para instâncias maiores ($n \geq 90$).

Figura 4.3: Dados de Inventário inicial e capacidade no grupo de instâncias em que o GC superou o HyPRR



Capítulo 5

Considerações Finais

Neste trabalho, foi proposta uma abordagem heurística baseada em busca populacional híbrida para o problema de minimização do *makespan* no sequenciamento em ambiente de máquina única, com datas de liberação e restrições de inventário, denotado, na notação de três campos como $1|r_j, inv|C_{max}$. O método proposto, denominado HyPRR, utiliza procedimentos de intensificação e diversificação. A intensificação envolve uma busca local com avaliação eficiente de movimentos, enquanto a diversificação emprega o operador *ruin-and-recreate* para que o algoritmo possa escapar de ótimos locais e encontrar soluções melhores.

Foram utilizadas 960 instâncias propostas por Davari et al. (2020) para avaliar os resultados do algoritmo. Dentre essas, foram selecionadas as 15 instâncias mais desafiadoras para a calibração dos parâmetros populacionais e dos tamanhos dos blocos de R&R e *block insertion*. Para validar a efetividade do algoritmo para o PSDLI, foi realizado outro experimento comparativo entre o HyPRR, diferentes operadores encontrados na literatura, bem como duas versões do algoritmo sem população e com indivíduo único. Com a efetividade do algoritmo validada, foi realizado um experimento final em todo o conjunto de instâncias para comparar o HyPRR com o algoritmo GC presente na literatura.

Os resultados mostraram que o HyPRR superou o GC em 21 instâncias, obteve os mesmos resultados em 927 casos e foi superado em apenas 12 instâncias. As instâncias

em que o HyPRR foi superado pelo GC apresentam maior restrição de capacidade de inventário. Apesar disso, o HyPRR foi substancialmente mais rápido do que o GC na resolução de instâncias maiores, a partir de $n \geq 30$.

Para trabalhos futuros, sugere-se a avaliação do algoritmo HyPRR em instâncias maiores ($n > 100$), além do desenvolvimento de métodos que possam melhorar a qualidade das soluções em instâncias com grandes restrições de capacidade de inventário. Outra possibilidade é investigar procedimentos adicionais que auxiliem o R&R a escapar de ótimos locais.

REFERÊNCIAS BIBLIOGRÁFICAS

AVDEENKO, T. V.; MEZENTSEV, Y. A.; ESTRAYKH, I. V. Heuristic approach to unrelated parallel machines scheduling under availability and resource constraints. *IFAC-PapersOnLine*, v. 50, n. 1, p. 13096–13101, 2017.

BARTELS, J.-H.; ZIMMERMANN, J. Scheduling tests in automotive r&d projects. *European Journal of Operational Research*, v. 193, n. 3, p. 805–819. ISSN 0377-2217, 2009.

BARTELS, J.-H.; GATHER, T.; ZIMMERMANN, J. Dismantling of nuclear power plants at optimal npv. *Annals of Operations Research*, v. 186, p. 407–427, 2011.

BARTELS, J.-H.; ZIMMERMANN, J. Scheduling tests in automotive r&d projects using a genetic algorithm. *Handbook on Project Management and Scheduling Vol. 2*, v. 193, p. 1157–1185, 2015.

BAZGOSHA, A.; RANJBAR, M.; JAMILI, N. Scheduling of loading and unloading operations in a multi stations transshipment terminal with release date and inventory constraints. *Computers & Industrial Engineering*, v. 106, p. 20–31. ISSN 0360-8352, 2017.

BLUM, C.; ROLI, A. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Comput. Surv.*, v. 35, n. 3, p. 268–308. ISSN 0360-0300. doi: 10.1145/937503.937505, 2003.

BLUM, C.; ROLI, A. *Hybrid Metaheuristics: An Introduction*, p. 1–30. Springer Berlin Heidelberg, Berlin, Heidelberg. ISBN 978-3-540-78295-7. doi: 10.1007/978-3-540-78295-7_1, 2008.

BRISKORN, D.; LEUNG, J. Y.-T. Minimizing maximum lateness of jobs in inventory constrained scheduling. *Journal of the Operational Research Society*, v. 64, n. 12, p. 1851–1864, 2013.

BRISKORN, D.; CHOI, B.-C.; LEE, K.; LEUNG, J.; PINEDO, M. Genetic algorithms for inventory constrained scheduling on a single machine. Technical report, Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel, 2009.

BRISKORN, D.; CHOI, B.-C.; LEE, K.; LEUNG, J.; PINEDO, M. Complexity of single machine scheduling subject to nonnegative inventory constraints. *European Journal of Operational Research*, v. 207, n. 2, p. 605–619, 2010.

BRISKORN, D.; JAEHN, F.; PESCH, E. Exact algorithms for inventory constrained scheduling on a single machine. *Journal of scheduling*, v. 16, p. 105–115, 2013.

BRISKORN, D.; PESCH, E. Variable very large neighbourhood algorithms for truck sequencing at transshipment terminals. *International Journal of Production Research*, v. 51, n. 23-24, p. 7140–7155, 2013.

BULHÕES, T.; SUBRAMANIAN, A.; ERDOĞAN, G.; LAPORTE, G. The static bike relocation problem with multiple vehicles and visits. *European Journal of Operational Research*, v. 264, n. 2, p. 508–523. doi: https://doi.org/10.1016/j.ejor.2017.06.028, 2018.

CAPPADONNA, F.; COSTA, A.; FICHERA, S. Makespan minimization of unrelated parallel machines with limited human resources. *Procedia Cirp*, v. 12, p. 450–455, 2013.

CARLIER, J.; KAN, A. R. Scheduling subject to nonrenewable-resource constraints. *Operations Research Letters*, v. 1, n. 2, p. 52–55, 1982.

CHEN, J.-F.; WU, T.-H. Total tardiness minimization on unrelated parallel machine scheduling with auxiliary equipment constraints. *Omega*, v. 34, n. 1, p. 81–89, 2006.

DAVARI, M.; RANJBAR, M.; DE CAUSMAECKER, P.; LEUS, R. Minimizing makespan on a single machine with release dates and inventory constraints. *European Journal of Operational Research*, v. 286, n. 1, p. 115–128. ISSN 0377-2217, 2020.

DAVIS, L. ET AL. Applying adaptive algorithms to epistatic domains. *IJCAI*, volume 85, p. 162–164. Citeseer, 1985.

FANJUL-PEYRO, L.; PEREA, F.; RUIZ, R. Models and matheuristics for the unrelated parallel machine scheduling problem with additional resources. *European Journal of Operational Research*, v. 260, n. 2, p. 482–493. ISSN 0377-2217, 2017.

FONSECA, G. B.; NOGUEIRA, T. H.; RAVETTI, M. G. A hybrid lagrangian metaheuristic for the cross-docking flow shop scheduling problem. *European Journal of Operational Research*, v. 275, n. 1, p. 139–154, 2019.

GHORBANZADEH, M.; RANJBAR, M.; JAMILI, N. Transshipment scheduling at a single station with release date and inventory constraints. *Journal of Industrial and Production Engineering*, v. 36, n. 5, p. 301–312, 2019.

GOLDBERG, D. E.; LINGLE, R. Alleles, loci, and the traveling salesman problem. *Proceedings of the first international conference on genetic algorithms and their applications*, p. 154–159. Psychology Press, 2014.

GRAHAM, R. L.; LAWLER, E. L.; LENSTRA, J. K.; KAN, A. R. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of discrete mathematics*, volume 5, p. 287–326. Elsevier, 1979.

GREFENSTETTE, J. J. *Genetic algorithms and their applications: proceedings of the second international conference on genetic algorithms*. Psychology Press, 2013.

HAN, X.; ZHAO, P.; MENG, Q.; YIN, S.; WAN, D. Optimal scheduling of airport ferry vehicles based on capacity network. *Annals of Operations Research*, v. 295, p. 163–182, 2020.

HOUSSEIN, E. H.; GAD, A. G.; WAZERY, Y. M.; SUGANTHAN, P. N. Task scheduling in cloud computing based on meta-heuristics: Review, taxonomy, open challenges, and future trends. *Swarm and Evolutionary Computation*, v. 62, p. 100841. ISSN 2210-6502. doi: https://doi.org/10.1016/j.swevo.2021.100841. URL `https://www.sciencedirect.com/science/article/pii/S221065022100002X`, 2021.

LEE, Y. H.; PINEDO, M. Scheduling jobs on parallel machines with sequence-dependent setup times. *European Journal of Operational Research*, v. 100, n. 3, p. 464–474, 1997.

MECLER, J.; SUBRAMANIAN, A.; VIDAL, T. A simple and effective hybrid genetic search for the job sequencing and tool switching problem. *Computers & Operations Research*, v. 127, p. 105153, 2021.

MORAIS, R.; BULHÕES, T.; SUBRAMANIAN, A. 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*, v. 315, n. 2, p. 442–453. doi: https://doi.org/10.1016/j.ejor.2023.11.024, 2024.

MORSY, E.; PESCH, E. Approximation algorithms for inventory constrained scheduling on a single machine. *Journal of Scheduling*, v. 18, p. 645–653, 2015.

NEUMANN, K.; ZIMMERMANN, J. Procedures for resource leveling and net present value problems in project scheduling with general temporal and resource constraints. *European Journal of Operational Research*, v. 127, n. 2, p. 425–443. ISSN 0377-2217, 2000.

NEUMANN, K.; SCHWINDT, C. Project scheduling with inventory constraints. *Mathematical Methods of Operations Research*, v. 56, p. 513–533, 2003.

PINEDO, M. *Scheduling: Theory, Algorithms, and Systems*. Springer Publishing Company, Incorporated, 6th edição. ISBN 978-3-031-05920-9. doi: 10.1007/978-3-031-05921-6, 2022.

PINOL, H.; BEASLEY, J. Scatter search and bionomic algorithms for the aircraft landing problem. *European Journal of Operational Research*, v. 171, n. 2, p. 439–462, 2006.

RANJBAR, M.; SABER, R. G. A variable neighborhood search algorithm for transshipment scheduling of multi products at a single station. *Applied Soft Computing*, v. 98, p. 106736, 2021.

RUIZ, R.; STÜTZLE, T. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, v. 177, n. 3, p. 2033–2049. ISSN 0377-2217, 2007.

SCHRIMPF, G.; SCHNEIDER, J.; STAMM-WILBRANDT, H.; DUECK, G. Record breaking optimization results using the ruin and recreate principle. *Journal of Computational Physics*, v. 159, n. 2, p. 139–171, 2000.

SOARES, L. C.; CARVALHO, M. A. Application of a hybrid evolutionary algorithm to resource-constrained parallel machine scheduling with setup times. *Computers & Operations Research*, v. 139, p. 105637, 2022.

SUBRAMANIAN, A.; DRUMMOND, L. M.; BENTES, C.; OCHI, L. S.; FARIAS, R. A parallel heuristic for the vehicle routing problem with simultaneous pickup and delivery. *Computers & Operations Research*, v. 37, n. 11, p. 1899–1911, 2010.

TALBI, E.-G. A taxonomy of hybrid metaheuristics. *Journal of heuristics*, v. 8, p. 541–564, 2002.

TAMAKI, H.; HASEGAWA, Y.; KOZASA, J.; ARAKI, M. Application of search methods to scheduling problem in plastics forming plant: A binary representation approach. *Proceedings of 32nd IEEE conference on decision and control*, p. 3845–3850. IEEE, 1993.

TEIXEIRA, E.; SUBRAMANIAN, A.; KRAMER, H. H. Busca populacional híbrida para o problema de flow shop com bloqueio e minimização do makespan. *Anais do Simpósio Brasileiro de Pesquisa Operacional*, São José dos Campos. Galoá, 2023.

TORABI, S. A.; SAHEBJAMNIA, N.; MANSOURI, S. A.; BAJESTANI, M. A. A particle swarm optimization for a fuzzy multi-objective unrelated parallel machines scheduling problem. *Applied Soft Computing*, v. 13, n. 12, p. 4750–4762, 2013.

VIDAL, T.; CRAINIC, T. G.; GENDREAU, M.; LAHRICHI, N.; REI, W. A hybrid genetic algorithm for multidepot and periodic vehicle routing problems. *Operations Research*, v. 60, n. 3, p. 611–624, 2012.

YEPES-BORRERO, J. C.; VILLA, F.; PEREA, F.; CABALLERO-VILLALOBOS, J. P. Grasp algorithm for the unrelated parallel machine scheduling problem with setup times and additional resources. *Expert Systems with Applications*, v. 141, p. 112959, 2020.

YU, W.; EGBELU, P. J. Scheduling of inbound and outbound trucks in cross docking systems with temporary storage. *European Journal of Operational Research*, v. 184, n. 1, p. 377–396. ISSN 0377-2217, 2008.

Apêndice A

Resultados detalhados

Instância	*Makespan*				*CPU* (s)			
	HyPRR		*GC*		*HyPRR*		*GC*	
	Best	*Avg.*	*Best*	*Avg.*	*Best*	*Avg.*	*Best*	*Avg.*
data10_1	60	60	60	-	0,006	0,010	0,000	-
data10_2	55	55	55	-	0,002	0,006	0,000	-
data10_3	58	58	58	-	0,003	0,009	0,000	-
data10_4	41	41	41	-	0,004	0,008	0,000	-
data10_5	69	69	69	-	0,000	0,007	0,000	-
data10_6	55	55	55	-	0,006	0,008	0,000	-
data10_7	56	56	56	-	0,002	0,002	0,000	-
data10_8	72	72	72	-	0,004	0,007	0,000	-
data10_9	48	48	48	-	0,001	0,002	0,000	-
data10_10	73	73	73	-	0,002	0,002	0,000	-
data10_11	48	48	48	-	0,002	0,003	0,000	-
data10_12	51	51	51	-	0,003	0,004	0,000	-
data10_13	525	525	525	-	0,006	0,009	0,010	-
data10_14	573	573	573	-	0,002	0,004	0,000	-
data10_15	460	460	460	-	0,004	0,008	0,000	-
data10_16	484	484	484	-	0,002	0,003	0,000	-

Instância	*Makespan*				*CPU* (s)			
	HyPRR		*GC*		*HyPRR*		*GC*	
	Best	*Avg.*	*Best*	*Avg.*	*Best*	*Avg.*	*Best*	*Avg.*
data10_17	501	501	501	-	0,003	0,006	0,010	-
data10_18	568	568	568	-	0,006	0,008	0,010	-
data10_19	545	545	545	-	0,002	0,004	0,020	-
data10_20	640	640	640	-	0,004	0,007	0,010	-
data10_21	502	502	502	-	0,004	0,007	0,020	-
data10_22	614	614	614	-	0,004	0,008	0,040	-
data10_23	598	598	598	-	0,002	0,003	0,010	-
data10_24	521	521	521	-	0,005	0,008	0,010	-
data10_25	53	53	53	-	0,005	0,009	0,000	-
data10_26	59	59	59	-	0,002	0,004	0,000	-
data10_27	57	57	57	-	0,004	0,009	0,000	-
data10_28	50	50	50	-	0,002	0,004	0,000	-
data10_29	61	61	61	-	0,002	0,002	0,000	-
data10_30	51	51	51	-	0,002	0,005	0,000	-
data10_31	52	52	52	-	0,004	0,008	0,000	-
data10_32	49	49	49	-	0,002	0,004	0,000	-
data10_33	72	72	72	-	0,005	0,008	0,000	-
data10_34	67	67	67	-	0,003	0,007	0,000	-
data10_35	46	46	46	-	0,008	0,009	0,000	-
data10_36	56	56	56	-	0,004	0,010	0,000	-
data10_37	578	578	578	-	0,003	0,009	0,000	-
data10_38	558	558	558	-	0,004	0,008	0,000	-
data10_39	672	672	672	-	0,003	0,008	0,000	-
data10_40	749	749	749	-	0,003	0,009	0,000	-
data10_41	571	571	571	-	0,002	0,002	0,020	-
data10_42	649	649	649	-	0,003	0,008	0,000	-
data10_43	428	428	428	-	0,002	0,005	0,010	-
data10_44	655	655	655	-	0,004	0,009	0,020	-
data10_45	726	726	726	-	0,003	0,008	0,000	-

Instância	*Makespan*				*CPU* (s)			
	HyPRR		*GC*		*HyPRR*		*GC*	
	Best	*Avg.*	*Best*	*Avg.*	*Best*	*Avg.*	*Best*	*Avg.*
data10_46	506	506	506	-	0,006	0,009	0,010	-
data10_47	472	472	472	-	0,009	0,009	0,010	-
data10_48	464	464	464	-	0,007	0,010	0,010	-
data10_49	57	57	57	-	0,004	0,012	0,000	-
data10_50	72	72	72	-	0,003	0,009	0,000	-
data10_51	73	73	73	-	0,002	0,003	0,000	-
data10_52	83	83	83	-	0,004	0,008	0,000	-
data10_53	82	82	82	-	0,003	0,007	0,000	-
data10_54	95	95	95	-	0,002	0,003	0,000	-
data10_55	78	78	78	-	0,004	0,008	0,000	-
data10_56	57	57	57	-	0,003	0,008	0,000	-
data10_57	72	72	72	-	0,004	0,008	0,000	-
data10_58	71	71	71	-	0,002	0,008	0,000	-
data10_59	91	91	91	-	0,003	0,009	0,000	-
data10_60	94	94	94	-	0,002	0,004	0,000	-
data10_61	713	713	713	-	0,004	0,008	0,000	-
data10_62	746	746	746	-	0,003	0,008	0,000	-
data10_63	997	997	997	-	0,003	0,003	0,000	-
data10_64	721	721	721	-	0,003	0,003	0,000	-
data10_65	858	858	858	-	0,003	0,007	0,000	-
data10_66	897	897	897	-	0,003	0,008	0,010	-
data10_67	702	702	702	-	0,003	0,008	0,000	-
data10_68	724	724	724	-	0,002	0,003	0,000	-
data10_69	1177	1177	1177	-	0,002	0,006	0,010	-
data10_70	1074	1074	1074	-	0,004	0,008	0,010	-
data10_71	756	756	756	-	0,003	0,008	0,020	-
data10_72	599	599	599	-	0,002	0,003	0,010	-
data10_73	137	137	137	-	0,004	0,008	0,000	-
data10_74	114	114	114	-	0,003	0,008	0,000	-

Instância	*Makespan*				*CPU* (s)			
	HyPRR		*GC*		*HyPRR*		*GC*	
	Best	*Avg.*	*Best*	*Avg.*	*Best*	*Avg.*	*Best*	*Avg.*
data10_75	59	59	59	-	0,002	0,003	0,000	-
data10_76	64	64	64	-	0,004	0,008	0,000	-
data10_77	83	83	83	-	0,002	0,003	0,000	-
data10_78	100	100	100	-	0,002	0,007	0,000	-
data10_79	84	84	84	-	0,004	0,008	0,000	-
data10_80	109	109	109	-	0,003	0,007	0,000	-
data10_81	100	100	100	-	0,005	0,009	0,000	-
data10_82	100	100	100	-	0,004	0,010	0,000	-
data10_83	66	66	66	-	0,005	0,009	0,000	-
data10_84	88	88	88	-	0,002	0,003	0,000	-
data10_85	1197	1197	1197	-	0,002	0,003	0,010	-
data10_86	717	717	717	-	0,002	0,003	0,010	-
data10_87	1087	1087	1087	-	0,003	0,006	0,000	-
data10_88	1087	1087	1087	-	0,004	0,008	0,000	-
data10_89	1304	1304	1304	-	0,004	0,007	0,010	-
data10_90	818	818	818	-	0,003	0,008	0,010	-
data10_91	889	889	889	-	0,003	0,008	0,000	-
data10_92	878	878	878	-	0,002	0,003	0,010	-
data10_93	1393	1393	1393	-	0,005	0,010	0,010	-
data10_94	662	662	662	-	0,003	0,008	0,010	-
data10_95	998	998	998	-	0,002	0,003	0,020	-
data10_96	648	648	648	-	0,002	0,002	0,010	-
data20_1	89	89	89	-	0,020	0,023	0,010	-
data20_2	136	136	136	-	0,019	0,027	0,150	-
data20_3	100	100	100	-	0,019	0,023	0,140	-
data20_4	103	103	103	-	0,019	0,019	0,080	-
data20_5	127	127	127	-	0,014	0,021	0,050	-
data20_6	103	103	103	-	0,015	0,018	0,140	-
data20_7	89	89	89	-	0,015	0,015	0,070	-

Instância	*Makespan*				*CPU* (s)			
	HyPRR		*GC*		*HyPRR*		*GC*	
	Best	*Avg.*	*Best*	*Avg.*	*Best*	*Avg.*	*Best*	*Avg.*
data20_8	113	113	113	-	0,015	0,015	0,180	-
data20_9	118	118	118	-	0,013	0,020	0,150	-
data20_10	96	96	96	-	0,013	0,013	0,130	-
data20_11	103	103	103	-	0,016	0,016	0,070	-
data20_12	110	110	110	-	0,013	0,013	0,140	-
data20_13	1005	1005	1005	-	0,017	0,017	0,060	-
data20_14	992	992	992	-	0,016	0,016	0,030	-
data20_15	1337	1337	1337	-	0,024	0,026	0,020	-
data20_16	846	846	846	-	0,024	0,029	0,040	-
data20_17	865	865	865	-	0,014	0,014	0,180	-
data20_18	833	833	833	-	0,014	0,014	0,200	-
data20_19	1201	1201	1201	-	0,014	0,014	0,230	-
data20_20	955	955	955	-	0,014	0,023	0,090	-
data20_21	1042	1042	1042	-	0,013	0,022	0,260	-
data20_22	1033	1033	1033	-	0,014	0,018	0,220	-
data20_23	1004	1004	1004	-	0,009	0,021	0,250	-
data20_24	1005	1005	1005	-	0,014	0,018	0,390	-
data20_25	151	151	151	-	0,021	0,029	0,000	-
data20_26	135	135	135	-	0,028	0,033	0,000	-
data20_27	129	129	129	-	0,020	0,021	0,020	-
data20_28	132	132	132	-	0,022	0,025	0,010	-
data20_29	106	106	106	-	0,020	0,027	0,010	-
data20_30	141	141	141	-	0,019	0,020	0,010	-
data20_31	118	118	118	-	0,020	0,025	0,010	-
data20_32	119	119	119	-	0,016	0,023	0,010	-
data20_33	96	96	96	-	0,015	0,018	0,010	-
data20_34	122	122	122	-	0,013	0,024	0,010	-
data20_35	114	114	114	-	0,016	0,023	0,010	-
data20_36	127	127	127	-	0,020	0,022	0,010	-

Instância	*Makespan*				*CPU* (s)			
	HyPRR		*GC*		*HyPRR*		*GC*	
	Best	*Avg.*	*Best*	*Avg.*	*Best*	*Avg.*	*Best*	*Avg.*
data20_37	1027	1027	1027	-	0,037	0,044	0,040	-
data20_38	1031	1031	1031	-	0,018	0,025	0,030	-
data20_39	1130	1130	1130	-	0,024	0,029	0,020	-
data20_40	1077	1077	1077	-	0,031	0,032	0,030	-
data20_41	1047	1047	1047	-	0,024	0,026	0,060	-
data20_42	1032	1032	1032	-	0,024	0,028	0,030	-
data20_43	1180	1180	1180	-	0,021	0,024	0,050	-
data20_44	1201	1201	1201	-	0,022	0,022	0,070	-
data20_45	1314	1314	1314	-	0,028	0,028	0,130	-
data20_46	1238	1238	1238	-	0,020	0,023	0,090	-
data20_47	1130	1130	1130	-	0,018	0,023	0,090	-
data20_48	1339	1339	1339	-	0,015	0,019	0,180	-
data20_49	159	159	159	-	0,020	0,026	0,060	-
data20_50	181	181	181	-	0,025	0,029	0,030	-
data20_51	132	132	132	-	0,022	0,022	0,010	-
data20_52	134	134	134	-	0,025	0,026	0,020	-
data20_53	135	135	135	-	0,020	0,023	0,080	-
data20_54	159	159	159	-	0,016	0,016	0,020	-
data20_55	137	137	137	-	0,020	0,024	0,000	-
data20_56	168	168	168	-	0,017	0,028	0,010	-
data20_57	131	131	131	-	0,018	0,022	0,090	-
data20_58	159	159	159	-	0,017	0,021	0,050	-
data20_59	141	141	141	-	0,016	0,021	0,060	-
data20_60	127	127	127	-	0,018	0,023	0,010	-
data20_61	1325	1325	1325	-	0,024	0,024	0,070	-
data20_62	1446	1446	1446	-	0,041	0,048	0,120	-
data20_63	1248	1248	1248	-	0,030	0,033	0,060	-
data20_64	1596	1596	1596	-	0,028	0,032	0,130	-
data20_65	1752	1752	1752	-	0,018	0,021	0,060	-

Instância	*Makespan*				*CPU* (s)			
	HyPRR		*GC*		*HyPRR*		*GC*	
	Best	*Avg.*	*Best*	*Avg.*	*Best*	*Avg.*	*Best*	*Avg.*
data20_66	1665	1665	1665	-	0,019	0,022	0,030	-
data20_67	1283	1283	1283	-	0,020	0,028	0,040	-
data20_68	1323	1323	1323	-	0,018	0,018	0,060	-
data20_69	1775	1775	1775	-	0,017	0,020	0,120	-
data20_70	1596	1596	1596	-	0,016	0,022	0,130	-
data20_71	1575	1575	1575	-	0,017	0,024	0,070	-
data20_72	1561	1561	1561	-	0,018	0,023	0,140	-
data20_73	187	187	187	-	0,025	0,029	0,010	-
data20_74	163	163	163	-	0,018	0,021	0,090	-
data20_75	221	221	221	-	0,022	0,031	0,000	-
data20_76	194	194	194	-	0,026	0,031	0,000	-
data20_77	170	170	170	-	0,019	0,021	0,010	-
data20_78	201	201	201	-	0,019	0,022	0,020	-
data20_79	196	196	196	-	0,019	0,026	0,010	-
data20_80	188	188	188	-	0,018	0,021	0,010	-
data20_81	167	167	167	-	0,018	0,018	0,080	-
data20_82	181	181	181	-	0,013	0,016	0,130	-
data20_83	192	192	192	-	0,016	0,016	0,140	-
data20_84	193	193	193	-	0,017	0,017	0,030	-
data20_85	1988	1988	1988	-	0,026	0,026	0,030	-
data20_86	1792	1792	1792	-	0,020	0,020	0,750	-
data20_87	1722	1722	1722	-	0,020	0,020	0,020	-
data20_88	1735	1735	1735	-	0,050	0,050	0,090	-
data20_89	2402	2402	2402	-	0,018	0,018	0,060	-
data20_90	1610	1610	1610	-	0,016	0,016	0,120	-
data20_91	2093	2093	2093	-	0,019	0,019	0,050	-
data20_92	1963	1963	1963	-	0,016	0,016	0,200	-
data20_93	2232	2232	2232	-	0,017	0,021	0,290	-
data20_94	2124	2124	2124	-	0,017	0,025	0,150	-

Instância	*Makespan*				*CPU* (s)			
	HyPRR		*GC*		*HyPRR*		*GC*	
	Best	*Avg.*	*Best*	*Avg.*	*Best*	*Avg.*	*Best*	*Avg.*
data20_95	1831	1831	1831	-	0,018	0,022	0,060	-
data20_96	2284	2284	2284	-	0,016	0,020	0,100	-
data30_1	143	143	143	-	0,082	0,090	35,010	-
data30_2	168	168	168	-	0,076	0,076	0,020	-
data30_3	157	157	157	-	0,079	0,084	0,000	-
data30_4	134	134	134	-	0,072	0,083	0,430	-
data30_5	170	170	170	-	0,061	0,066	0,010	-
data30_6	158	158	158	-	0,055	0,056	0,020	-
data30_7	147	147	147	-	0,047	0,048	0,010	-
data30_8	152	152	152	-	0,052	0,057	0,010	-
data30_9	155	155	155	-	0,052	0,054	0,050	-
data30_10	152	152	152	-	0,062	0,066	0,290	-
data30_11	133	133	133	-	0,048	0,053	0,220	-
data30_12	153	153	153	-	0,055	0,055	0,030	-
data30_13	1667	1667	1667	-	0,073	0,075	0,030	-
data30_14	1620	1620	1620	-	0,106	0,108	3,950	-
data30_15	1586	1586	1586	-	0,064	0,071	0,280	-
data30_16	1347	1347	1347	-	0,077	0,082	0,210	-
data30_17	1841	1841	1841	-	0,050	0,050	0,490	-
data30_18	1487	1487	1487	-	0,049	0,049	0,230	-
data30_19	2017	2017	2017	-	0,053	0,055	0,340	-
data30_20	1460	1460	1460	-	0,055	0,061	0,500	-
data30_21	1538	1538	1538	-	0,045	0,045	3,070	-
data30_22	1389	1389	1389	-	0,046	0,051	0,220	-
data30_23	1558	1558	1558	-	0,046	0,050	0,190	-
data30_24	1343	1343	1343	-	0,047	0,050	0,190	-
data30_25	144	144,2	144	-	0,088	0,142	0,320	-
data30_26	128	128	128	-	0,077	0,086	0,010	-
data30_27	212	212	212	-	0,093	0,098	0,460	-

Instância	*Makespan*				*CPU* (s)			
	HyPRR		*GC*		*HyPRR*		*GC*	
	Best	*Avg.*	*Best*	*Avg.*	*Best*	*Avg.*	*Best*	*Avg.*
data30_28	140	140	140	-	0,101	0,107	7,280	-
data30_29	178	178	178	-	0,079	0,085	0,180	-
data30_30	194	194	194	-	0,067	0,072	1,180	-
data30_31	171	171	171	-	0,068	0,071	4,810	-
data30_32	137	137	137	-	0,073	0,079	0,660	-
data30_33	202	202	202	-	0,070	0,074	52,340	-
data30_34	148	148	148	-	0,069	0,070	0,030	-
data30_35	165	165	165	-	0,061	0,066	9,600	-
data30_36	143	143	143	-	0,068	0,072	20,880	-
data30_37	1582	1582	1582	-	0,098	0,103	0,070	-
data30_38	1901	1901	1901	-	0,173	0,178	5,450	-
data30_39	1415	1415	1415	-	0,111	0,116	23,300	-
data30_40	1623	1623	1623	-	0,113	0,118	51,480	-
data30_41	1906	1906	1906	-	0,071	0,080	0,330	-
data30_42	1727	1727	1727	-	0,101	0,108	0,470	-
data30_43	1318	1318	1318	-	0,095	0,102	41,880	-
data30_44	1724	1724	1724	-	0,067	0,072	84,800	-
data30_45	1539	1539	1539	-	0,087	0,090	0,110	-
data30_46	1817	1817	1817	-	0,080	0,084	0,130	-
data30_47	1692	1692	1692	-	0,071	0,073	0,210	-
data30_48	1813	1813	1813	-	0,097	0,103	37,100	-
data30_49	185	185	185	-	0,124	0,168	0,260	-
data30_50	252	252	252	-	0,101	0,103	0,010	-
data30_51	240	240	240	-	0,103	0,110	0,300	-
data30_52	205	205	205	-	0,094	0,098	0,010	-
data30_53	269	269	269	-	0,078	0,081	0,010	-
data30_54	230	230	230	-	0,074	0,075	0,010	-
data30_55	227	227	227	-	0,066	0,067	0,010	-
data30_56	223	223	223	-	0,073	0,075	0,010	-

Instância	*Makespan*				*CPU* (s)			
	HyPRR		*GC*		*HyPRR*		*GC*	
	Best	*Avg.*	*Best*	*Avg.*	*Best*	*Avg.*	*Best*	*Avg.*
data30_57	233	233	233	-	0,071	0,071	0,010	-
data30_58	185	185	185	-	0,063	0,066	0,020	-
data30_59	127	127	127	-	0,066	0,076	0,010	-
data30_60	213	213	213	-	0,055	0,060	0,010	-
data30_61	2605	2605	2605	-	0,104	0,109	4,020	-
data30_62	2149	2149	2149	-	0,107	0,112	0,030	-
data30_63	2534	2534	2534	-	0,148	0,157	1,490	-
data30_64	1896	1896	1896	-	0,138	0,145	1,140	-
data30_65	1983	1983	1983	-	0,075	0,080	0,100	-
data30_66	2444	2444	2444	-	0,075	0,083	0,100	-
data30_67	2421	2421	2421	-	0,071	0,076	97,320	-
data30_68	2347	2347	2347	-	0,069	0,072	11,480	-
data30_69	2145	2145	2145	-	0,074	0,078	0,160	-
data30_70	2055	2055	2055	-	0,097	0,102	20,730	-
data30_71	2327	2327	2327	-	0,077	0,080	156,570	-
data30_72	2301	2301	2301	-	0,081	0,083	140,050	-
data30_73	291	291	291	-	0,087	0,092	0,220	-
data30_74	305	305	305	-	0,103	0,118	0,570	-
data30_75	314	314	314	-	0,092	0,098	1,050	-
data30_76	271	271	271	-	0,096	0,102	0,000	-
data30_77	301	301	301	-	0,080	0,080	18,990	-
data30_78	310	310	310	-	0,081	0,086	0,530	-
data30_79	297	297	297	-	0,083	0,088	0,000	-
data30_80	311	311	311	-	0,064	0,072	0,010	-
data30_81	318	318	318	-	0,063	0,067	0,020	-
data30_82	279	279	279	-	0,063	0,069	0,010	-
data30_83	316	316	316	-	0,068	0,073	0,020	-
data30_84	282	282	282	-	0,062	0,062	0,020	-
data30_85	2743	2743	2743	-	0,078	0,080	2,870	-

Instância	*Makespan*				*CPU* (s)			
	HyPRR		*GC*		*HyPRR*		*GC*	
	Best	*Avg.*	*Best*	*Avg.*	*Best*	*Avg.*	*Best*	*Avg.*
data30_86	3329	3329	3329	-	0,067	0,069	0,030	-
data30_87	3313	3313	3313	-	0,081	0,088	179,720	-
data30_88	3719	3719	3719	-	0,089	0,099	0,060	-
data30_89	3466	3466	3466	-	0,061	0,067	0,090	-
data30_90	2499	2499	2499	-	0,073	0,084	0,060	-
data30_91	2766	2766	2766	-	0,056	0,059	0,090	-
data30_92	3205	3205	3205	-	0,065	0,067	0,130	-
data30_93	2831	2831	2831	-	0,057	0,064	36,130	-
data30_94	2333	2333	2333	-	0,058	0,066	0,190	-
data30_95	2801	2801	2801	-	0,056	0,065	0,170	-
data30_96	2513	2513	2513	-	0,068	0,072	145,070	-
data40_1	197	197	197	-	0,184	0,215	0,210	-
data40_2	214	214	214	-	0,184	0,189	0,190	-
data40_3	222	222	222	-	0,236	0,246	0,240	-
data40_4	218	218	218	-	0,207	0,229	0,130	-
data40_5	220	220	220	-	0,148	0,155	0,240	-
data40_6	214	214	214	-	0,133	0,136	0,250	-
data40_7	214	214	214	-	0,134	0,136	0,260	-
data40_8	198	198	198	-	0,140	0,147	0,420	-
data40_9	195	195	195	-	0,127	0,130	0,220	-
data40_10	179	179	179	-	0,148	0,152	0,230	-
data40_11	208	208	208	-	0,137	0,142	0,280	-
data40_12	191	191	191	-	0,148	0,155	0,240	-
data40_13	2183	2183	2183	-	0,227	0,232	0,290	-
data40_14	1922	1922	1922	-	0,183	0,206	0,280	-
data40_15	2111	2111	2111	-	0,180	0,183	2,150	-
data40_16	1647	1647	1647	-	0,172	0,187	0,940	-
data40_17	1975	1975	1975	-	0,175	0,193	3,100	-
data40_18	2423	2423	2423	-	0,163	0,176	0,430	-

Instância	*Makespan*				*CPU* (s)			
	HyPRR		*GC*		*HyPRR*		*GC*	
	Best	*Avg.*	*Best*	*Avg.*	*Best*	*Avg.*	*Best*	*Avg.*
data40_19	2289	2289	2289	-	0,123	0,133	68,470	-
data40_20	2036	2036	2036	-	0,152	0,158	0,470	-
data40_21	2102	2102	2102	-	0,159	0,167	1,580	-
data40_22	1655	1655	1655	-	0,130	0,135	16,930	-
data40_23	2312	2312	2312	-	0,128	0,135	48,980	-
data40_24	2142	2142	2142	-	0,121	0,124	0,600	-
data40_25	232	232	232	-	0,244	0,269	11,610	-
data40_26	235	235,6	235	-	0,344	0,422	0,190	-
data40_27	225	226,8	225	-	0,337	0,497	0,280	-
data40_28	248	250,6	248	-	0,366	0,448	0,070	-
data40_29	241	241	241	-	0,200	0,225	0,070	-
data40_30	250	250	250	-	0,194	0,254	0,250	-
data40_31	187	187	187	-	0,191	0,200	0,060	-
data40_32	240	240	240	-	0,199	0,227	0,080	-
data40_33	230	230	230	-	0,162	0,175	0,260	-
data40_34	207	207	207	-	0,179	0,203	0,050	-
data40_35	231	231	231	-	0,167	0,176	0,110	-
data40_36	197	197	197	-	0,179	0,239	0,110	-
data40_37	2457	2457	2457	-	0,291	0,314	0,140	-
data40_38	2141	2141	2141	-	0,343	0,384	0,420	-
data40_39	2287	2287	2287	-	0,326	0,373	0,070	-
data40_40	2187	2187	2187	-	0,292	0,301	0,840	-
data40_41	1911	1911	1911	-	0,218	0,231	0,510	-
data40_42	2116	2116	2116	-	0,256	0,263	0,270	-
data40_43	2207	2207	2207	-	0,214	0,228	246,540	-
data40_44	2247	2247	2247	-	0,249	0,266	0,470	-
data40_45	2248	2248	2248	-	0,251	0,258	21,990	-
data40_46	2118	2118	2118	-	0,188	0,198	0,560	-
data40_47	2035	2035	2035	-	0,205	0,217	0,280	-

Instância	*Makespan*				*CPU* (s)			
	HyPRR		*GC*		*HyPRR*		*GC*	
	Best	*Avg.*	*Best*	*Avg.*	*Best*	*Avg.*	*Best*	*Avg.*
data40_48	2471	2471	2471	-	0,213	0,224	0,550	-
data40_49	277	277	277	-	0,205	0,213	0,200	-
data40_50	260	260	260	-	0,291	0,313	0,010	-
data40_51	344	344	344	-	0,279	0,325	8,070	-
data40_52	324	324	323	-	0,354	0,424	0,100	-
data40_53	274	274	274	-	0,181	0,182	0,180	-
data40_54	276	276	276	-	0,175	0,178	0,180	-
data40_55	332	332	332	-	0,165	0,166	0,190	-
data40_56	291	291	291	-	0,208	0,218	0,050	-
data40_57	350	350	350	-	0,152	0,165	0,220	-
data40_58	258	258	258	-	0,173	0,180	0,180	-
data40_59	270	270	270	-	0,166	0,167	0,220	-
data40_60	306	306	306	-	0,176	0,188	0,190	-
data40_61	2877	2877	2877	-	0,271	0,288	60,930	-
data40_62	2272	2272	2272	-	0,243	0,261	0,240	-
data40_63	3586	3586	3586	-	0,301	0,328	0,270	-
data40_64	3041	3041,3	3041	-	0,265	0,352	1007,570	-
data40_65	3431	3431	3431	-	0,168	0,177	0,450	-
data40_66	2773	2773	2773	-	0,208	0,216	343,800	-
data40_67	3946	3946	3946	-	0,236	0,254	0,460	-
data40_68	3481	3481	3487	-	0,239	0,249	392,480	-
data40_69	3116	3116	3116	-	0,213	0,218	0,670	-
data40_70	3158	3158	3158	-	0,163	0,165	0,820	-
data40_71	2668	2668	2668	-	0,175	0,181	0,670	-
data40_72	3199	3199	3199	-	0,180	0,183	0,520	-
data40_73	361	361	361	-	0,288	0,305	4,690	-
data40_74	411	411	411	-	0,227	0,239	300,760	-
data40_75	379	379	379	-	0,243	0,252	0,040	-
data40_76	332	332	332	-	0,221	0,225	0,290	-

Instância	*Makespan*				*CPU* (s)			
	HyPRR		*GC*		*HyPRR*		*GC*	
	Best	*Avg.*	*Best*	*Avg.*	*Best*	*Avg.*	*Best*	*Avg.*
data40_77	431	431	431	-	0,161	0,170	0,170	-
data40_78	409	409	409	-	0,174	0,178	0,170	-
data40_79	443	443	443	-	0,150	0,156	0,180	-
data40_80	349	349	349	-	0,161	0,173	0,170	-
data40_81	478	478	478	-	0,150	0,155	0,240	-
data40_82	355	355	355	-	0,158	0,163	0,230	-
data40_83	379	379	379	-	0,148	0,154	0,250	-
data40_84	370	370	370	-	0,157	0,162	0,240	-
data40_85	3878	3878	3878	-	0,307	0,331	23,610	-
data40_86	3649	3649	3649	-	0,286	0,294	18,740	-
data40_87	4441	4441	4441	-	0,250	0,260	1001,740	-
data40_88	4553	4553	4553	-	0,373	0,405	4,380	-
data40_89	4161	4161	4161	-	0,179	0,192	1000,690	-
data40_90	3878	3878	3878	-	0,165	0,171	0,450	-
data40_91	3707	3707	3707	-	0,188	0,193	0,400	-
data40_92	4058	4058	4058	-	0,185	0,196	0,460	-
data40_93	3931	3931	3931	-	0,140	0,143	0,670	-
data40_94	4078	4078	4078	-	0,185	0,190	0,450	-
data40_95	4076	4076	4076	-	0,159	0,167	0,730	-
data40_96	3990	3990	3990	-	0,146	0,157	0,780	-
data50_1	261	261	261	-	0,324	0,330	0,250	-
data50_2	255	255	255	-	0,423	0,574	0,500	-
data50_3	279	279	279	-	0,429	0,451	0,240	-
data50_4	280	280	280	-	0,310	0,320	0,230	-
data50_5	266	266	266	-	0,290	0,310	0,200	-
data50_6	291	291	291	-	0,298	0,305	20,310	-
data50_7	268	268	268	-	0,262	0,276	3,240	-
data50_8	254	254	254	-	0,290	0,297	0,260	-
data50_9	239	239	239	-	0,266	0,276	0,640	-

Instância	*Makespan*				*CPU* (s)			
	HyPRR		*GC*		*HyPRR*		*GC*	
	Best	*Avg.*	*Best*	*Avg.*	*Best*	*Avg.*	*Best*	*Avg.*
data50_10	248	248	248	-	0,283	0,292	9,290	-
data50_11	263	263	263	-	0,236	0,251	0,300	-
data50_12	232	232	232	-	0,289	0,297	1,520	-
data50_13	2412	2412,7	2412	-	0,469	0,577	1,430	-
data50_14	2710	2710	2710	-	0,426	0,439	1,520	-
data50_15	2328	2328	2328	-	0,578	0,732	27,230	-
data50_16	2189	2189	2189	-	0,420	0,433	0,360	-
data50_17	2459	2459	2459	-	0,288	0,305	37,740	-
data50_18	2590	2590	2590	-	0,297	0,303	110,150	-
data50_19	2723	2723	2723	-	0,264	0,276	0,580	-
data50_20	2462	2462	2462	-	0,264	0,273	28,330	-
data50_21	2029	2029	2029	-	0,271	0,293	16,670	-
data50_22	2610	2610	2610	-	0,234	0,239	0,680	-
data50_23	2395	2395	2395	-	0,249	0,260	0,650	-
data50_24	2443	2443	2443	-	0,296	0,311	3,030	-
data50_25	293	296,6	293	-	0,636	1,319	0,060	-
data50_26	278	278	278	-	0,620	0,762	1,790	-
data50_27	272	272,1	272	-	0,537	0,689	0,070	-
data50_28	315	315,5	315	-	0,653	0,774	0,110	-
data50_29	296	296	296	-	0,394	0,436	0,170	-
data50_30	277	277	277	-	0,419	0,439	33,600	-
data50_31	249	249	249	-	0,360	0,399	0,470	-
data50_32	248	248	248	-	0,402	0,420	0,350	-
data50_33	284	284	284	-	0,341	0,366	0,180	-
data50_34	249	249	249	-	0,346	0,355	183,740	-
data50_35	269	269,6	269	-	0,399	0,445	0,610	-
data50_36	264	264,2	264	-	0,463	0,548	0,100	-
data50_37	2825	2825	2825	-	0,733	0,852	2,620	-
data50_38	2599	2599,5	2599	-	1,182	1,381	587,070	-

Instância	*Makespan*				*CPU* (s)			
	HyPRR		*GC*		*HyPRR*		*GC*	
	Best	*Avg.*	*Best*	*Avg.*	*Best*	*Avg.*	*Best*	*Avg.*
data50_39	2801	2801	2801	-	0,841	0,953	5,430	-
data50_40	2600	2600	2600	-	0,787	0,850	0,450	-
data50_41	2693	2693	2722	-	0,543	0,556	1001,290	-
data50_42	2769	2769	2769	-	0,568	0,598	0,590	-
data50_43	3020	3020	3020	-	0,677	0,699	249,720	-
data50_44	2707	2707	2707	-	0,650	0,790	9,040	-
data50_45	2775	2775	2775	-	0,412	0,428	1,760	-
data50_46	2910	2910	2910	-	0,395	0,411	1014,810	-
data50_47	2511	2511	2511	-	0,632	0,648	0,700	-
data50_48	2537	2537	2537	-	0,419	0,447	0,650	-
data50_49	405	405	405	-	0,600	0,747	4,430	-
data50_50	420	420	420	-	0,591	0,703	0,210	-
data50_51	354	354	354	-	0,673	0,838	7,980	-
data50_52	385	385	385	-	0,444	0,483	0,070	-
data50_53	382	382	382	-	0,372	0,387	0,260	-
data50_54	383	383	383	-	0,370	0,384	0,240	-
data50_55	336	336	336	-	0,362	0,374	0,240	-
data50_56	420	420	420	-	0,359	0,373	0,430	-
data50_57	380	380	380	-	0,341	0,386	53,630	-
data50_58	372	372	372	-	0,347	0,372	0,210	-
data50_59	393	393	393	-	0,327	0,351	0,220	-
data50_60	387	387	387	-	0,328	0,338	0,220	-
data50_61	3679	3679	3679	-	0,539	0,578	633,510	-
data50_62	3704	3704	3704	-	0,569	0,605	20,350	-
data50_63	3972	3972	3972	-	0,578	0,652	0,100	-
data50_64	3802	3802	3802	-	0,655	0,679	0,130	-
data50_65	3596	3596	3596	-	0,375	0,394	1003,440	-
data50_66	3481	3481	3481	-	0,350	0,379	0,460	-
data50_67	3501	3501	3501	-	0,369	0,392	0,570	-

Instância	*Makespan*				*CPU* (s)			
	HyPRR		*GC*		*HyPRR*		*GC*	
	Best	*Avg.*	*Best*	*Avg.*	*Best*	*Avg.*	*Best*	*Avg.*
data50_68	3712	3712	3712	-	0,343	0,353	0,610	-
data50_69	3528	3528	3528	-	0,378	0,392	0,870	-
data50_70	3678	3678	3678	-	0,281	0,304	0,620	-
data50_71	3769	3769	3769	-	0,332	0,347	1,010	-
data50_72	3353	3353	3353	-	0,384	0,398	0,800	-
data50_73	538	538	538	-	0,445	0,459	1006,720	-
data50_74	520	520	520	-	0,384	0,404	1006,690	-
data50_75	553	553	553	-	0,635	0,705	1,830	-
data50_76	500	500	500	-	0,510	0,541	0,230	-
data50_77	561	561	561	-	0,366	0,377	0,220	-
data50_78	509	509	509	-	0,337	0,358	0,200	-
data50_79	489	489	489	-	0,345	0,355	0,410	-
data50_80	553	553	553	-	0,359	0,381	1000,730	-
data50_81	497	497	497	-	0,279	0,295	0,240	-
data50_82	456	456	456	-	0,348	0,378	0,180	-
data50_83	500	500	500	-	0,322	0,334	0,200	-
data50_84	532	532	532	-	0,377	0,400	120,200	-
data50_85	5389	5389	5389	-	0,448	0,452	46,690	-
data50_86	4543	4543	4543	-	0,402	0,436	0,420	-
data50_87	4380	4380	4380	-	0,671	0,704	1006,150	-
data50_88	5146	5146	5146	-	0,454	0,474	44,010	-
data50_89	5488	5488	5488	-	0,417	0,445	0,480	-
data50_90	5176	5176	5176	-	0,372	0,390	186,330	-
data50_91	4857	4857	4857	-	0,264	0,285	0,730	-
data50_92	5193	5193	5193	-	0,310	0,328	0,700	-
data50_93	5449	5449	5449	-	0,309	0,317	0,680	-
data50_94	4663	4663	4663	-	0,296	0,308	0,950	-
data50_95	5201	5201	5201	-	0,311	0,324	1,080	-
data50_96	4436	4436	4436	-	0,319	0,335	1,030	-

Instância	*Makespan*				*CPU* (s)			
	HyPRR		*GC*		*HyPRR*		*GC*	
	Best	*Avg.*	*Best*	*Avg.*	*Best*	*Avg.*	*Best*	*Avg.*
data60_1	294	294	294	-	0,716	0,743	0,400	-
data60_2	308	308	308	-	0,767	0,801	2,250	-
data60_3	321	321	321	-	0,655	0,686	0,440	-
data60_4	312	312	312	-	0,774	0,850	0,710	-
data60_5	320	320	320	-	0,572	0,613	0,590	-
data60_6	308	308	308	-	0,571	0,595	0,500	-
data60_7	292	292	292	-	0,581	0,594	0,320	-
data60_8	310	310	310	-	0,567	0,595	0,830	-
data60_9	308	308	308	-	0,539	0,555	0,380	-
data60_10	318	318	318	-	0,602	0,639	5,370	-
data60_11	299	299	299	-	0,579	0,605	1007,570	-
data60_12	308	308	308	-	0,481	0,503	1,260	-
data60_13	3483	3483	3491	-	0,729	0,818	7,810	-
data60_14	2969	2969	2969	-	0,809	0,827	2,590	-
data60_15	3070	3074,1	3070	-	1,117	1,645	0,190	-
data60_16	3046	3046	3046	-	0,718	0,741	1,630	-
data60_17	2761	2761	2761	-	0,618	0,651	1,320	-
data60_18	3172	3172	3172	-	0,570	0,592	0,640	-
data60_19	2988	2988	2988	-	0,492	0,509	127,120	-
data60_20	3122	3122	3122	-	0,572	0,625	6,130	-
data60_21	2875	2875	2875	-	0,779	0,845	477,420	-
data60_22	2881	2881	2881	-	0,559	0,591	16,340	-
data60_23	2905	2905	2905	-	0,581	0,620	11,150	-
data60_24	2961	2961	2961	-	0,468	0,477	133,220	-
data60_25	334	340	333	-	1,731	2,923	0,070	-
data60_26	279	279	279	-	1,198	1,458	0,330	-
data60_27	334	334,4	334	-	1,133	1,487	4,450	-
data60_28	305	305,6	305	-	2,198	2,630	0,010	-
data60_29	296	296,1	296	-	0,823	1,102	41,510	-

Instância	*Makespan*				*CPU* (s)			
	HyPRR		*GC*		*HyPRR*		*GC*	
	Best	*Avg.*	*Best*	*Avg.*	*Best*	*Avg.*	*Best*	*Avg.*
data60_30	309	309,1	309	-	0,836	1,102	69,040	-
data60_31	336	336	336	-	0,786	0,823	0,050	-
data60_32	365	365	365	-	0,879	0,953	0,040	-
data60_33	363	363	363	-	0,700	0,741	37,230	-
data60_34	339	339	339	-	0,747	0,817	366,960	-
data60_35	313	313,1	313	-	0,743	0,960	0,100	-
data60_36	315	315	315	-	0,624	0,680	0,500	-
data60_37	2800	2807	2800	-	1,986	2,766	14,040	-
data60_38	3001	3001,5	3001	-	1,532	1,852	0,100	-
data60_39	3594	3599,3	3594	-	2,121	4,057	117,760	-
data60_40	3856	3858,8	3856	-	1,769	2,293	1001,650	-
data60_41	3565	3565	3565	-	1,030	1,137	76,160	-
data60_42	3738	3738	3738	-	0,805	0,855	757,010	-
data60_43	3271	3271	3282	-	1,078	1,114	1001,770	-
data60_44	3520	3520	3520	-	0,930	0,982	7,730	-
data60_45	3059	3059	3059	-	1,221	1,259	51,520	-
data60_46	3019	3019	3019	-	1,046	1,105	1001,240	-
data60_47	3202	3202	3202	-	1,082	1,109	838,470	-
data60_48	3355	3355	3355	-	0,959	1,031	1,710	-
data60_49	456	456	456	-	0,819	0,862	0,210	-
data60_50	478	478	478	-	1,039	1,138	1000,600	-
data60_51	472	472,2	472	-	1,303	1,567	7,590	-
data60_52	452	452	452	-	1,107	1,408	0,060	-
data60_53	435	435	435	-	0,730	0,763	0,200	-
data60_54	400	400	400	-	0,749	0,811	0,190	-
data60_55	483	483	483	-	0,767	0,782	1000,290	-
data60_56	554	554	554	-	0,747	0,784	1000,300	-
data60_57	475	475	475	-	0,734	0,806	1005,020	-
data60_58	494	494	494	-	0,701	0,739	0,250	-

Instância	*Makespan*				*CPU* (s)			
	HyPRR		*GC*		*HyPRR*		*GC*	
	Best	*Avg.*	*Best*	*Avg.*	*Best*	*Avg.*	*Best*	*Avg.*
data60_59	447	447	447	-	0,668	0,685	0,230	-
data60_60	425	425	425	-	0,626	0,657	0,200	-
data60_61	4388	4388	4388	-	1,653	1,818	1011,570	-
data60_62	5296	5296	5296	-	1,325	1,471	1008,510	-
data60_63	4425	4425	4425	-	0,815	0,862	103,900	-
data60_64	4331	4331	4331	-	1,285	1,355	33,480	-
data60_65	4189	4189	4189	-	0,675	0,690	0,710	-
data60_66	4698	4698	4698	-	0,885	0,917	1000,490	-
data60_67	4958	4958	4958	-	0,832	0,869	0,610	-
data60_68	4668	4668	4668	-	0,836	0,881	0,460	-
data60_69	4798	4798	4798	-	0,881	0,937	0,800	-
data60_70	5074	5074	5074	-	0,630	0,683	1,860	-
data60_71	4159	4159	4159	-	0,629	0,660	1003,680	-
data60_72	4412	4412	4412	-	0,655	0,702	6,930	-
data60_73	583	583	583	-	0,972	1,025	0,210	-
data60_74	595	595	595	-	1,084	1,241	1003,800	-
data60_75	581	581	581	-	1,093	1,200	4,240	-
data60_76	593	593	593	-	0,877	0,913	0,230	-
data60_77	665	665	665	-	0,743	0,773	0,190	-
data60_78	612	612	612	-	0,611	0,629	0,120	-
data60_79	635	635	635	-	0,624	0,650	0,200	-
data60_80	664	664	664	-	0,663	0,692	0,200	-
data60_81	554	554	554	-	0,615	0,633	0,240	-
data60_82	612	612	612	-	0,595	0,614	0,210	-
data60_83	664	664	664	-	0,642	0,662	0,200	-
data60_84	589	589	589	-	0,572	0,595	0,250	-
data60_85	5468	5468	5468	-	1,338	1,402	1002,220	-
data60_86	6009	6009	6009	-	1,076	1,143	78,900	-
data60_87	5000	5000	5000	-	1,100	1,170	48,170	-

Instância	*Makespan*				*CPU* (s)			
	HyPRR		*GC*		*HyPRR*		*GC*	
	Best	*Avg.*	*Best*	*Avg.*	*Best*	*Avg.*	*Best*	*Avg.*
data60_88	5861	5861	5861	-	0,810	0,838	0,380	-
data60_89	6093	6093	6093	-	0,559	0,600	0,830	-
data60_90	6801	6801	6801	-	0,548	0,607	0,890	-
data60_91	6354	6354	6354	-	0,751	0,781	1007,250	-
data60_92	5765	5765	5765	-	0,630	0,666	0,740	-
data60_93	6284	6284	6284	-	0,766	0,804	1003,590	-
data60_94	6865	6865	6865	-	0,532	0,599	1,900	-
data60_95	6121	6121	6121	-	0,500	0,530	1,410	-
data60_96	6135	6135	6135	-	0,595	0,637	1,130	-
data70_1	333	333	333	-	1,273	1,331	0,710	-
data70_2	348	348	348	-	1,148	1,250	1,800	-
data70_3	335	335	335	-	1,024	1,053	0,840	-
data70_4	376	377,4	376	-	1,251	1,628	1001,540	-
data70_5	356	356	356	-	0,890	0,922	0,280	-
data70_6	390	390	390	-	0,846	0,863	2,210	-
data70_7	381	381	381	-	0,896	0,974	1,080	-
data70_8	356	356	356	-	0,860	0,886	0,240	-
data70_9	357	357	357	-	0,737	0,812	23,640	-
data70_10	386	386	386	-	0,765	0,793	104,220	-
data70_11	381	381	381	-	0,728	0,758	0,510	-
data70_12	330	330	330	-	0,766	0,809	3,330	-
data70_13	3591	3591	3591	-	1,009	1,130	60,930	-
data70_14	3536	3536	3536	-	1,077	1,098	1,470	-
data70_15	3737	3737	3744	-	1,102	1,205	1011,730	-
data70_16	3565	3566,4	3565	-	1,497	2,080	443,740	-
data70_17	3868	3868	3868	-	0,887	0,930	62,570	-
data70_18	3819	3819	3819	-	0,779	0,815	0,810	-
data70_19	3546	3546	3546	-	0,941	0,977	50,420	-
data70_20	3205	3205	3205	-	1,139	1,194	27,950	-

Instância	*Makespan*				*CPU* (s)			
	HyPRR		*GC*		*HyPRR*		*GC*	
	Best	*Avg.*	*Best*	*Avg.*	*Best*	*Avg.*	*Best*	*Avg.*
data70_21	3398	3398	3398	-	0,690	0,706	823,450	-
data70_22	3502	3502	3502	-	0,773	0,821	162,040	-
data70_23	2873	2873	2873	-	0,883	0,914	75,310	-
data70_24	3343	3343	3343	-	0,667	0,677	1,720	-
data70_25	392	392,8	392	-	1,776	2,934	2,590	-
data70_26	453	456,3	453	-	2,455	3,269	0,120	-
data70_27	444	447,6	444	-	2,164	4,103	135,210	-
data70_28	367	371,7	364	-	3,116	4,448	148,370	-
data70_29	374	374	374	-	1,267	1,377	1000,180	-
data70_30	353	353,8	353	-	1,577	2,163	1000,280	-
data70_31	418	418	418	-	0,961	0,996	1000,880	-
data70_32	333	333	333	-	1,088	1,226	7,670	-
data70_33	358	358	358	-	1,259	1,479	10,530	-
data70_34	392	392	392	-	1,112	1,241	0,280	-
data70_35	416	416,2	416	-	1,407	1,773	4,460	-
data70_36	399	399	399	-	1,126	1,362	101,640	-
data70_37	3819	3819	3819	-	2,758	3,265	77,890	-
data70_38	3772	3772,7	3794	-	2,153	2,593	1003,090	-
data70_39	4128	4128	4128	-	1,900	2,426	0,350	-
data70_40	3760	3761,2	3763	-	2,315	3,059	1016,090	-
data70_41	3717	3717	3717	-	1,280	1,397	8,390	-
data70_42	3781	3781	3781	-	1,505	1,644	1005,250	-
data70_43	4251	4251	4261	-	1,790	2,027	1003,580	-
data70_44	3489	3489	3489	-	1,413	1,487	0,960	-
data70_45	3622	3622	3622	-	1,373	1,430	1,150	-
data70_46	3681	3681	3681	-	1,288	1,350	1004,370	-
data70_47	3972	3972	3972	-	1,432	1,502	1,960	-
data70_48	3728	3728	3728	-	1,333	1,407	35,220	-
data70_49	501	501	501	-	1,621	1,801	0,260	-

Instância	*Makespan*				*CPU* (s)			
	HyPRR		*GC*		*HyPRR*		*GC*	
	Best	*Avg.*	*Best*	*Avg.*	*Best*	*Avg.*	*Best*	*Avg.*
data70_50	568	568	568	-	1,550	1,620	15,250	-
data70_51	527	527	527	-	1,745	1,949	0,190	-
data70_52	621	621	621	-	1,569	1,863	1001,530	-
data70_53	526	526	526	-	1,106	1,154	1003,060	-
data70_54	568	568	568	-	1,397	1,652	1000,950	-
data70_55	575	575	575	-	1,089	1,122	0,240	-
data70_56	565	565	565	-	0,992	1,047	0,250	-
data70_57	481	481	481	-	0,883	0,929	0,340	-
data70_58	512	512	512	-	1,010	1,053	0,300	-
data70_59	507	507	507	-	1,144	1,240	0,240	-
data70_60	513	513	513	-	0,964	1,077	0,250	-
data70_61	5153	5153	5153	-	1,853	1,934	0,390	-
data70_62	4071	4071	4071	-	2,150	2,244	1020,200	-
data70_63	5015	5015	5015	-	1,663	1,696	0,450	-
data70_64	4415	4415	4415	-	2,262	2,345	0,220	-
data70_65	5132	5132	5132	-	1,142	1,232	1015,110	-
data70_66	5417	5417	5417	-	1,204	1,276	1,070	-
data70_67	5628	5628	5628	-	1,152	1,221	0,920	-
data70_68	5264	5264	5264	-	1,026	1,080	507,330	-
data70_69	5254	5254	5254	-	1,170	1,223	1,370	-
data70_70	5734	5734	5734	-	0,891	0,933	1,410	-
data70_71	5146	5146	5146	-	1,316	1,419	0,780	-
data70_72	4920	4920	4920	-	1,078	1,131	1,770	-
data70_73	664	664	664	-	1,206	1,253	14,500	-
data70_74	632	632	632	-	1,692	1,939	1003,490	-
data70_75	690	690	690	-	1,806	2,081	1013,960	-
data70_76	730	730,1	730	-	1,840	2,340	1004,710	-
data70_77	756	756	756	-	0,986	1,032	1,790	-
data70_78	678	678	678	-	1,150	1,292	1001,140	-

Instância	*Makespan*				*CPU* (s)			
	HyPRR		*GC*		*HyPRR*		*GC*	
	Best	*Avg.*	*Best*	*Avg.*	*Best*	*Avg.*	*Best*	*Avg.*
data70_79	719	719	719	-	0,990	1,024	0,280	-
data70_80	715	715	715	-	0,859	0,932	0,230	-
data70_81	737	737	737	-	0,969	1,012	0,270	-
data70_82	674	674	674	-	0,990	1,016	0,260	-
data70_83	664	664	664	-	0,937	0,976	0,290	-
data70_84	741	741	741	-	0,862	0,897	1000,540	-
data70_85	6299	6299	6299	-	1,855	2,080	1006,900	-
data70_86	7014	7014	7014	-	1,977	2,074	0,080	-
data70_87	7514	7514	7514	-	1,410	1,445	1001,230	-
data70_88	6743	6743	6743	-	1,601	1,684	1006,700	-
data70_89	6397	6397	6397	-	1,050	1,108	0,670	-
data70_90	8036	8036	8036	-	1,016	1,058	0,690	-
data70_91	6690	6690	6690	-	0,984	1,054	416,010	-
data70_92	7102	7102	7102	-	1,024	1,075	1004,240	-
data70_93	7563	7563	7563	-	0,963	1,008	1,080	-
data70_94	6887	6887	6887	-	0,908	0,944	1,580	-
data70_95	8142	8142	8142	-	0,868	0,930	1,840	-
data70_96	7184	7184	7184	-	0,922	0,972	1,170	-
data80_1	397	397,3	397	-	1,784	2,099	1,050	-
data80_2	404	404	404	-	1,879	1,945	0,310	-
data80_3	397	397	397	-	1,892	1,959	0,360	-
data80_4	396	396	395	-	2,227	2,823	4,550	-
data80_5	410	410	410	-	1,232	1,278	67,950	-
data80_6	411	411	411	-	1,235	1,261	13,600	-
data80_7	410	410	410	-	1,321	1,429	7,880	-
data80_8	395	395	395	-	1,300	1,344	3,410	-
data80_9	462	462	462	-	1,201	1,239	1001,380	-
data80_10	379	379	379	-	1,249	1,306	1002,070	-
data80_11	407	407	407	-	1,155	1,191	0,590	-

Instância	*Makespan*				*CPU* (s)			
	HyPRR		*GC*		*HyPRR*		*GC*	
	Best	*Avg.*	*Best*	*Avg.*	*Best*	*Avg.*	*Best*	*Avg.*
data80_12	403	403	403	-	1,182	1,215	3,940	-
data80_13	4231	4231	4231	-	1,990	2,182	0,150	-
data80_14	4174	4174	4174	-	2,066	2,245	0,460	-
data80_15	3679	3679	3679	-	1,885	1,989	5,880	-
data80_16	3945	3945	3945	-	1,730	1,850	12,720	-
data80_17	3954	3954	3954	-	1,233	1,286	1009,760	-
data80_18	3779	3779	3779	-	1,407	1,441	13,190	-
data80_19	4007	4007	4007	-	1,382	1,406	34,940	-
data80_20	4018	4018	4018	-	1,453	1,521	65,610	-
data80_21	3973	3973	3973	-	1,085	1,118	1004,270	-
data80_22	4138	4138	4138	-	1,283	1,358	8,250	-
data80_23	4154	4154	4154	-	1,226	1,309	1013,580	-
data80_24	4443	4443	4443	-	1,676	1,785	384,190	-
data80_25	446	447,3	446	-	2,796	4,162	0,870	-
data80_26	377	379,3	378	-	2,326	3,752	1003,630	-
data80_27	473	475,1	473	-	3,145	4,955	1000,430	-
data80_28	451	451	451	-	2,419	2,877	1001,900	-
data80_29	417	417,3	417	-	2,252	3,091	1027,360	-
data80_30	422	422	422	-	1,862	2,140	4,320	-
data80_31	466	466	466	-	2,166	2,663	22,280	-
data80_32	395	395	395	-	1,682	1,909	2,830	-
data80_33	405	405	405	-	1,581	1,762	78,170	-
data80_34	428	428	428	-	1,974	2,377	0,380	-
data80_35	398	398	398	-	1,849	2,196	0,770	-
data80_36	407	407	407	-	1,736	1,913	1002,980	-
data80_37	4621	4623,6	4621	-	4,445	5,974	7,070	-
data80_38	3642	3642,2	3642	-	4,698	6,703	523,330	-
data80_39	4118	4124,6	4120	-	4,438	6,501	1011,570	-
data80_40	4348	4357,1	4348	-	3,922	4,487	132,820	-

Instância	*Makespan*				*CPU* (s)			
	HyPRR		*GC*		*HyPRR*		*GC*	
	Best	*Avg.*	*Best*	*Avg.*	*Best*	*Avg.*	*Best*	*Avg.*
data80_41	4436	4436	4436	-	2,732	2,978	1004,010	-
data80_42	4534	4534	4534	-	2,862	3,085	1030,440	-
data80_43	3926	3926	3926	-	2,550	2,611	0,740	-
data80_44	4132	4132	4132	-	2,231	2,388	1012,010	-
data80_45	4054	4054	4054	-	2,079	2,244	3,680	-
data80_46	4316	4316	4336	-	2,022	2,128	1002,430	-
data80_47	4812	4812	4812	-	1,659	1,728	133,290	-
data80_48	4376	4376	4376	-	2,050	2,289	1005,550	-
data80_49	608	608	608	-	2,961	3,928	16,500	-
data80_50	661	661	661	-	2,707	2,984	0,010	-
data80_51	623	623	623	-	2,631	2,876	0,660	-
data80_52	606	606,1	606	-	2,820	3,810	9,660	-
data80_53	589	589	589	-	1,882	1,998	1001,750	-
data80_54	660	660	660	-	1,870	1,977	33,740	-
data80_55	655	655	655	-	1,747	1,819	0,210	-
data80_56	578	578	578	-	1,618	1,804	0,270	-
data80_57	613	613	613	-	1,511	1,586	0,350	-
data80_58	607	607	607	-	1,664	1,824	0,380	-
data80_59	628	628	628	-	1,866	1,991	1011,790	-
data80_60	547	547	547	-	1,369	1,511	0,350	-
data80_61	6546	6546	6546	-	4,211	4,567	1011,860	-
data80_62	5416	5416	5416	-	2,378	2,498	226,640	-
data80_63	6080	6080	6080	-	3,267	3,454	0,120	-
data80_64	6146	6146,1	6146	-	3,534	5,117	1000,700	-
data80_65	6332	6332	6332	-	2,293	2,395	1,650	-
data80_66	6597	6597	6597	-	1,815	1,943	1,350	-
data80_67	5865	5865	5865	-	2,284	2,385	1001,390	-
data80_68	5584	5584	5584	-	1,454	1,546	1,110	-
data80_69	5743	5743	5743	-	1,394	1,462	2,770	-

Instância	*Makespan*				*CPU* (s)			
	HyPRR		*GC*		*HyPRR*		*GC*	
	Best	*Avg.*	*Best*	*Avg.*	*Best*	*Avg.*	*Best*	*Avg.*
data80_70	6377	6377	6377	-	1,723	1,796	2,000	-
data80_71	6002	6002	6002	-	1,645	1,732	1,530	-
data80_72	5812	5812	5812	-	1,457	1,539	1,720	-
data80_73	866	866	866	-	2,164	2,243	27,370	-
data80_74	787	787	787	-	2,059	2,159	1004,660	-
data80_75	822	822	822	-	2,333	2,495	1,200	-
data80_76	776	776	776	-	2,369	2,450	0,270	-
data80_77	836	836	836	-	1,473	1,509	0,350	-
data80_78	817	817	817	-	1,587	1,681	0,290	-
data80_79	752	752	752	-	1,532	1,607	0,320	-
data80_80	795	795	795	-	1,348	1,455	0,260	-
data80_81	766	766	766	-	1,364	1,451	0,300	-
data80_82	831	831	831	-	1,302	1,362	0,280	-
data80_83	774	774	774	-	1,504	1,601	1,180	-
data80_84	760	760	760	-	1,377	1,448	0,370	-
data80_85	7325	7325	7325	-	2,132	2,257	0,420	-
data80_86	8135	8135	8135	-	2,184	2,261	0,610	-
data80_87	8317	8317	8317	-	3,074	3,157	139,300	-
data80_88	8625	8625	8625	-	2,116	2,191	0,480	-
data80_89	8157	8157	8157	-	1,683	1,735	1,110	-
data80_90	8158	8158	8158	-	1,433	1,564	1,740	-
data80_91	7383	7383	7383	-	1,444	1,521	1,100	-
data80_92	7269	7269	7269	-	1,421	1,507	1,210	-
data80_93	7974	7974	7974	-	1,522	1,581	2,160	-
data80_94	8173	8173	8173	-	1,215	1,293	3,020	-
data80_95	8118	8118	8118	-	1,538	1,587	2,090	-
data80_96	8410	8410	8410	-	1,641	1,766	1,780	-
data90_1	464	464	464	-	2,592	2,666	0,540	-
data90_2	476	476	476	-	3,177	3,598	0,020	-

Instância	*Makespan*				*CPU* (s)			
	HyPRR		*GC*		*HyPRR*		*GC*	
	Best	*Avg.*	*Best*	*Avg.*	*Best*	*Avg.*	*Best*	*Avg.*
data90_3	432	432	432	-	2,758	3,461	11,020	-
data90_4	423	423	423	-	3,130	3,431	0,760	-
data90_5	496	496	496	-	2,071	2,203	94,460	-
data90_6	409	409	409	-	1,749	1,850	1006,530	-
data90_7	459	459	459	-	2,035	2,068	25,650	-
data90_8	482	482	482	-	2,026	2,162	1008,630	-
data90_9	421	421	421	-	1,837	1,893	191,650	-
data90_10	459	459	459	-	1,639	1,739	12,070	-
data90_11	454	454	454	-	1,901	2,034	44,110	-
data90_12	449	449	449	-	1,826	1,895	21,900	-
data90_13	4702	4702	4702	-	3,108	3,937	262,300	-
data90_14	4763	4763	4763	-	2,407	2,526	2,670	-
data90_15	4689	4689	4689	-	2,936	3,193	2,300	-
data90_16	4938	4938	4938	-	3,095	3,461	3,340	-
data90_17	4588	4588	4588	-	2,021	2,163	191,590	-
data90_18	4741	4741	4741	-	1,961	2,076	20,010	-
data90_19	4548	4548	4548	-	1,973	2,047	3,270	-
data90_20	4169	4169	4169	-	1,857	1,973	3,490	-
data90_21	4923	4923	4923	-	1,736	1,784	1006,990	-
data90_22	4750	4750	4750	-	1,544	1,622	1005,570	-
data90_23	4478	4478	4478	-	1,599	1,733	1005,260	-
data90_24	4796	4796	4796	-	1,770	1,962	62,280	-
data90_25	475	478	473	-	5,951	10,052	1010,570	-
data90_26	556	558,6	554	-	6,246	10,226	1002,690	-
data90_27	513	513,4	513	-	4,201	5,646	88,860	-
data90_28	500	505	499	-	5,760	10,922	110,300	-
data90_29	434	434,4	434	-	3,527	4,516	52,850	-
data90_30	515	515	515	-	2,687	3,260	1000,710	-
data90_31	517	517	517	-	2,375	2,558	0,170	-

Instância	*Makespan*				*CPU* (s)			
	HyPRR		*GC*		*HyPRR*		*GC*	
	Best	*Avg.*	*Best*	*Avg.*	*Best*	*Avg.*	*Best*	*Avg.*
data90_32	434	434	434	-	2,624	2,973	72,570	-
data90_33	438	439,3	438	-	5,469	6,702	1000,920	-
data90_34	504	504,1	504	-	3,195	4,448	290,560	-
data90_35	443	443	443	-	2,582	3,353	1003,830	-
data90_36	495	495	495	-	2,111	2,411	883,440	-
data90_37	4587	4587	4587	-	5,348	6,158	1003,600	-
data90_38	4857	4868	4857	-	6,296	8,616	4,340	-
data90_39	4492	4492	4492	-	5,658	6,152	0,350	-
data90_40	5520	5520	5583	-	8,188	10,607	1004,150	-
data90_41	4563	4563	4570	-	4,547	4,906	1013,230	-
data90_42	4969	4969	4994	-	3,692	3,864	1008,500	-
data90_43	4625	4625	4634	-	3,206	3,322	1003,740	-
data90_44	5200	5200	5200	-	4,074	4,294	9,780	-
data90_45	4859	4859	4859	-	3,738	3,958	50,320	-
data90_46	5251	5251	5251	-	4,123	4,435	105,840	-
data90_47	4714	4714	4714	-	3,984	4,156	1017,600	-
data90_48	3881	3881	3903	-	3,727	4,122	1002,670	-
data90_49	695	695,9	695	-	4,943	7,056	1017,810	-
data90_50	659	659	659	-	3,425	3,732	1,240	-
data90_51	691	691,1	691	-	4,560	5,596	1011,710	-
data90_52	656	656	656	-	3,578	4,006	1005,800	-
data90_53	699	699	699	-	2,557	2,676	0,410	-
data90_54	646	646	646	-	2,746	2,837	0,320	-
data90_55	677	677	677	-	2,287	2,390	0,310	-
data90_56	667	667	667	-	2,090	2,192	1009,490	-
data90_57	616	616	616	-	2,265	2,366	0,470	-
data90_58	651	651	651	-	2,403	2,743	0,350	-
data90_59	701	701	701	-	2,597	2,728	1001,010	-
data90_60	602	602	602	-	2,371	2,466	0,350	-

Instância	*Makespan*				*CPU* (s)			
	HyPRR		*GC*		*HyPRR*		*GC*	
	Best	*Avg.*	*Best*	*Avg.*	*Best*	*Avg.*	*Best*	*Avg.*
data90_61	6460	6460	6460	-	5,115	5,539	0,090	-
data90_62	7297	7297	7297	-	4,880	5,044	0,490	-
data90_63	7413	7413	7413	-	3,801	3,940	1001,520	-
data90_64	6942	6942	6942	-	6,613	8,174	1000,150	-
data90_65	7353	7353	7353	-	2,609	2,677	1,400	-
data90_66	6464	6464	6464	-	3,562	4,321	1012,890	-
data90_67	6829	6829	6829	-	2,556	2,741	1017,040	-
data90_68	6659	6659	6659	-	2,931	3,104	1,140	-
data90_69	6174	6174	6174	-	2,199	2,478	3,440	-
data90_70	7911	7911	7911	-	2,598	2,750	2,130	-
data90_71	6586	6586	6586	-	2,292	2,394	1004,630	-
data90_72	7252	7252	7252	-	2,286	2,387	1020,020	-
data90_73	884	884	884	-	2,596	2,935	1,090	-
data90_74	896	896	896	-	3,180	3,349	0,360	-
data90_75	878	878	878	-	3,717	4,482	26,650	-
data90_76	910	910	910	-	3,790	4,161	23,330	-
data90_77	824	824	824	-	2,143	2,260	0,250	-
data90_78	836	836	836	-	2,403	2,440	73,460	-
data90_79	896	896	896	-	2,197	2,304	0,400	-
data90_80	916	916	916	-	2,289	2,431	0,330	-
data90_81	854	854	854	-	1,845	2,063	0,290	-
data90_82	998	998	998	-	1,834	1,904	0,300	-
data90_83	884	884	884	-	1,859	1,938	0,380	-
data90_84	900	900	900	-	1,707	1,775	1005,740	-
data90_85	8723	8723	8723	-	3,835	4,264	1011,480	-
data90_86	8374	8374	8374	-	4,255	4,392	1004,970	-
data90_87	8762	8762	8762	-	2,785	2,934	0,750	-
data90_88	7928	7928	7928	-	3,009	3,113	0,860	-
data90_89	9092	9092	9092	-	2,235	2,348	1,250	-

Instância	*Makespan*				*CPU* (s)			
	HyPRR		*GC*		*HyPRR*		*GC*	
	Best	*Avg.*	*Best*	*Avg.*	*Best*	*Avg.*	*Best*	*Avg.*
data90_90	8267	8267	8267	-	2,162	2,217	2,190	-
data90_91	8817	8817	8817	-	2,677	2,771	1,300	-
data90_92	9545	9545	9545	-	2,192	2,268	2,480	-
data90_93	8388	8388	8388	-	1,900	1,963	5,750	-
data90_94	9091	9091	9091	-	1,852	2,010	5,700	-
data90_95	8664	8664	8664	-	2,031	2,145	2,700	-
data90_96	8728	8728	8728	-	2,053	2,124	2,460	-
data100_1	513	513	513	-	4,364	4,726	0,690	-
data100_2	555	555	555	-	3,543	3,708	2,570	-
data100_3	503	503	503	-	3,279	3,404	0,820	-
data100_4	527	527	527	-	3,515	3,804	57,250	-
data100_5	499	499	499	-	2,807	2,959	19,930	-
data100_6	520	520	520	-	2,573	2,711	11,340	-
data100_7	557	557	557	-	3,014	3,103	1005,640	-
data100_8	513	513	513	-	2,525	2,602	61,110	-
data100_9	482	482	482	-	2,368	2,461	7,380	-
data100_10	491	491	491	-	2,429	2,538	1,020	-
data100_11	539	539	539	-	2,883	3,090	1017,670	-
data100_12	494	494	494	-	2,645	2,714	19,180	-
data100_13	4617	4617	4625	-	4,564	5,057	1004,480	-
data100_14	4917	4917	4917	-	3,502	3,654	12,910	-
data100_15	5061	5061	5061	-	3,580	3,943	1001,040	-
data100_16	4820	4820	4820	-	3,451	3,608	16,500	-
data100_17	4595	4595	4595	-	2,334	2,487	2,330	-
data100_18	5410	5410	5410	-	3,053	3,233	355,010	-
data100_19	4943	4943	4943	-	2,369	2,532	1010,990	-
data100_20	5693	5693	5693	-	2,777	2,914	1005,380	-
data100_21	5375	5375	5375	-	2,180	2,317	1014,610	-
data100_22	5099	5099	5099	-	2,643	2,749	1018,070	-

Instância	*Makespan*				*CPU* (s)			
	HyPRR		*GC*		*HyPRR*		*GC*	
	Best	*Avg.*	*Best*	*Avg.*	*Best*	*Avg.*	*Best*	*Avg.*
data100_23	5063	5063	5063	-	2,303	2,485	1022,300	-
data100_24	5358	5358	5358	-	2,370	2,525	6,240	-
data100_25	512	520	510	-	8,731	14,712	0,090	-
data100_26	501	502,4	500	-	8,151	12,614	104,380	-
data100_27	505	511,3	503	-	8,348	14,879	1000,340	-
data100_28	589	593	587	-	9,343	17,920	1,660	-
data100_29	535	535	535	-	3,924	5,863	1012,760	-
data100_30	507	507,3	507	-	4,534	5,878	1010,800	-
data100_31	488	488,8	488	-	4,392	5,658	1001,020	-
data100_32	480	480,4	480	-	5,019	7,384	83,120	-
data100_33	515	515	515	-	3,530	4,435	1000,740	-
data100_34	540	540,5	540	-	4,391	6,955	14,750	-
data100_35	540	540,3	540	-	4,243	5,842	328,200	-
data100_36	563	563	563	-	3,900	4,494	0,600	-
data100_37	5295	5295	5295	-	7,992	9,268	0,480	-
data100_38	5757	5758,8	5757	-	9,420	13,476	0,510	-
data100_39	5980	5981,6	5989	-	14,286	17,378	1001,650	-
data100_40	5408	5408	5408	-	8,295	9,146	0,810	-
data100_41	5644	5644	5644	-	5,046	5,259	11,350	-
data100_42	5419	5419	5419	-	5,648	5,955	40,330	-
data100_43	5880	5880	5880	-	4,642	4,958	975,210	-
data100_44	4823	4823	4878	-	5,297	5,693	1005,740	-
data100_45	4300	4300	4309	-	5,374	5,648	1008,690	-
data100_46	5363	5363	5363	-	3,880	4,168	3,510	-
data100_47	5437	5437	5445	-	6,206	6,639	1011,730	-
data100_48	5199	5199	5199	-	5,875	6,263	230,370	-
data100_49	802	802	802	-	5,772	6,673	0,510	-
data100_50	724	724	724	-	6,250	7,539	1016,340	-
data100_51	728	733,3	727	-	9,344	12,697	1000,990	-

Instância	*Makespan*				*CPU* (s)			
	HyPRR		*GC*		*HyPRR*		*GC*	
	Best	*Avg.*	*Best*	*Avg.*	*Best*	*Avg.*	*Best*	*Avg.*
data100_52	699	699,2	699	-	8,084	11,412	1001,340	-
data100_53	787	787	787	-	3,580	3,967	1001,900	-
data100_54	812	812	812	-	3,490	3,780	1002,010	-
data100_55	693	693	693	-	3,298	3,515	0,420	-
data100_56	785	785	785	-	3,345	3,765	1001,200	-
data100_57	724	724	724	-	3,358	4,235	0,380	-
data100_58	708	708	708	-	3,252	3,343	0,650	-
data100_59	734	734	734	-	3,139	3,494	0,890	-
data100_60	799	799	799	-	3,103	3,331	0,350	-
data100_61	7351	7351	7351	-	6,233	7,253	9,010	-
data100_62	6409	6409	6409	-	7,549	7,748	215,450	-
data100_63	7182	7182	7182	-	7,316	8,044	36,700	-
data100_64	7146	7146	7146	-	5,862	6,129	0,850	-
data100_65	7965	7965	7965	-	3,920	4,191	2,800	-
data100_66	7057	7057	7057	-	3,628	3,911	2,000	-
data100_67	8151	8151	8151	-	4,433	4,599	1,570	-
data100_68	7007	7007	7007	-	3,862	3,944	1,680	-
data100_69	7819	7819	7819	-	3,234	3,468	5,990	-
data100_70	8169	8169	8169	-	3,904	4,266	2,620	-
data100_71	7172	7172	7172	-	3,593	3,760	3,100	-
data100_72	7182	7182	7182	-	4,356	4,625	1007,210	-
data100_73	976	976	976	-	6,102	7,390	0,450	-
data100_74	951	951	951	-	4,090	4,384	0,490	-
data100_75	1047	1047	1047	-	5,375	5,855	1006,580	-
data100_76	995	995	995	-	4,364	4,852	1004,140	-
data100_77	994	994	994	-	3,384	3,855	1007,220	-
data100_78	1106	1106	1106	-	3,962	4,583	1000,030	-
data100_79	913	913	913	-	3,226	3,350	1016,080	-
data100_80	1014	1014	1014	-	3,184	3,281	1003,210	-

Instância	*Makespan*				*CPU* (s)			
	HyPRR		*GC*		*HyPRR*		*GC*	
	Best	*Avg.*	*Best*	*Avg.*	*Best*	*Avg.*	*Best*	*Avg.*
data100_81	1022	1022	1022	-	2,681	2,764	0,520	-
data100_82	994	994	994	-	2,670	2,848	0,520	-
data100_83	961	961	961	-	2,586	2,645	0,390	-
data100_84	994	994	994	-	2,680	2,990	0,330	-
data100_85	10187	10187	10187	-	6,534	6,817	1015,740	-
data100_86	9615	9615	9615	-	5,488	6,728	1000,030	-
data100_87	10409	10409	10409	-	5,281	5,427	1027,650	-
data100_88	10397	10397	10397	-	5,433	5,711	0,850	-
data100_89	9856	9856	9856	-	3,037	3,179	1005,340	-
data100_90	9980	9980	9980	-	3,778	3,940	1,830	-
data100_91	9177	9177	9177	-	2,958	3,010	3,200	-
data100_92	9745	9745	9745	-	3,526	3,754	1016,060	-
data100_93	10654	10654	10654	-	3,181	3,324	7,480	-
data100_94	9473	9473	9473	-	2,714	2,971	5,120	-
data100_95	9724	9724	9724	-	2,455	2,597	7,450	-
data100_96	9336	9336	9336	-	2,704	2,801	6,030	-