UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO – UFES
CENTRO TECNOLÓGICO
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

GILDÁSIO LECCHI CRAVO

ALGORITMOS HEURÍSTICOS INTELIGENTES PARA PROBLEMAS DE LAYOUT EM LINHA ÚNICA E LINHA DUPLA

VITÓRIA, ES
2021

GILDÁSIO LECCHI CRAVO

ALGORITMOS HEURÍSTICOS INTELIGENTES PARA PROBLEMAS DE LAYOUT EM LINHA ÚNICA E LINHA DUPLA

Tese submetida ao Programa de Pós-Graduação em Informática do Centro Tecnológico da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do Grau de Doutor em Ciência da Computação.

Orientador: Prof. Dr. André Renato Sales Amaral

VITÓRIA, ES

2021

Ficha catalográfica disponibilizada pelo Sistema Integrado de Bibliotecas - SIBI/UFES e elaborada pelo autor

---

Cravo, Gildásio Lecchi, 1982-
C898a ALGORITMOS HEURÍSTICOS INTELIGENTES PARA PROBLEMAS DE LAYOUT EM LINHA ÚNICA E LINHA DUPLA / Gildásio Lecchi Cravo. - 2021.
166 f. : il.

Orientador: André Renato Sales Amaral.
Tese (Doutorado em Informática) - Universidade Federal do Espírito Santo, Centro Tecnológico.

1. Facility layout. 2. Single-Row Facility Layout Problem (SRFLP). 3. Double-Row Layout Problem (DRLP). 4. Parallel Row Ordering Problem (PROP). 5. Bi-objetive Corridor Allocation Problem (bCAP). 6. Meta-heurísticas. I. Amaral, André Renato Sales. II. Universidade Federal do Espírito Santo. Centro Tecnológico. III. Título.

CDU: 004

ALGORITMOS HEURÍSTICOS INTELIGENTES PARA PROBLEMAS DE LAYOUT EM LINHA ÚNICA E LINHA DUPLA

Gildásio Lecchi Cravo

Tese submetida ao Programa de Pós-Graduação em Informática da Universidade Federal do Espírito Santo como requisito parcial para a obtenção do grau de Doutor em Ciência da Computação.

Aprovada em 03 de agosto de 2021:

Prof. Dr. André Renato Sales Amaral
Orientador

Profª. Drª. Maria Claudia Silva Boeres
Membro Interno

Prof. Dr. Mário Mestria
Membro Externo

Prof. Dr. Luciano Lessa Lorenzoni
Membro Externo

Prof. Dr. Geraldo Regis Mauri
Membro Interno

UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
Vitória-ES, 03 de agosto de 2021.

Este documento foi assinado digitalmente por [redacted]
Para verificar o original visite: https://api.lepisma.ufes.br/arquivos-assinados/242869?tipoArquivo=O

Dedico este trabalho à minha família, minha esposa Iara e minha filha Alice.

AGRADECIMENTOS

Agradeço à minha família, pelo amor e apoio incondicional.

À Iara, pelo amor e companheirismo durante os anos do doutorado e pela força nos momentos difíceis.

À minha filha Alice, pelo infinito amor e alegria diária de ser pai.

Ao meu orientador, André Renato Sales Amaral, pela orientação e pela competência técnica dedicada, que me serviram de exemplo profissional e pessoal. Além da confiança, paciência e conversas de apoio no decorrer dos anos de estudos.

Aos membros da banca examinadora, pela participação na defesa deste trabalho e também pelas valiosas contribuições cedidas.

RESUMO

O estudo de layout de facilidades tem como objetivo determinar a melhor utilização do espaço disponível, resultando em processos de fabricação mais efetivos no contexto da indústria. Esta tese aborda quatro problemas de layout de facilidades, categorizados como layout em linha, em que facilidades devem ser organizadas em uma ou duas linhas retas, respeitando algumas restrições de alocação. Inicialmente é abordado o problema de layout de facilidades em linha única (SRFLP), que consiste em arranjar facilidades ao longo de uma linha reta, a fim de minimizar a soma ponderada das distâncias entre todos os pares de facilidades. Os outros três problemas abordados neste estudo são extensões do SRFLP, em que as facilidades são dispostas em duas linhas, sendo eles: o problema de layout em linha dupla (DRLP), o problema de ordenação em linhas paralelas (PROP) e o problema de alocação de corredor biobjetivo (bCAP). Um algoritmo denominado GRASP-F é proposto para o SRFLP. Os experimentos computacionais mostram a eficiência do método com a melhora dos valores conhecidos para 29 das 93 instâncias da literatura com até 1000 facilidades. Até a presente data, este é o segundo trabalho a considerar problemas dessa magnitude. Para o DRLP, uma abordagem com estratégias heurísticas, denominada PSO-DRLP, é proposta baseando-se na meta-heurística *Particle Swarm Optimization*. O PSO-DRLP apresentou valores iguais ou melhores que os valores conhecidos para 35 de 51 instâncias da literatura, e, para as 16 restantes, os valores encontrados estão bem próximos dos melhores valores conhecidos. O algoritmo de solução para o PROP, denominado AILS, baseia-se na meta-heurística ILS, mas diferentemente do padrão dessa meta-heurística, foram utilizadas novas estratégias de intensificação e diversificação, além de utilizar técnicas para acelerar o cálculo do ganho na função objetivo no movimento de vizinhança usado na busca local. Os resultados encontrados melhoraram 49 de 100 instâncias com resultados prévios conhecidos e para 51 instâncias restantes, os melhores resultados conhecidos foram alcançados. Além desses testes, foram realizados experimentos utilizando 6 instâncias com até 300 facilidades, inéditas no contexto do PROP. Para o bCAP, um algoritmo semelhante ao AILS-PROP foi proposto, também em duas fases e com técnicas para acelerar o cálculo do ganho na função objetivo nos movimentos de vizinhança utilizados na busca local, tendo obtido excelentes resultados para 76 instâncias testadas. De modo geral, as propostas de soluções para os quatro problemas podem ser consideradas excelentes alternativas para resolver problemas de layout em linhas única e dupla, sendo possível a obtenção de resultados de alta qualidade para problemas de grande porte em tempos computacionais baixos.

Palavras-chaves: Layout de facilidades, SRFLP, DRLP, PROP, bCAP, Meta-heurísticas.

ABSTRACT

The study of layout of facilities aims to determine the best use of available space, resulting in more effective manufacturing processes in the context of industry. This thesis addresses four facility layout problems, categorized as row layout, in which facilities must be arranged in one or two straight lines, respecting some allocation constraints. Initially, the problem of single-row facility layout (SRFLP) is addressed, which consists of arranging facilities along a straight line, in order to minimize the weighted sum of the distances between all pairs of facilities. The other three problems addressed in this study are SRFLP extensions, in which the facilities are arranged in two lines, namely: the double-row layout problem (DRLP), the parallel row ordering problem (PROP) and the bi-objective corridor allocation problem (bCAP). An algorithm called GRASP-F is proposed for SRFLP. The computational experiments show the efficiency of the method by improving the known-values for 29 out of 93 instances in the literature with up to 1000 facilities. To date, this is the second work to consider problems of this magnitude. For DRLP, a purely heuristic approach, called PSO-DRLP, is proposed based on the Particle Swarm Optimization meta-heuristic. The PSO-DRLP presented values equal to or better than the known-values for 35 of 51 instances in the literature, and, for the remaining 16, the values found are very close to the best-known values. The solution algorithm for PROP, called AILS, is based on the ILS meta-heuristic, but unlike the standard, two phases with different intensification and diversification characteristics were used, in addition to using techniques to accelerate the calculation of the gain in the objective function in the neighborhood move used in the local search. The results found improved 49 out of 100 instances with previous known results and for the remaining 51 instances the best-known results were achieved. In addition to these tests, experiments were carried out using 6 instances with up to 300 facilities, unprecedented in the context of PROP. For bCAP, an algorithm similar to AILS-PROP was proposed, also in two phases and with techniques to accelerate the calculation of the gain in the objective function in the neighborhood movements used in the local search, having obtained excellent results for 76 tested instances. In general, the proposed solutions for the four problems can be considered excellent alternatives to solve layout problems in single and double lines, being possible to obtain high quality results for large problems in low computational times.

Keywords: Facility Layout, SRFLP, DRLP, PROP, bCAP, Meta-heuristics.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ALGORITMOS

SUMÁRIO

1 INTRODUÇÃO

O layout de facilidades é um estudo sistemático de como arranjar as facilidades que concorrem na produção, de forma ótima, dentro de um espaço disponível. Em outras palavras, é a melhor utilização do espaço disponível para que resulte em um processo industrial mais efetivo, através da menor distância entre as facilidades e no menor tempo de processamento das tarefas (deslocamento de peças, montagens de peças, passagens de objetos ou pessoas de um setor para outro, deslocamento de produtos, dentre outras) que são enviadas entre as facilidades (DIAS, 1993).

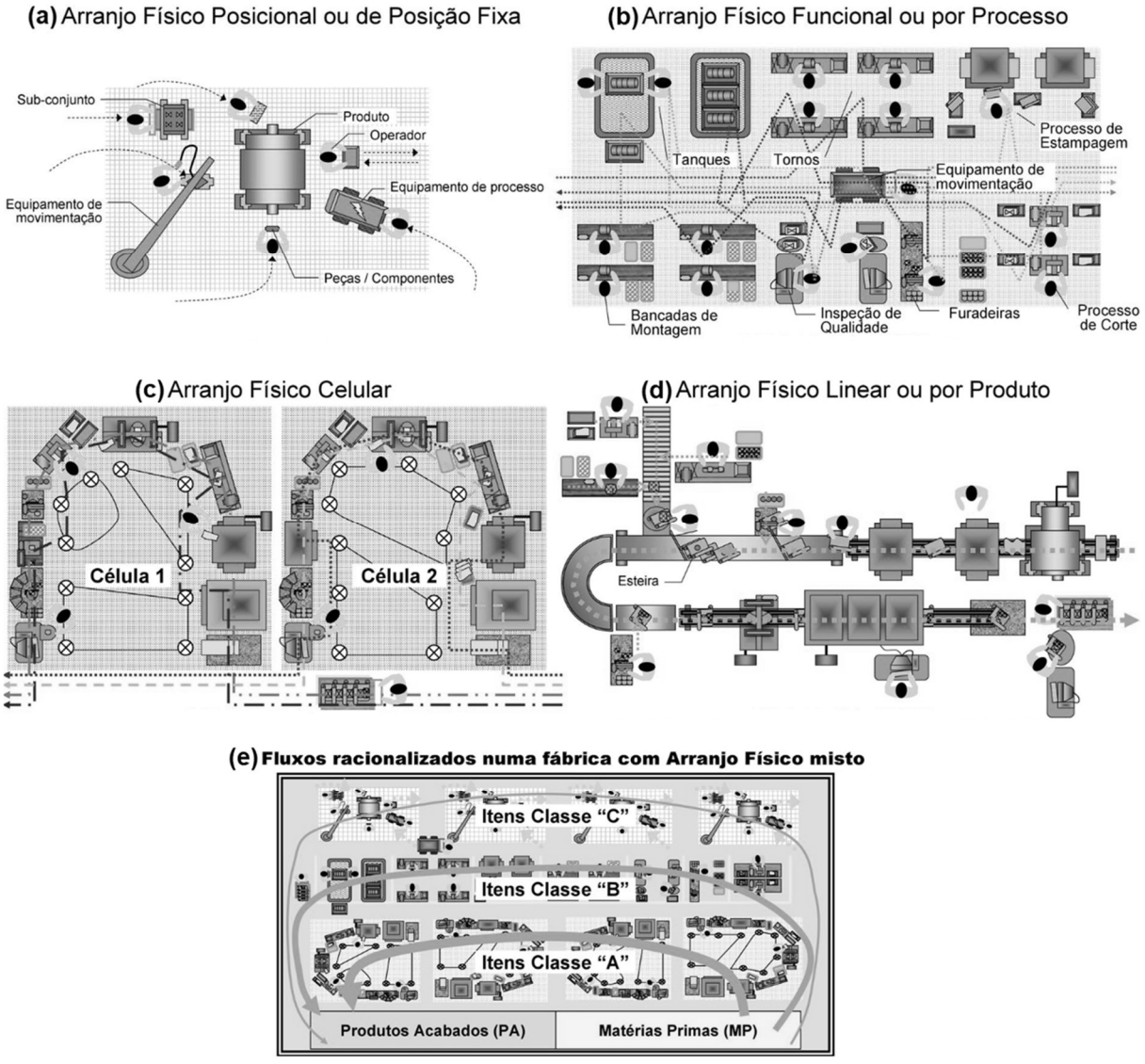
As facilidades podem ser máquinas, centros de trabalho, células de manufatura, departamentos de um edifício, armazéns, robôs em sistemas de manufatura, equipamentos e pessoal de produção ou até mesmo cilindros de um disco magnético (HERAGU, 1997; PICARD; QUEYRANNE, 1981; NEARCHOU, 2006).

Dependendo da forma como as facilidades são organizadas, os seguintes tipos de layouts são definidos (CURY, 2006; MARTINS; LAUGENI, 1998):

- Posicional (ou posição fixa): o produto não se movimenta, ficando a cargo do operário ou máquina (facilidades) mover-se no ambiente. Esse tipo de layout é usado quando a produção é pequena e o custo da movimentação dos produtos é muito alto, conforme exemplo na Figura 1(a);
- Por processo (ou funcional): o processo é intermitente ou em série, sendo as facilidades agrupadas conforme a natureza da operação, garantindo um maior controle na operação de produtos de alta precisão, como exemplificado na Figura 1(b);
- Celular: consiste em arranjar em um só local (a célula) as diversas facilidades necessárias à fabricação completa de um produto, como mostra a Figura 1(c);
- Por produto (ou linear): o processo de produção é contínuo e as facilidade são fixas. Assim, os produtos se movimentam entre as facilidades, como no exemplo da Figura 1(d); e
- Layout combinado (ou misto): ocorre com a combinação dos anteriores, de modo a aproveitar em um determinado processo as vantagens de cada layout, como no exemplo da Figura 1(e), com a distribuição das funcionalidades agrupadas em subprocesso organizados de acordo com os tipos de layout anteriores.

Cada um dos tipos tem como finalidade aproveitar ao máximo o espaço disponível e assegurar a disposição mais eficiente das facilidades aumentando a produção e, consequentemente, o lucro na linha de produção. Tais fatores podem ser alcançados com a utilização de layouts de facilidades otimizados na indústria (DRIRA; PIERREVAL; HAJRI-GABOUJ, 2007).

Figura 1 – Tipos de layouts de facilidades de acordo com a classificação apresentada



Fonte: Miyake (2005).

De modo geral, no momento de planejar um layout (arranjo físico), devem ser consideradas especificações como espaços necessários, distâncias a serem cobertas, circulação de pessoas ou materiais, estimativas de demanda de cada produto no sistema, os requisitos de processo, quantidade de operações, fluxos entre os vários componentes, a necessidade de espaços para a

operação e manutenção dos equipamentos e a disponibilidade de espaço para novas configurações (CELLIN, 2017).

Uma característica importante em instalações modernas é a flexibilidade do layout. A flexibilidade é necessária para garantir uma alta taxa de utilização devido ao grande valor de investimento nessas instalações. A falta de flexibilidade torna as instalações ociosas quando ocorrem necessidades de reorganizar o layout. Além disso, um layout flexível proporciona uma resposta rápida às mudanças na demanda dos clientes (HASSAN, 1994).

O projeto de um layout é de importância fundamental na implantação de um sistema flexível de manufatura (FMS, do inglês *Flexible Manufacturing Systems*), pois é difícil projetar um layout e as modificações são custosas. O layout afeta de forma significativa toda a eficiência do sistema de produção (TUBAILEH; SIAM, 2017).

Para Tompkins et al. (1996), no planejamento das instalações, o sistema de manuseio de material tem um efeito significativo na produtividade de todo o sistema, sendo estimado que até metade do total das despesas operacionais é devida ao manuseio de materiais. Garantir um bom desempenho do sistema de manuseio de materiais é fundamental para melhorar o desempenho de um FMS. Assim, é de grande importância desenvolver um layout considerando o sistema de manuseio desde o estágio inicial do projeto (TUBAILEH; SIAM, 2017).

Para aumentar a flexibilidade do sistema é importante que se utilize um dispositivo de manipulação automatizado, como um veículo guiado automaticamente (VGA). O VGA é superior ao transportador convencional no que diz respeito à utilização, custo e flexibilidade. Assim, estes são amplamente empregados em FMS (AMARAL, 2018). Os VGAs têm melhor desempenho quando se movem em linhas retas, o que torna interessante a utilização de máquinas organizadas em linhas retas em um FMS (TUBAILEH; SIAM, 2017). Desse contexto, surgem os problemas de layout de facilidades em linha (RLP, do inglês, *Row Layout Problem*), os quais objetivam determinar o arranjo de facilidades ao longo de $k \geq 1$ linhas retas.

Os RLPs que têm recebido bastante atenção nos últimos anos, com diversas formulações e abordagens para solução (ANJOS; VIEIRA, 2017; DRIRA; PIERREVAL; HAJRI-GABOUJ, 2007; HOSSEINI-NASAB et al., 2018; PÉREZ-GOSENDE; MULA; DÍAZ-MADROÑERO, 2021), são os denominados:

- Problema de layout em linha única (SRFLP, do inglês *Single-Row Facility Layout Problem*) (SIMMONS, 1969);

- Problema de layout em múltiplas linhas (MRLP, do inglês *Multi-Row Facility Layout Problem*) (GEN; IDA; CHENG, 1995); e
- Problema de layout em linha dupla (DRLP, do inglês *Double-Row Layout Problem*) (CHUNG; TANCHOCO, 2010), incluindo as especializações denominadas como problema de ordenação em linhas paralelas (PROP, do inglês *Parallel Row Ordering Problem*) (AMARAL, 2013a) e problema de alocação em corredor (CAP, do inglês *Corridor Allocation Problem*) (AMARAL, 2012).

Como forma de mensurar quão bom é um layout, utiliza-se normalmente uma função de custo total de fluxo de material. O fluxo de material representa uma medida relativa aos pares de facilidades envolvidas no layout, como, por exemplo, a quantidade de materiais manuseados ou informações trocadas entre as facilidades, a quantidade de pessoas que trafegam entre departamentos, entre outros.

Os problemas são de natureza combinatória e difíceis de serem resolvidos (TUBAILEH; SIAM, 2017). Os problemas de layout são NP-Difíceis em geral (DRIRA; PIERREVAL; HAJRI-GABOUJ, 2007). Assim, o uso de heurísticas e meta-heurísticas é fundamental para que seja possível a resolução dos problemas de layouts em tempos computacionais aceitáveis, possibilitando o tratamento de problemas de maior porte.

Dado o grande interesse no contexto da indústria, manufatura e projetos de layouts em geral, tanto pelo valor econômico, quanto pela grande aplicabilidade nas diversas áreas do conhecimento, como Administração, Programação e Controle da Produção, Pesquisa Operacional e Ciência da Computação, é justificável o estudo e proposição de soluções para os diversos problemas de layout em linha.

1.1 OBJETIVOS E CONTRIBUIÇÕES

Devido ao grande interesse e aplicabilidade dos problemas no mundo real, principalmente em FMS, o objetivo geral desta tese é contribuir com o desenvolvimento de soluções para resolver problemas de layout de facilidades.

Como objetivos específicos desse trabalho, têm-se o estudo e o desenvolvimento de algoritmos heurísticos inteligentes para a resolver o problema de layout em linha única (SRFLP); o problema de layout de facilidades em linha dupla (DRLP); o problema de ordenação em linhas

paralelas (PROP); e o problema de alocação de corredor biobjetivo (bCAP) que além do custo total do fluxo de material, também objetiva minimizar o comprimento do corredor. Além disso, as soluções propostas são capazes de resolver instâncias consideradas de grande porte para os respectivos problemas tratados nesta tese.

As eficácias dos métodos propostos são avaliadas através de experimentos computacionais onde os resultados encontrados pelos algoritmos propostos são comparados com as melhores propostas da literatura recente que exploram problemas de grande porte. Nos experimentos são analisadas métricas como os melhores resultados, soluções médias e os desvios-padrão, além de medidas de desempenho como desvio percentual relativo das soluções médias e o desvios-padrão relativos. Em todos os casos, os tempos de execução são compatíveis com os algoritmos utilizados na comparação dos resultados.

Nos testes computacionais realizados com o algoritmo proposto para o SRFLP são utilizadas 93 instâncias contendo de 110 a 1000 facilidades e a comparação é feita com os resultados dos algoritmos de Ozcelik (2012), Palubeckis (2015), Rubio-Sánchez et al. (2016) e Palubeckis (2017); para o DRLP as instâncias utilizadas possuem tamanhos até $n = 80$ facilidades, tendo os resultados das instâncias com até $n = 17$ comparados com soluções exatas e, para instâncias maiores, a comparação é feita com as heurísticas de Amaral (2020) e Guan et al. (2020); e instâncias com até $n = 300$ facilidades são utilizadas nos testes do PROP e o bCAP. Os resultados obtidos para o PROP serão comparados com o melhor algoritmo de Maadi, Javidnia e Jamshidi (2017) e, os do bCAP, comparados com o algoritmo de Kalita, Datta e Palubeckis (2019).

Os objetivos específicos levam às contribuições obtidas e destacadas a seguir:

- Proposta de solução para o SRFLP, denominada GRASP-F, utilizando a meta-heurística GRASP (*Greedy Randomized Adaptive Search Procedure*), sendo propostas as seguintes inovações: a adição de uma fase de diversificação e a aplicação de técnicas para acelerar o cálculo do ganho na função objetivo, nos movimentos de vizinhanças da busca local, o que possibilitou a resolução de instâncias de grande porte do SRFLP com até 1000 facilidades. Até então, havia somente um trabalho na literatura que considerava problemas desse porte;
- A proposta de solução, denominado PSO-DRLP, é definhada para o DRLP com estratégias heurísticas utilizando um PSO (*Particle Swarm Optimization*) de duas fases e hibridizada com uma busca local. O PSO-DRLP organiza o exame de partícula em

subpopulações. A primeira fase busca as melhores partículas em cada subpopulação que compõem uma população elite a ser otimizada pela segunda fase do algoritmo;

- O algoritmo proposto para o PROP, denominado *Adaptive Iterated Local Search* (AILS), é implementado de forma inovadora, apresentando as seguintes características: (a) o algoritmo utiliza duas fases, Fase 1 e Fase 2, que consideram aspectos de intensificação e diversificação, respectivamente, de acordo com seus critérios de aceitação distintos; (b) o algoritmo alterna entre as duas fases de forma autônoma; (c) o algoritmo regula automaticamente a intensidade da perturbação em cada fase;
- Uma solução derivada do AILS é proposta para o bCAP. O algoritmo, denominado AILS-bCAP, executa duas fases com comportamentos distintos, dados pelos critérios de aceitação e intensidade da perturbação diferentes em cada fase;
- Melhoria na técnica de Guan e Lin (2016), permitindo acelerar o cálculo do ganho no valor da função objetivo no movimento de vizinhança por inserção para o SRFLP;
- Adaptação de uma técnica de Kothari e Ghosh (2013b) para o PROP a fim de acelerar o cálculo do ganho no movimento de vizinhança. Essa técnica adaptada é usada pelo procedimento de busca local no algoritmo AILS;
- Adaptação das técnicas para acelerar o cálculo do ganho na função objetivo nos movimentos vizinhança por troca (KOTHARI; GHOSH, 2013b) e nos movimentos de inserção (GUAN; LIN, 2016, CRAVO; AMARAL, 2019) para o bCAP são usadas na busca local proposta para o AILS-bCAP;
- Publicação dos estudos relacionados ao SRFLP, sendo os resultados preliminares do estudo, durante o curso de doutorado, publicados em um trabalho no XLIX Simpósio Brasileiro de Pesquisa Operacional e a melhor versão do GRASP-F publicada em um periódico considerado de grande relevância na área de Ciência da Computação e Otimização Combinatória, avaliado como Qualis A1 (QUALIS CAPES, 2021);
- Publicação do estudo relacionado ao DRLP e os resultados obtidos com o PSO em periódico na área de Inteligência Computacional, avaliado como Qualis B2 (QUALIS CAPES, 2021); e
- Contribuição para a literatura relacionada fornecendo resultados competitivos para as instâncias dos experimentos realizados com soluções melhores para 23 das 93

instâncias do SRFLP; obtenção de soluções melhores ou iguais para 35 das 51 instâncias para o DRLP; melhores soluções foram obtidas para 49 das 100 instâncias do PROP e, para as 51 instâncias restantes, as solução foram iguais à literatura, sendo ainda consideradas instâncias inéditas com até 300 facilidades; e para o bCAP, das 76 instâncias utilizadas, considerando o extremo da fronteira de Pareto que minimiza o custo total do fluxo de material, foram encontradas soluções que dominam as soluções da literatura em 41 instâncias. Em apenas 5 instâncias as soluções foram dominadas e as 30 restantes são iguais ou indiferentes. Ao considerar o extremo que minimiza o comprimento do corredor, 66 soluções dominam as soluções da literatura e apenas 6 soluções foram dominadas, sendo as 4 restantes iguais à literatura.

1.2 ORGANIZAÇÃO DO TRABALHO

Este trabalho está organizado em 7 capítulos na seguinte ordem:

- Capítulo 1: a introdução, com os objetivos e contribuições;
- Capítulo 2: apresenta uma revisão da literatura sobre os problemas de layout em linha tratados;
- Capítulo 3: descreve as meta-heurísticas nas quais as soluções propostas se baseiam;
- Capítulo 4: descreve a solução proposta para o SRFLP e apresenta os resultados alcançados;
- Capítulo 5: apresenta solução proposta para o DRLP e seus resultados;
- Capítulo 6: descreve a solução desenvolvida para o PROP comparando os resultados com a literatura;
- Capítulo 7: tem-se o algoritmo proposto para o bCAP com os experimentos computacionais realizados; e
- Capitulo 8: são apresentadas as conclusões e recomendações para trabalhos futuros.

Ao final, tem-se as referências bibliográficas seguido do Apêndice A, com os detalhes da melhoria no cálculo do ganho no movimento de inserção e o Apêndice B, onde é mostrado a implementação de uma heurística para calibração dos algoritmos dessa tese.

2 REVISÃO DE LITERATURA

Neste capítulo é apresentada uma revisão da literatura relacionada aos problemas de layout de facilidades em linha tratados nesta tese.

Nos problemas de layout de facilidades (FLP, do inglês, *Facility Layout Problem*) deseja-se determinar a disposição das facilidades em um determinado layout. Os problemas podem ser categorizados em diferentes tipos, dependendo dos fatores observados (DRIRA; PIERREVAL; HAJRI-GABOUJ, 2007). Hosseini-Nasab et al. (2018) apresentam um sistema de classificação dos FLPs, baseado na revisão de 186 publicações realizadas nos anos de 1987 a 2016, em que os problemas são categorizados conforme as características físicas da instalação e como os problemas são abordados, sendo pela forma e dimensionamento, sistemas de manufatura, sistema de manuseio de materiais e fluxo de movimentação. Enquanto Pérez-Gosende, Mula e Díaz-Madroñero (2021) propõem a inclusão dos critérios de classificação como tipo de problema, abordagem e fase de planejamento, características das instalações de produção, configuração do sistema de manuseio de materiais e métodos para gerar e avaliar alternativas de layout, sendo avaliados 232 artigos publicados nos anos de 2010 a 2019.

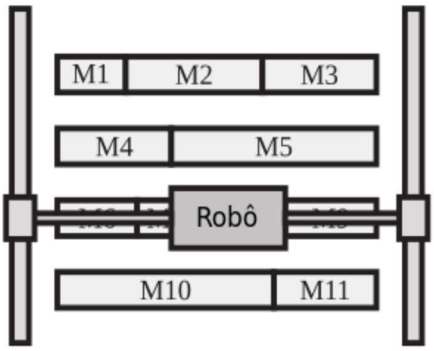
Uma das mais importantes categorias dos FLPs é a que aborda os problemas relacionados à configuração do manuseio de materiais, tendo como grande desafio os problemas de layout em linha (RLP), pois tratam-se de problemas de natureza combinatória e NP-difíceis, o que evidencia a dificuldade prática de resolvê-los de forma exata em tempo computacional razoável (YANG et al, 2019).

O mais geral dos RLP é o denominado problema de layout em múltiplas linhas (MRLP). Normalmente, uma instância do MRLP consiste de um conjunto de n facilidades retangulares com comprimento l_1, ..., l_n, um dado número de linhas $k > 1$ e os fluxos de materiais entre cada par de facilidades c_{ij}. O que se pretende é encontrar uma associação de facilidades para cada linha e as posições das facilidades em cada uma das linhas, tal que a multiplicação dos fluxos pelas distâncias, centro-a-centro, entre todas os pares de facilidades, seja minimizada (HUNGERLÄNDER; ANJOS, 2015). De modo geral, são consideradas somente as distâncias entre as facilidades em relação ao eixo-x no plano cartesiano.

O MRLP tem inúmeras aplicações tais como os problemas de organizar edifícios em um campus universitário (DICKEY; HOPKINS, 1972), layouts hospitalares (ELSHAFEI, 1977), organização de máquinas em linhas paralelas num sistema de manufatura flexível (HERAGU; KUSIAK, 1988), disposição de componentes em computadores (STEINBERG, 1961), layouts

com múltiplos andares com elevador (KOCHHAR, 1998), design de teclado para escrita (POLLATSCHEK; GERSHONI; RADDAY, 1976), entre outros. A Figura 2 mostra como seria o layout de um FMS considerando o MRLP.

Figura 2 – Robô em um FMS



Fonte: adaptado de Hungerländer e Anjos (2015).

Na Figura 2, um robô do tipo pórtico (*Gantry robot*) é posicionado sobre as máquinas (M1, M2, ... M11) e assim movimenta-se no plano cartesiano, controlando o manuseio de materiais entre as máquinas.

Ao considerar o número de linhas $k = 2$, o MRLP reduz-se a uma categoria de problemas denominados layout em linha dupla. São exemplos desses problemas o DRLP, o CAP e o PROP.

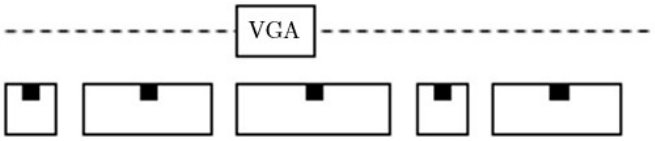
Mais detalhes sobre o MRLP, incluindo propostas de resolução, podem ser vistos em Pérez-Gosende, Mula e Díaz-Madroñero (2021), Hosseini-Nasab et al. (2018), Anjos, Fischer e Hungerländer (2015), Tubaileh e Siam (2017), Cellin (2017), Permanhane (2016), Hungerländer e Anjos (2015), Drira, Pierreval e Hajri-Gabouj (2007) e Gen, Ida e Cheng (1995).

Nas seções seguintes serão mostrados detalhes do SRFLP, dos problemas de layout em linha dupla (DRLP e PROP) e a versão biobjetivo do CAP, incluindo as soluções encontradas na literatura recente e por fim, são mostrados exemplos com soluções factíveis para cada um dos problemas tratados nesta tese.

2.1 PROBLEMA DE LAYOUT DE FACILIDADES EM LINHA ÚNICA - SRFLP

No SRFLP o que se deseja é definir um arranjo de facilidades, de comprimentos diversos, ao longo de uma linha reta com o objetivo de organizar as facilidades, de modo a minimizar a soma ponderada das distâncias entre todos os pares de facilidades (AMARAL, 2006). As aplicações do SRFLP incluem o arranjo de departamentos ao longo de um corredor em supermercados, hospitais ou escritórios (SIMMONS, 1969), o arranjo de livros em uma prateleira (PICARD; QUEYRANNE, 1981) e o projeto de sistemas de manufatura (HERAGU; KUSIAK, 1988), entre outros. Um exemplo de aplicação do SRFLP em uma linha de produção é mostrado na Figura 3.

Figura 3 – Esquema de arranjo de facilidades em linha única



Fonte: Cravo e Amaral (2017).

Na Figura 3, o VGA (Veículo guiado automaticamente) desloca-se em linha reta e cada retângulo representa uma estação de trabalho, onde o veículo deverá entregar ou coletar ferramentas e/ou equipamentos.

O problema de layout de facilidades em fila única (SRFLP) foi proposto primeiramente por Simmons (1969). Considera-se l_i, o comprimento da facilidade i. Também é dada uma matriz $C = [c_{ij}]$ de tamanho $n \times n$, onde c_{ij} é o fluxo de materiais entre as facilidades i e j. A distância entre duas facilidades é dada pela distância entre os seus centros. Formalmente, pode-se definir o problema em encontrar uma permutação $\Pi = \{\pi_1, \pi_2, ..., \pi_n\}$ do conjunto de facilidades $N = \{1, 2, ..., n\}$ que forneça o menor valor para a Equação (2.1) (KOTHARI; GHOSH, 2012).

$$\sum_{1 \le i < j \le n} c_{\pi_i \pi_j} d_{\pi_i \pi_j} \tag{2.1}$$

Onde:

$$d_{\pi_i \pi_j} = \frac{l_{\pi_i} + l_{\pi_j}}{2} + \sum_{i<k<j} l_{\pi_k} \tag{2.2}$$

A Equação (2.2) define a distância entre os centros das facilidades π_i e π_j na permutação Π.

Ao longo dos anos, o SRFLP vem recebendo considerável atenção dos pesquisadores. Simmons (1969) propôs um algoritmo *branch-and-bound*; Picard e Queyranne (1981) apresentaram um algoritmo de programação dinâmica; Amaral (2009) propôs um algoritmo de planos de corte; Hungerländer e Rendl (2012) testaram uma abordagem baseada em programação semi-definida; e Amaral e Letchford (2013) propuseram um algoritmo *branch-and-cut*.

Diferentes formulações de programação inteira estão disponíveis em Amaral (2006, 2008a), Heragu e Kusiak (1991), Love e Wong (1976). Andrade e Ferreira (2017) propõem uma melhora no modelo de Amaral (2006), apresentando um desempenho melhor em comparação com os modelos de Amaral (2006) e de Love e Wong (1976) em termos de tempos computacionais. Hungerländer e Rendl (2012) reportam soluções exatas para problemas com até 40 facilidades. Entretanto, para problema de maior porte, torna-se proibitiva a aplicação de métodos exatos, já que o SRFLP é um problema NP-Difícil (AMARAL, 2006).

Métodos para encontrar limitantes inferiores para o SRFLP são descritos em Anjos, Kennings e Vannelli (2005), Amaral (2009), Anjos e Yen (2009), Hungerländer e Rendl (2012) e Amaral e Letchford (2013).

Diversas abordagens heurísticas foram propostas para o SRFLP tais como heurística gulosa (KUMAR et al., 2008); *Simulated Annealing* (HERAGU; ALFA, 1992; DE ALVARENGA; NEGREIROS-GOMES; MESTRIA, 2000; TUBAILEH; SIAM, 2017); *Enhanced local search* (AMARAL, 2008b); Busca Tabu (DE ALVARENGA; NEGREIROS-GOMES; MESTRIA, 2000; SAMARGHANDI; ESHGHI, 2010; KOTHARI; GHOSH, 2013b); Algoritmos Genéticos (OZCELIK, 2012; DATTA; AMARAL; FIGUEIRA, 2011; KOTHARI; GHOSH, 2013a); *Scatter Search* (KUMAR et al., 2008; KOTHARI; GHOSH, 2014); Colônia de formigas (SOLIMANPUR; VRAT; SHANKAR, 2005; TUBAILEH; SIAM, 2017); *Particle Swarm Optimization* (SAMARGHANDI; TAABAYAN; JAHANTIGH, 2010); e GRASP (CRAVO; AMARAL, 2015, 2017).

Resultados interessantes foram obtidos em um estudo conduzido por Rubio-Sánchez et al. (2016). Os autores apresentam o GRASP-PR, um algoritmo que combina a meta-heurística GRASP e *Path Relinking* (PR). O GRASP-PR foi comparado com dois algoritmos da literatura: o algoritmo genético GENALGO (KOTHARI; GHOSH, 2013a) e a melhor variação do *Scatter Search* de Kothari e Ghosh (2014). Os experimentos computacionais mostraram que o GRASP-PR supera ambos os algoritmos em instâncias de até 500 facilidades.

O GRASP-PR preenche primeiro um conjunto de soluções denominadas de conjunto elite com soluções construídas pelo GRASP. Em seguida, o algoritmo gera mais soluções do GRASP, que são combinadas com as do conjunto elite por meio de um *path relinking*. Assim, as soluções elites podem ser substituídas pelas soluções obtidas a partir da aplicação do *path relinking*, melhorando a qualidade geral do conjunto elite. Duas estratégias são propostas para construir caminhos entre soluções de alta qualidade. Uma dessas abordagens favorece a diversificação dos caminhos construídos, uma variante da estratégia de *path relinking* de Kothari e Ghosh (2014) com um fator aleatório, enquanto a outra, mais sofisticada, está relacionada a resolver o problema da subsequência mais longa (*longest increasing subsequence problem*) (RUBIO-SÁNCHEZ et al., 2016).

Palubeckis (2015) introduziu técnicas de buscas locais rápidas, as quais permitiram-lhe aplicar um VNS (*Variable Neighborhood Search*) para instâncias com até 300 facilidades em tempo razoável e Guan e Lin (2016) apresentaram um algoritmo híbrido entre VNS e Colônia de formigas, fazendo uso de técnicas para acelerar o cálculo do ganho na função objetivo nos movimentos de vizinhança das buscas locais utilizadas, permitindo a resolução de instâncias com até 500 facilidades.

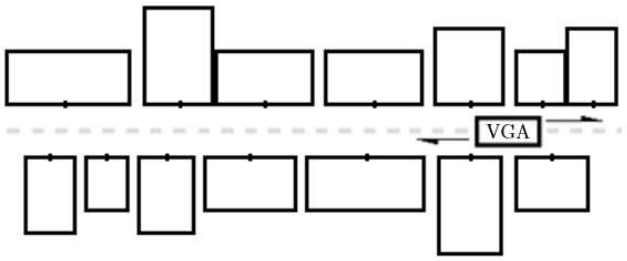
Progressos foram alcançados por Palubeckis (2017) com a proposta de um algoritmo MSA (*Multi-start Simulated Annealing*), o qual usa dois tipos de movimentos de vizinhança: troca de pares e inserção. Os movimentos utilizam-se de procedimentos rápidos para o cálculo do ganho na função objetivo. O autor apresenta duas variantes do MSA, denominadas MSA_0 e MSA_0.5. Os algoritmos obtiveram excelentes resultados para instâncias de problemas de grande porte com até 1000 facilidades, com os melhores resultados obtidos pelo algoritmo MSA_0. Somente a partir do trabalho de Palubeckis (2017), instâncias de problemas desse porte são consideradas para o SRFLP.

2.2 PROBLEMA DE LAYOUT EM LINHA DUPLA - DRLP

O DRLP é um caso especial do MRLP e as formulações são baseadas no SRFLP. No DRLP as facilidades são dispostas em ambos os lados de uma linha reta central e o sistema de manuseio de materiais desloca-se na região central, como mostra na Figura 4. Os pontos centrais de cada máquina é o local de carga e descarga de materiais pelo VGA.

O DRLP aparece na linha de produção de display de cristal líquido (CHUNG; TANCHOCO, 2010), na fabricação de semicondutores (MURRAY; SMITH; ZHANG, 2013), aplicações na construção civil, a disposição de salas e departamentos normalmente é colocado em ambos os lados de um corredor. Ferramentas também são colocadas de forma semelhante em FMS, para um melhor manuseio por um VGA e de modo a manter o uso eficiente do espaço (CHUNG; TANCHOCO, 2010). O DRLP possui uma participação significativa no processo produtivo na indústria, em termos de custo e tempo (MOHAMMADI; FORGHANI, 2016).

Figura 4 – Exemplo de arranjo de máquinas em linha dupla



Fonte: próprio autor.

O problema tem recebido menos atenção na literatura do que o SRFLP (HOSSEINI-NASAB et al., 2018). O DRLP contém aspectos combinatórios, que aparecem pela necessidade de se determinar a ordenação das facilidades no layout, e aspectos contínuos, com a necessidade de se determinar as posições absolutas das facilidades no layout. Assim, as abordagens exatas tradicionais de solução não são adequadas para obter soluções dentro de um tempo razoável. O problema vem sendo estudado por autores como Heragu e Kusiak (1988), Chung e Tanchoco (2010), Murray, Smith e Zhang (2013) e Amaral (2013b, 2018, 2021).

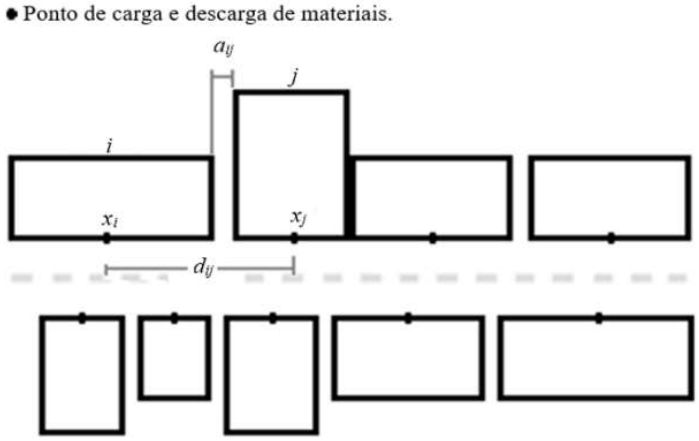
Matematicamente, o DRLP pode ser formulado como um conjunto de máquinas $N = \{1, 2, ..., n\}$, com seus respectivos comprimentos $\{l_1, ..., l_n\}$ e um fluxo não negativo c_{ij}. Em um layout viável para o DRLP, d_{ij} é a distância entre os centros dos pares (i, j) de máquinas. Dessa forma, o objetivo do DRLP é encontrar um layout viável que minimize a Equação (2.3) (AMARAL, 2018).

$$\sum_{1 \leq i < j \leq n} c_{ij} d_{ij} \tag{2.3}$$

Na Figura 5 são mostradas as variáveis envolvidas na representação do DRLP. Os pontos pretos na Figura 5 representam o ponto de carga e descarga de materiais, a variável a_{ij} representa o

espaçamento mínimo entre máquinas adjacentes, denominado folgas requeridas (*clearance*), e d_{ij} é a distância entre o par (i, j) de máquinas e x_i o centro da máquina i.

Figura 5 – Variáveis do DRLP



Fonte: próprio autor.

As folgas são utilizadas em aplicações práticas nas quais máquinas precisam de um espaço para manutenção ou operação. Heragu e Kusiak (1988) supuseram que essa folga entre cada par de máquinas fosse dada e, nesse caso, os dados para uma instância DRLP consistirão no número n de máquinas, os comprimentos das máquinas, a quantidade de fluxo c_{ij} entre cada par (i, j) de máquinas e a folga a_{ij} requerida entre cada par (i, j) de máquinas. Entretanto, os valores das folgas podem ser considerados de diversas formas. Como a folga é uma característica da máquina para uso em manutenção, nesse caso, a folga pode ser incluída no comprimento da máquina (AMARAL, 2018). Mais especificamente, quando os valores das folgas são iguais a um dado valor a, o comprimento de cada máquina pode ser aumentado pelo valor a e o problema pode ser tratado como um problema sem folgas, assim, $a_{ij} = 0$ para todo par (i, j) de máquinas (HERAGU; KUSIAK, 1988; ANJOS; VANNELLI, 2008; AMARAL, 2009; BRUNESE; TANCHOCO, 2013). Se as folgas são incluídas em uma instância, é dito que o DRLP tem folgas explícitas. No entanto, quando os valores não aparecem nos dados da instância, porque eles já foram incluídos nos comprimentos das máquinas, é dito que a instância tem folgas implícitas. Se as folgas entre as máquinas são dependentes da ordenação, não sendo todas iguais, as folgas implícitas não poderão ser utilizadas (AMARAL, 2018).

Chung e Tanchoco (2010) apresentam um modelo MIP (do inglês, *Mixed Integer Programming*) para o DRLP derivado do modelo para o SRFLP desenvolvido por Heragu e Kusiak (1991). Zhang e Murray (2012) mostraram que o modelo não estava correto quando os

valores das folgas a_{ij} não são todos zeros. Entretanto, o modelo para todo $a_{ij} = 0$ é um modelo válido para o DRLP (AMARAL, 2018).

Amaral (2013b) propôs um modelo MIP para o DRLP, sendo definidas desigualdades válidas, que são inseridas no modelo. O modelo apresenta um número menor de variáveis (tanto contínuas, quanto binárias). Com a adição das restrições definidas nas desigualdades, houve um ganho significativo em termo computacional em comparação com o modelo proposto por Chung e Tanchoco (2010).

Posteriormente, Amaral (2018) apresenta uma nova formulação via MIP para o DRLP, o qual compara com os modelos de Chung e Tanchoco (2010) e de Amaral (2013b) para instâncias com até $n = 13$ máquinas. Os resultados computacionais mostram que o modelo proposto tem um desempenho superior ao de Chung e Tanchoco (2010), sendo que, em relação ao modelo de Amaral (2013b), não apresentou, estatisticamente, diferença significativa. Outro modelo MIP para o DRLP, também baseado no modelo de Amaral (2013b), é apresentado por Secchin e Amaral (2018), que proporcionou melhora no desempenho, de modo que foi possível a resolução do problema para instâncias de até $n = 15$ máquinas.

Nas formulações MIP para o DRLP mais recentes, Chae e Regan (2020) modificam o modelo de Secchin e Amaral (2018), introduzindo novas restrições. O novo modelo melhora o desempenho, em termos de tempo de execução, para resolver problemas com até $n = 16$ máquinas; Amaral (2021) apresenta uma nova formulação MIP para o DRLP baseado em uma extensão da relação de ordem entre elementos de um conjunto, o que permitiu o uso de informações adicionais sobre a posição relativa das máquinas, diminuindo, assim, o número de variáveis binárias no modelo pela metade. O autor compara o novo modelo com modelos ditos fundamentais da literatura (AMARAL, 2013b; 2018), resolvendo instâncias com até $n = 13$ máquinas e duas variações do novo modelo são comparadas com o melhor modelo evoluído (derivado de um fundamental) (CHAE; REGAN; 2020) para instâncias com até $n = 15$ máquinas; e Dahlbeck, Fischer e Fischer (2020) apresentam limites inferiores combinatórios para instâncias com até 50 facilidades. As soluções propostas por Amaral (2013b, 2018, 2021), Secchin e Amaral (2018), Chae e Regan (2020) e Dahlbeck, Fischer e Fischer (2020) não consideram as folgas explícitas, ou seja, são válidos para o DRLP apenas quando $a_{ij} = 0$.

Sendo o DRLP classificado como NP-difícil, ocorre que a solução exata em tempo razoável é uma tarefa muito difícil. Assim, o uso de métodos heurísticos também se faz necessário quando se pretende resolver instâncias de grande porte. Entretanto, não há na literatura muitas pesquisas

com abordagens com estratégias heurísticas para o DRLP (HUNGERLÄNDER; ANJOS, 2015). As abordagens heurísticas normalmente utilizam-se de modelagens matemáticas para determinar as variáveis contínuas do problema. O Quadro 1 apresenta as soluções mais recentes propostas para o DRLP que tratam de problemas com $n > 15$ máquinas, tendo na primeira coluna os nomes dos autores, seguida da coluna com a abordagem utilizada.

Quadro 1 – Métodos heurísticos da literatura para o DRLP

Autores	**Soluções propostas**
Chung e Tanchoco (2010)	Heurísticas construtivas com Programação Linear
Murray, Smith e Zhang (2013)	Heurísticas construtivas com MIP
Hungerländer e Anjos (2015)	Relaxação de programação semi-definida
Permanhane (2016)	Iterated Local Search (ILS) com MIP
Cellin (2017)	Variable Neabourhood Search (VNS) com MIP, VNS/ILS com MIP
Guan et al. (2020)	*Decomposition-based Algorithm* (DBA)
Amaral (2020)	Heurísticas de melhoria com Programação Linear

Fonte: próprio autor.

No Quadro 1, os estudos utilizam-se de métodos exatos para a resolução das variáveis contínuas definidas no DRLP, com exceção do algoritmo DBA (GUAN et al., 2020). Chung e Tanchoco (2010) propõem heurísticas construtivas para o DRLP, onde o processo iterativo é guiado por uma das cinco regras, denominadas MinLCF (*minimum location cost first*), MinFF (*minimum flow first*), MaxFF (*maximum flow first*), MinWF (*minimum width first*), e MaxWF (*maximum width first*). Após a determinação da ordem das máquinas no layout pela heurística construtiva, as posições em cada linha são determinadas por um modelo de programação linear. Hungerländer e Anjos (2015) resolvem o DRLP via relaxação do modelo de programação semi-definida e obtêm a viabilidade utilizando uma heurística proposta por Anjos e Hungerländer (2012), não considerando espaços entre as máquinas. Após uma solução viável ser obtida, a determinação das posições das máquinas com a adição de espaços de comprimento 0,5 à solução (máquinas "artificiais" de comprimento 0,5 e fluxo de custo zero). Murray, Smith e Zhang (2013) fazem uma combinação de heurísticas construtivas e de busca local com o uso de soluções obtidas por um modelo MIP para máquinas na mesma linha. Os autores consideram valores explícitos das folgas ($a_{ij} \neq 0$) e matriz de fluxos assimétrica. Permanhane (2016) resolve o DRLP utilizando a meta-heurística ILS para fazer o posicionamento relativo nas linhas, fazendo uso de um modelo de MIP para fazer o posicionamento das facilidades no eixo-x. Cellin

(2017) apresenta uma heurística VNS e uma hibridização do VNS com a meta-heurística ILS. Em ambas estratégias, as posições das máquinas, ao longo do corredor, são determinadas pela resolução do modelo MIP. Tais estudos tratam o DRLP com instâncias de até 50 máquinas.

Guan et al. (2020) propuseram um algoritmo denominado DBA que resolve o DRLP considerando a decomposição do mesmo em dois subproblemas. O primeiro subproblema considera a parte combinatória do DRLP, que consiste na ordenação das máquinas, desconsiderando as folgas entre as mesmas. O segundo subproblema consiste em determinar as folgas para a solução encontrada no primeiro subproblema, ou seja, resolve-se as variáveis contínuas do DRLP, determinando os valores das abscissas dos centros das máquinas ao longo do eixo-x. No DBA, inicialmente é gerado aleatoriamente um arranjo de máquinas considerando uma única linha. Em seguida, encontra-se um ponto de quebra para definir os arranjos em ambos os lados do corredor do DRLP e assim, calcula-se os valores iniciais das abscissas em ambas as linhas. Com a solução corrente, o algoritmo realiza três processos: aplicação de uma busca local com movimentos de troca 2-opt; determinação das folgas entre as máquinas, utilizando uma meta-heurística PSO, assim encontrando a solução completa para o DRLP; e por fim, a solução corrente discreta, desconsiderando as folgas, é perturbada para a execução da próxima iteração do algoritmo. Nos testes computacionais, os autores consideram problemas com matriz de fluxos assimétrica, onde o fluxo de materiais entre duas facilidades depende das suas posições relativas no arranjo, e valores explícitos das folgas para algumas instâncias dos testes realizados, como Murray, Smith e Zhang (2013).

Recentemente, Amaral (2020) apresenta quatro algoritmos para resolver o DRLP. O autor propõe uma abordagem em duas fases semelhante a Guan et al. (2020). Na primeira fase é resolvido o problema de layout desconsiderando os espaços entre as máquinas no arranjo, definindo assim a ordem das máquinas no layout. No segundo momento, as soluções encontradas pelas heurísticas são melhoradas utilizando o modelo MIP de Amaral (2018), onde são definidas as posições absolutas das máquinas no layout. O autor trata problemas com tamanho $n = 50$ máquinas, considerando valores implícitos e explícitos para as folgas requeridas e matriz de fluxos simétrica e assimétrica.

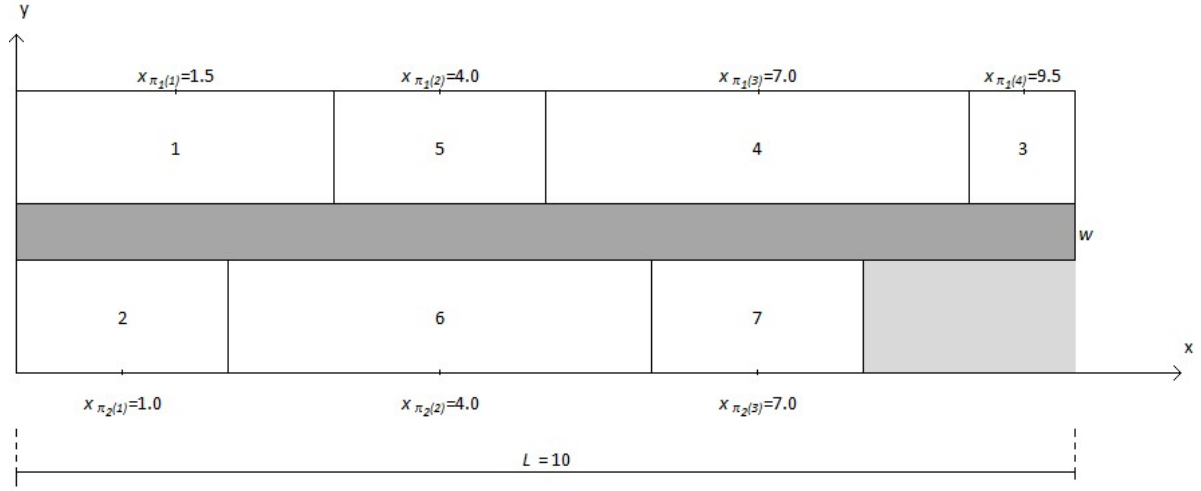
2.3 PROBLEMA DE ALOCAÇÃO DE CORREDOR BIOBJETIVO - bCAP

O problema de alocação em corredor biobjetivo (bCAP) é uma extensão do CAP. No CAP o que se deseja é encontrar um arranjo de n departamentos ao longo dos dois lados de um corredor. O objetivo no CAP é a minimização do custo total de comunicação entre os departamentos, considerando as duas condições: (1) não é permitido espaço entre dois departamentos adjacentes e (2) os pontos mais à esquerda dos arranjos, em ambas as linhas do corredor, devem começar em um ponto em comum (MAADI; JAVIDNIA; JAMSHIDI, 2017). As aplicações do CAP incluem arranjo de salas em edifícios comerciais, hospitais, shoppings e escolas (MAADI; JAVIDNIA; JAMSHIDI, 2017; AMARAL, 2012).

Em uma instância padrão do CAP é dado l_i, o comprimento de cada facilidade i e uma matriz $C = [c_{ij}]$ de fluxos não negativos entre os pares de facilidades i e j. A distância entre duas facilidades i e j é calculada pela diferença entre os seus centros no eixo-x, sendo, tal como no DRLP, desprezada a largura do corredor. Assim, o custo total é a soma ponderada das distâncias entre cada par de facilidades com os pesos c_{ij} (AHONEN; DE ALVARENGA; AMARAL, 2014). A Figura 6 mostra um exemplo de um arranjo viável para o CAP com 7 facilidades, sendo l_1=3, l_2=2, l_3=1, l_4=4, l_5=2, l_6=4 e l_7=2.

Na Figura 6, $x_{\pi_r(i)}$ é a abscissa do centro da facilidade da posição i no arranjo da linha $r \in \{1, 2\}$. A distância entre as facilidades 1 e 3, por exemplo, é dada pela diferença posições $x_{\pi_1(1)}$ e $x_{\pi_1(4)}$. L e w são o comprimento e a largura do corredor, respectivamente.

Figura 6 – Exemplo de arranjo para o CAP



Fonte: próprio autor.

Amaral (2012) apresenta uma formulação para um layout do CAP representado por uma permutação $\pi = [\pi_1, \pi_2]$, onde π_1 e π_2 são os arranjos em cada linha paralela. Na Figura 6, $\pi_1 = [1, 5, 4, 3]$ e $\pi_2 = [2, 6, 7]$. Assim, considerando a seguinte notação:

- n: o número fixo de facilidades alocadas ao longo do corredor;
- $c_{\pi_r(i)\pi_s(j)}$: o fluxo entre a facilidade i da linha $r \in \{1, 2\}$ e a facilidade j da linha $s \in \{1, 2\}$;
- $x_{\pi_r(i)}$: a abscissa do centro da facilidade i da linha $r \in \{1,2\}$; e
- Π_n: o conjunto de todas as permutações π de $N = \{1, 2, ..., n\}$.

Então, a formulação matemática do CAP é dada por:

$$\min_{\{\pi_1,\pi_2\}:[\pi_1,\pi_2]\in\Pi_n} \left\{ \begin{array}{l} \sum_{i=1}^{|\pi_1|-1} \sum_{j=i+1}^{|\pi_1|} c_{\pi_1(i),\pi_1(j)} \left| x_{\pi_1(i)} - x_{\pi_1(j)} \right| + \\ \sum_{i=1}^{|\pi_2|-1} \sum_{j=i+1}^{|\pi_2|} c_{\pi_2(i),\pi_2(j)} \left| x_{\pi_2(i)} - x_{\pi_2(j)} \right| + \\ \sum_{i=1}^{|\pi_2|} \sum_{j=1}^{|\pi_2|} c_{\pi_1(i),\pi_2(j)} \left| x_{\pi_1(i)} - x_{\pi_2(j)} \right| \end{array} \right\} \tag{2.4}$$

Onde:

$$x_{\pi_{r(j)}} = \frac{l_{x_{\pi_{r(j)}}}}{2} + \sum_{i=1}^{j-1} l_{\pi_{r(j)}}, \; r \in \{1, 2\}; \; 1 \leq j \leq n \tag{2.5}$$

Na formulação, a Equação (2.4) define o custo do manuseio de materiais ou da comunicação entre as facilidades que estão na mesma linha e entre as linhas paralelas. A Equação (2.5) define o centro de cada facilidade, garantindo que não haja espaço entre facilidades adjacentes e que não sejam sobrepostas, sendo $l_{\pi_{r(j)}}$ o comprimento da facilidade da posição j da linha $r \in \{1, 2\}$. Além disso, a restrição da Equação (2.5) implica que o ponto mais à esquerda do arranjo no eixo-x, em ambas as linhas, seja zero.

Argumentando que em algumas aplicações os comprimentos dos corredores gerados em um layout do CAP podem não corresponder com a realidade, Kalita e Datta (2014) apresentam uma variação do CAP, denominado Problema de Alocação de Corredor Biobjetivo (bCAP, do inglês *Bi-objective Corridor Allocation Problem*). O bCAP foi proposto como um problema de otimização biobjetivo não linear e irrestrito que minimiza (f_1) o custo total de manuseio de

materiais entre as facilidades dispostas nas duas linhas paralelas em ambos os lados do corredor e (f_2) a diferença entre os comprimentos dos arranjos nos dois lados do corredor, implicando na minimização do comprimento total do corredor (KALITA; DATTA; PALUBECKIS, 2019).

Outros problemas de layout em linha consideram mais de uma função objetivo, como por exemplo, o DDRLP (do inglês, *Dinamic Double Row Layout Problem*), em que os fluxos de materiais mudam ao longo do tempo em diferentes períodos de processamento, sendo considerada, assim, a minimização do custo total de manuseio de materiais e o custo de reorganização do layout (WANG et al., 2015); o EDRLP (do inglês, *Extended Double Row Layout Problem*), que considera a minimização tanto do custo total de manuseio de materiais quanto a área do layout (ZUO; MURRAY; SMITH, 2014); e o cbCAP (do inglês, *Constrained Bi-objective Corridor Allocation Problem*), uma versão do bCAP com a restrição de que algumas facilidades são fixadas, à priori, em uma das linhas do layout, similar ao PROP, com os objetivos de minimizar o custo total de manuseio de materiais e o comprimento do corredor (KALITA; DATTA, 2020).

Kalita, Datta e Palubeckis (2019) propõem a formulação do bCAP similar à formulação de Kalita e Datta (2014). A nova formulação altera a expressão da segunda função objetivo, passando a minimizar o desvio percentual do comprimento do corredor em relação ao comprimento mínimo teórico. Considerando a permutação $\pi = [\pi_1, \pi_2]$, onde π_1 e π_2 são os arranjos em cada linha paralela, os parâmetros e variáveis da formulação do problema foram definidos como segue:

- n: o número fixo de facilidades alocados ao longo do corredor;
- $c_{\pi_r(i)\pi_s(j)}$: o fluxo entre as facilidades i da linha $r \in \{1, 2\}$ e o facilidade j da linha $s \in \{1, 2\}$;
- $x_{\pi_r(i)}$: a abscissa do centro da facilidade i da linha $r \in \{1, 2\}$;
- Π_n: o conjunto de todas as permutações π de $N = \{1, 2, ..., n\}$;
- $e_{i,i+1}$: folga requerida entre as facilidades adjacentes i e i+1;
- w: largura do corredor;

Então, a formulação matemática do bCAP é dada por:

$$f_1: \min_{\{\pi_1,\pi_2\}:[\pi_1,\pi_2]\in\Pi_n} \left\{ \begin{array}{l} \sum_{i=1}^{|\pi_1|-1} \sum_{j=i+1}^{|\pi_1|} c_{\pi_1(i),\pi_1(j)} \left| x_{\pi_1(i)} - x_{\pi_1(j)} \right| + \\ \sum_{i=1}^{|\pi_2|-1} \sum_{j=i+1}^{|\pi_2|} c_{\pi_2(i),\pi_2(j)} \left| x_{\pi_2(i)} - x_{\pi_2(j)} \right| + \\ \sum_{i=1}^{|\pi_1|} \sum_{j=1}^{|\pi_2|} c_{\pi_1(i),\pi_2(j)} \sqrt{\left(x_{\pi_1(i)} - x_{\pi_2(j)}\right) + w^2} \end{array} \right\} \quad (2.5)$$

$$f_2: min \left(\frac{L^{\pi} - L}{L} \times 100 \right) \quad (2.6)$$

Onde:

$$L^{\pi} = max \left\{ \left(x_{\pi_1(|\pi_1|)} + \frac{l_{\pi_{1(|\pi_1|)}}}{2} \right), \left(x_{\pi_2(|\pi_2|)} + \frac{l_{\pi_{2(|\pi_2|)}}}{2} \right) \right\} \quad (2.7)$$

$$L = \frac{1}{2} \sum_{i=1}^{n} l_i \quad (2.8)$$

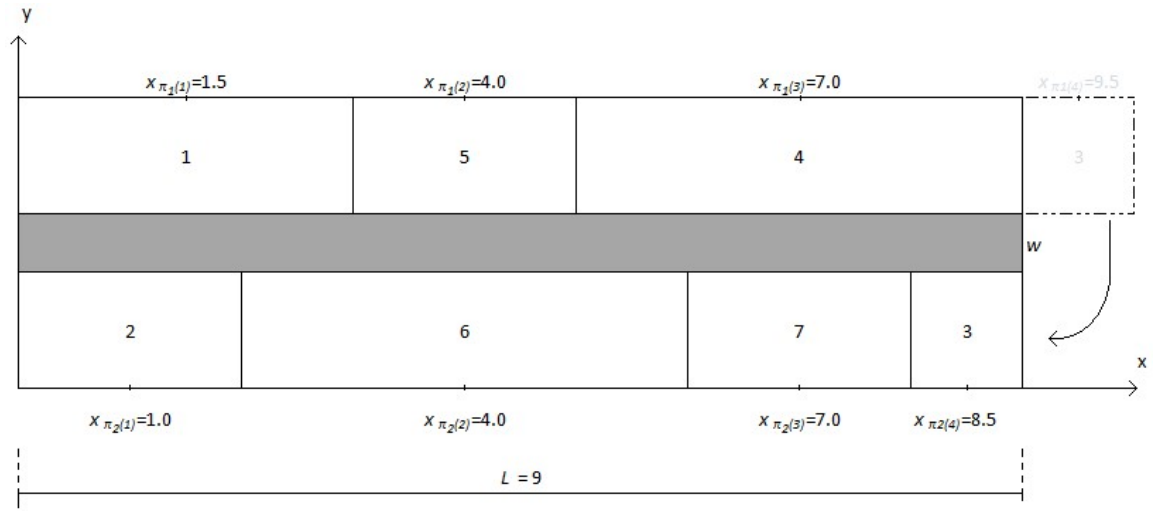
$$x_{\pi_{r(p)}} = \frac{l_{x_{\pi_{r(1)}}}}{2} + \sum_{k=2}^{p} \left(l_{\pi_{r(k-1)}} + e_{x_{\pi_{r(k-1)}} x_{\pi_{r(k)}}} + l_{\pi_{r(k)}} \right), \; r \in \{1,2\}; \; 1 \leq p \leq n \quad (2.9)$$

A Equação (2.5) define a função objetivo que minimiza o custo total de manuseio de materiais entre as facilidades do arranjo π. O primeiro e o segundo termo correspondem ao custo do manuseio de materiais entre as facilidades que estão nos arranjos das linhas 1 e 2, respectivamente. O terceiro termo é o custo de manuseio de materiais entre as facilidades em linhas opostas, onde a distância entre as facilidades é dada pela hipotenusa do triângulo retângulo formado pela distância $\left| x_{\pi_r(i)} - x_{\pi_r(j)} \right|$ e a largura do corredor w. Deve-se notar que $w \neq 0$, na Equação (2.5), torna a formulação não linear, fazendo com que qualquer método de programação inteira mista seja inadequado para manipulá-lo (KALITA; DATTA; PALUBECKIS, 2019). Entretanto, na literatura, a largura do corredor é desconsiderada, ou seja, $w = 0$. Também pode-se observar que, caso $|\pi_1| = n$, tem-se a função objetivo da formulação do SRFLP.

A Equação (2.6) minimiza o desvio percentual do comprimento do corredor em relação ao comprimento mínimo teórico L, tomado como a metade do comprimento total de todas as facilidades, dado pela Equação (2.8). O comprimento L^{π}, do corredor em um arranjo π, é dado pelo valor máximo entre os comprimentos de π_1 e π_2, conforme Equação (2.7).

A Figura 7 exemplifica um layout para o bCAP, com 7 facilidades, em que o objetivo f_2 é melhorado em relação ao layout da Figura 6 ao considerar a facilidade 3, que estava na posição $x_{\pi_1(4)}$, em uma nova posição $x_{\pi_2(4)}$, em que o comprimento do corredor passa a ser $L = l_1 + l_5 + l_4 = 9$. Assim, o novo arranjo na Figura 7 tem o valor de $f_2 = 0\%$, melhorando em relação ao layout da Figura 6 que era $f_2 = 11{,}11\%$.

Figura 7 – Exemplo de arranjo para o bCAP



Fonte: próprio autor.

Para o CAP, soluções exatas são apresentadas em Amaral (2012) e Fischer, Fischer e Hungerländer (2019); limitantes inferiores são apresentados via programação semi-definida em Anjos e Hungerländer (2012); soluções heurísticas também estão disponíveis na literatura e incluem três heurísticas de Amaral (2012), que resolvem instâncias com até 30 facilidades; um Algoritmo Genético hibridizado com busca local e um *Scatter Search* com *path-relinking* para instâncias com até 49 facilidades é encontrado em Ghosh e Kothari (2012) e as metaheurísticas *Tabu Search* e *Simulated Annealing* para a solução de problemas com até 70 facilidades podem ser vistas em Ahonen, De Alvarenga e Amaral (2014).

Considerando o bCAP, Kalita e Datta (2014) apresentam uma customização do algoritmo *permutation-based Genetic Algorithm* (pGA) proposto por Datta, Amaral e Figueira (2011) para o SRFLP. O pGA adaptado inicializa os indivíduos (soluções) considerando somente uma linha como o SRFLP, formando uma população de tamanho *N*. Em seguida, avalia os indivíduos com uma solução bCAP, dividindo em duas linhas. Para a classificação da população, é criado um *rank* de não-dominância, considerando soluções em uma fronteira de Pareto, e, para manter

a diversificação das soluções, uma medida de densidade para cada solução em relação às demais soluções da população é definida. O algoritmo itera criando um *pool* de tamanho *N*, usando o operador de seleção de torneio para realizar o cruzamento dos indivíduos e, assim, gerar uma população de descendentes de tamanho *N*, usando um operador de crossover. O algoritmo explora a vizinhança, diversificando as soluções, aplicando um operador de mutação, dada uma probabilidade. Em seguida, avalia a solução do bCAP dividindo o descendente em duas linhas. Os novos indivíduos formam a população para a próxima iteração do pGA, sendo aplicado um operador de elitismo. O algoritmo repete um número predefinido de gerações. O pGA é aplicado a instâncias de problemas com até 80 facilidades.

Kalita, Datta e Palubeckis (2019) apresentam melhorias no algoritmo pGA, propondo uma hibridização com uma busca local com movimentos de inserção. A busca local utiliza uma técnica, inicialmente usada para o SRFLP em Palubeckis (2015), para acelerar a avaliação do ganho na função objetivo do custo do manuseio de materiais nos movimentos de inserção. O algoritmo aplica a busca local em soluções geradas após as operações de crossover e mutação. Os autores apresentam soluções para instâncias com 60 a 80 facilidades, comparando os resultados com Ahonen, De Alvarenga e Amaral (2014) e Kalita e Datta (2014), e passam a tratar problemas ainda maiores, com tamanhos de 100 a 300 facilidades.

2.4 PROBLEMA DE ORDENAÇÃO EM LINHAS PARALELAS - PROP

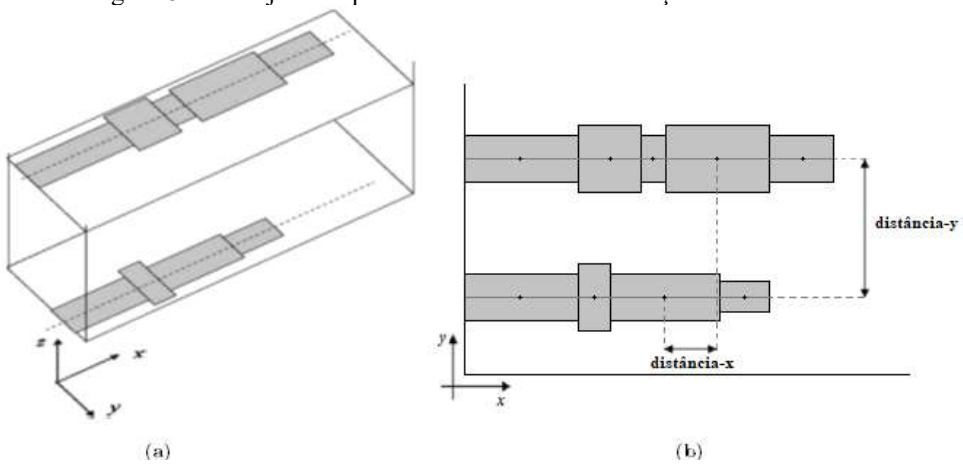
O PROP foi formalmente proposto por Amaral (2013a) e, assim como o CAP, consiste em arranjar *n* departamentos ao longo de duas linhas paralelas. Entretanto, os departamentos são previamente alocados em cada linha. Assim, supõe-se que existam *t* facilidades (1, ..., *t*) que, por alguma característica, devam ser atribuídas a uma mesma linha (linha 1) e as (*n* - *t*) facilidades restantes, na outra linha (linha 2). As facilidades colocadas em cada linha não mudam (apenas a ordenação em cada linha). Dessa forma, o PROP busca uma ordenação π^1 das *t* facilidades da linha 1 e uma ordenação π^2 das (*n* - *t*) facilidades da linha 2.

As aplicações do PROP incluem o arranjo de salas ao longo de um corredor em uma planta baixa na construção de edifícios de múltiplos andares (AMARAL, 2013a), mas nesses casos, devendo respeitar a exigência de alocação prévia de departamentos em determinado andar, como por exemplo, a preferência por máquinas pesadas, estacionamentos, salas para portadores

de necessidades especiais, entre outros, em um determinado piso. A Figura 8 exemplifica um arranjo de departamentos em uma construção em dois andares.

A Figura 8(a) mostra os departamentos ordenados em cada andar em perspectiva e a Figura 8(b) apresenta as linhas das cotas destacando as distâncias entre os departamentos em uma vista "frontal" da Figura 8(a). Como é possível notar, não há espaços entre as facilidades e as distâncias entre cada par (i, j) de facilidades são dadas pelas distâncias dos seus centros, pontos pretos, na Figura 8(b). Somente a distância no eixo-x entre os departamentos tem sido considerada.

Figura 8 – Arranjo de departamentos em uma construção de dois andares.



Fonte: adaptado Amaral (2013a).

Para formular o PROP, Amaral (2013a) define Π_t^1 como sendo o conjunto de todas as permutações π^1 de $N_1 = \{1, ..., t\}$ e Π_{n-t}^2 como o conjunto de todas as permutações de π^2 de $N_2 = \{t + 1, \ldots, n\}$. Então, o PROP pode ser matematicamente definido pela Equação (2.6).

$$\min_{\pi^1 \in N_t^1, \pi^2 \in N_{n-t}^2} \left\{ \left(\sum_{i,j \in N_1; i<j} c_{ij} d_{ij}^{\pi^1} \right) + \left(\sum_{i,j \in N_2; i<j} c_{ij} d_{ij}^{\pi^2} \right) + \left(\sum_{i \in N_1, j \in N_2; i<j} c_{ij} d_{ij}^{\pi^1, \pi^2} \right) \right\} \qquad (2.6)$$

Onde $d_{ij}^{\pi^1}$ é a distância entre as facilidades $i, j \in N_1$ em relação a permutação π^1; $d_{ij}^{\pi^2}$ é a distância entre as facilidades $i, j \in N_2$ em relação a permutação π^2; e $d_{ij}^{\pi^1, \pi^2}$, a distância entre as facilidades $i \in N_1$ e $j \in N_2$ em relação as permutações π^1 e π^2.

Na Equação (2.6), os dois primeiros somatórios representam a equação que define a função objetivo do SRFLP; se considerar $t = n$, o PROP é reduzido ao SRFLP.

O PROP possui poucos trabalhos na literatura. Formulação exata é apresentada em Amaral (2013a), que propõe um modelo de programação linear inteira mista que resolve problemas com até 23 facilidades. Yang et al. (2019) propuseram um modelo de programação inteira mista para o PROP. O modelo apresentado é baseado no modelo proposto para o SRFLP de Andrade e Ferreira (2017) e é capaz de resolver problemas com até 23 facilidades com tempos computacionais inferiores ao modelo proposto por Amaral (2013a). Os autores ainda apresentam limites inferiores e superiores para problemas com até 30 facilidades.

Como o PROP é um problema NP-Difícil (AMARAL, 2013a), a tratativa do mesmo com abordagens exatas para problemas de grande porte são computacionalmente inviáveis. Soluções heurísticas são propostas em Hungerländer e Anjos (2015), com uma abordagem via relaxação de um modelo de programação semi-definida, com a adição de uma heurística para encontrar a viabilidade da solução utilizada para resolução de diversos problemas de layout em linha, incluindo o PROP, onde os autores reportam limitantes inferiores e superiores para problemas com até 70 facilidades. Cellin (2017) propõe um *Simulated Annealing* e um algoritmo híbrido denominado VNS/ILS, sendo que o *Simulated Annealing* apresenta um desempenho superior em relação ao VNS/ILS, tanto nos resultados quanto nos tempos computacionais, nos testes realizados pelo autor.

Maadi, Javidnia e Jamshidi (2017) propõem duas estratégias para a resolução do PROP sendo um Algoritmo Genético e uma solução denominada PSA (*Population-Based Simulated Annealing Algorithm*) que é uma extensão do *Simulated Annealing* padrão. Os autores apresentam as comparações entre os dois algoritmos para instâncias de até 70 facilidades e os resultados mostram que o PSA é superior ao Algoritmo Genético nos testes realizados, sendo o melhor algoritmo para o PROP, até a presente data, com resultados publicados.

2.5 EXEMPLO DE INSTÂNCIAS E SOLUÇÕES PARA O SRFLP, DRLP, PROP e bCAP

Como dito, os dados para uma instância do problema de layout consistem da informação dos comprimentos l_i de cada facilidade i e a matriz de custo com os elementos c_{ij} representando o fluxo de materiais entre os pares de facilidades i e j. A Figura 9 ilustra instância com 5 facilidades tal como encontrada na literatura.

Na Figura 9, o valor 5 mostrado na linha 1 da listagem define o tamanho da instância, os valores na segunda linha são os comprimentos de cada uma das facilidades e a matriz simétrica contém os valores c_{ij} dos fluxos de materiais entre as facilidades.

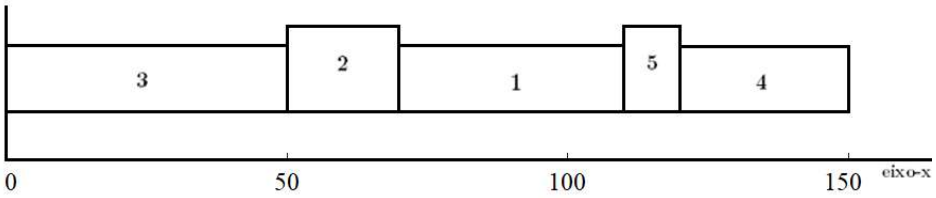
Figura 9 – Instância com 5 facilidades

```
1:  5
2: 40 20 50 30 10
3:  0  5  2  4  1
4:  5  0  3  0  2
5:  2  3  0  0  0
6:  4  0  0  0  5
7:  1  2  0  5  0
```

Fonte: próprio autor.

Assim, para o SRFLP, uma solução viável é uma lista de facilidade, por exemplo, o arranjo S = [3 2 1 5 4] representa uma solução para o problema da Figura 9 com custo total igual a 800,00. A visualização da solução é mostrada na Figura 10.

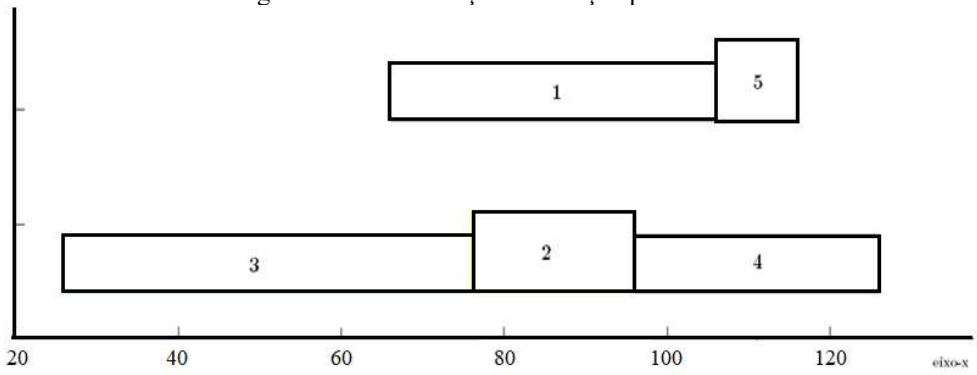
Figura 10 – Visualização da solução para SRFLP



Fonte: próprio autor.

Para o DRLP, um arranjo é representado em duas linhas e a solução $S = [L_1\ L_2]$, com L_1 = [3 (51,0) 2 (86,0) 4 (111,0)] e L_2 = [1 (86,0) 5(111,0)], sendo que os valores dentro dos parênteses são as posições absolutas no eixo-x das facilidades, com custo total igual a 350,00 e pode ser vista na Figura 11.

Figura 11 – Visualização da solução para DRLP

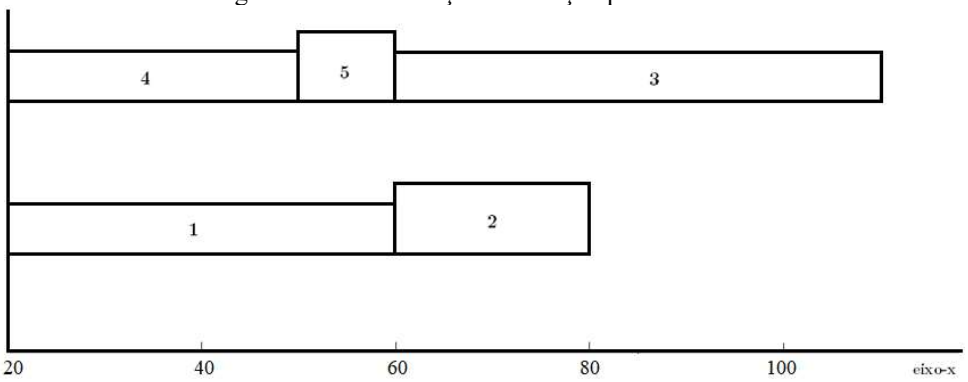


Fonte: próprio autor.

Na solução demonstrada, as facilidades estão alocadas adjacentes umas às outras, sem espaço entre si, mas não é obrigatoriedade no DRLP. Outro ponto a observar é que o início de cada linha (facilidade mais à esquerda), como pode ser visto, não começa em um ponto em comum.

No caso do PROP, o arranjo também é representado em duas linhas. A solução $S = [L_1\, L_2]$ sendo $L_1 = [1\ 2]$ e $L_2 = [4\ 5\ 3]$, com custo total de 450,00 é mostrada na Figura 12.

Figura 12 – Visualização da solução para o PROP

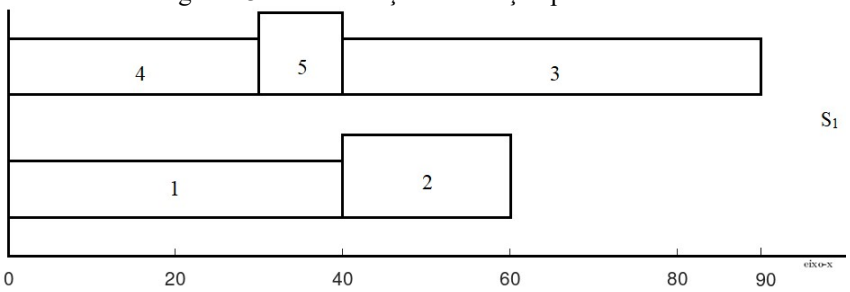


Fonte: próprio autor

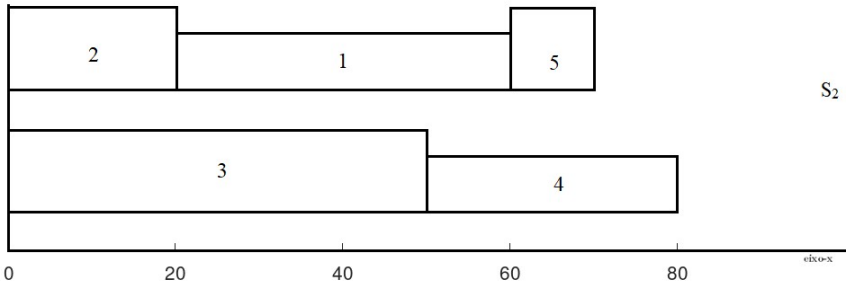
Pode-se observar que, diferentemente do DRLP, a solução do PROP deve respeitar a alocação prévia das facilidades de 1, ..., t (no exemplo, $t = 2$) em uma linha e o restante t+1, ..., n na segunda linha; as linhas devem iniciar em um ponto em comum à esquerda e sem espaços entre as facilidades.

Por último, tem-se os arranjos para o bCAP. Na Figura 13(a), é exibida a visualização do arranjo $S_1 = [L_1\, L_2]$, sendo $L_1 = [1\ 2]$ e $L_2 = [4\ 5\ 3]$ no extremo (f_1^{best}, f_2) com custos $f_1 = 450{,}0$ e $f_2 =$ 20% e na Figura 13(b) o arranjo $S_2 = [L_1\, L_2]$ com $L_1 = [3\ 4]$ e $L_2 = [2\ 1\ 5]$ no extremo (f_1, f_2^{best}) com custos $f_1 = 460{,}0$ e $f_2 = 6{,}66\%$.

Figura 13 – Visualização da solução para o bCAP



(a) Melhor custo de manuseio de materiais. $f_1 = 450{,}0$ e $f_2 = 20\%$



(b) Melhor comprimento do corredor. $f_1 = 460{,}0$ e $f_2 = 6{,}667\%$

Fonte: próprio autor.

Como o bCAP é um problema biobjetivo, não se pode dizer que a solução da Figura 13(a) é melhor ou pior que a Figura 13(b). Dependerá de como está sendo considerada a avaliação das soluções, por um tomador de decisão, para que possa ser feita a escolha da melhor solução.

No próximo capítulo são apresentadas as meta-heurísticas nas quais são baseadas as propostas de resolução dos problemas de layout SRFLP, DRLP, PROP e bCAP deste trabalho.

3 META-HEURÍSTICAS UTILIZADAS

Os problemas de otimização geralmente envolvem um grande número de variáveis e restrições. Nesses problemas o que se deseja encontrar é uma solução x^* tal que $f(x^*) \leq f(x)$, $\forall x \in X$, sendo X o espaço de soluções viáveis e f, uma função de avaliação da solução (BAZARAA; JARVIS; SHERALI, 2009). Alguns problemas podem ter características específicas, podendo restringir seu domínio ao conjunto dos números inteiros, denominados de problemas de programação inteira; ou mesmo aos conjuntos dos números inteiros e reais, chamados de problemas de programação inteira mista (BISSOLI, 2018).

Os problemas de layouts em linha são exemplos de problemas de otimização modelados como problemas de programação inteira e inteira mista de difícil resolução exata (AMARAL, 2006; TUBAILEH; SIAM, 2017), o que justifica as pesquisas com métodos por abordagens heurísticas ou meta-heurísticas. Neste capítulo são apresentadas as meta-heurísticas que norteiam as propostas de soluções para os problemas estudados.

3.1 GREEDY RANDOMIZED ADAPTIVE SEARCH PROCEDURE - GRASP

O GRASP, proposto por Feo e Resende (1995), é um método iterativo constituído de duas fases: a de construção e a de busca local. Na fase de construção, é gerada uma lista de candidatos (LC) ordenados de acordo com sua contribuição na função objetivo (ou por algum critério definido). Uma solução é construída elemento a elemento. Essa construção é aleatória, pois a partir de uma lista, denominada lista de candidatos restrita (LCR), formada pelos melhores elementos da LC, é sorteado o novo elemento a compor a solução. A heurística também é adaptativa, pois a cada elemento inserido na solução, as contribuições dos elementos restantes da LC são atualizadas para refletir as mudanças ocasionadas pela seleção do elemento na iteração anterior. Um parâmetro denominado $\alpha \in [0, 1]$ determina o tamanho da LCR, que, por sua vez, determina o quão guloso ou aleatório será o procedimento de construção.

Tendo em vista que a construção de uma solução apresenta uma característica de aleatoriedade, as soluções geradas na fase construtiva, provavelmente, não serão localmente ótimas. Daí a importância da segunda fase do GRASP, que busca melhorar a solução construída na fase anterior, trabalhando na sua vizinhança.

Devido à sua facilidade implícita, existem na literatura diversas aplicações práticas, o GRASP vem sendo aplicado em grande escala na pesquisa operacional e em problemas de otimização na indústria. Na literatura existem aplicações do GRASP em diversos problemas de otimização, como, por exemplo, rotulação cartográfica de pontos (CRAVO; RIBEIRO; LORENA, 2008) ou projeto de rede de transmissão (FARIA et al., 2005). Diversas outras aplicações do GRASP, incluindo problemas de mapeamento, lógica, particionamento, localização, teoria dos grafos, transporte, telecomunicações, design VLSI, podem ser encontrados em Festa e Resende (2009a, 2009b).

O pseudocódigo apresentado no Algoritmo 1 representa a meta-heurística GRASP para um problema de otimização com uma função objetivo de minimização.

No Algoritmo 1 têm-se como parâmetros o número máximo de iterações (*MaxIter*) e o valor de α que define quantos elementos irão compor a LCR.

Algoritmo 1 – Pseudocódigo da meta-heurística GRASP

```
entrada: MaxIter, α
saída: S*
 1: f* ← ∞, LC ← CriarListaCandidatos (),
 2: Para i ← 1 até MaxIter faça
 3:    S ← ContruçãoAleatoriaGulosa (LC, α)
 4:    S ← BuscalLocal (S)
 5:    Se f(S) < f* então
 6:       f* ← f(S)
 7:       S* ← S
 8:    Fim
 9: Fim
10: retorne S*
```

Na Linha 1, é inicializado o valor da melhor função objetivo e também é criada a LC. A cada iteração é construída uma solução gulosa e aleatória (Linha 3) e é aplicada a essa solução uma heurística de busca local (Linha 4). Se a solução melhorou (Linha 5), a melhor solução encontrada até então é atualizada (Linhas 6 e 7). Ao final do procedimento, a melhor solução armazenada em S^* é retornada (Linha 10).

Algumas variações do GRASP também foram propostas como um GRASP Reativo (PRAIS; RIBEIRO, 2000), que é um GRASP no qual não se usa um valor fixo para o parâmetro que define o tamanho da LCR durante a fase construtiva. O GRASP Reativo previamente se auto ajusta de acordo com a qualidade das soluções encontradas. Resende e Ribeiro (2005) apresentaram vários avanços e aplicações para o GRASP com *Path relinking*. Para uma revisão mais completa sobre o GRASP e suas aplicações, veja Festa e Resende (2009a, 2009b), Aiex, Binato e Resende (2003) e Resende e Ribeiro (2003).

3.2 PARTICLE SWARM OPTIMIZATION – PSO

O uso da inteligência de enxame é um ramo da inteligência artificial que estuda o comportamento de sistemas coletivos complexos constituídos de simples agentes que interagem de forma organizada formando pequenas sociedades (enxames). O comportamento agregado do enxame exibe características de inteligência (ALFEREZ, 2013).

A otimização por enxame de partículas (PSO, do inglês *Particle Swarm Optimization*) é uma técnica muito parecida com paradigmas populares de computação evolutiva como Algoritmos Genéticos (HOLLAND, 1975) e Evolução Diferencial (STORN; PRICE, 1997) e tem sido amplamente utilizada para o tratamento de funções mal estruturadas, contínuas, discretas, incluindo aplicações recentes em clusterização de dados (SENGUPTA; BASAK; PETER II, 2019, BONYADI; MICHALEWICZ, 2017). No funcionamento interno, o PSO faz uso de regras de transição probabilísticas para realizar buscas paralelas do hiperespaço de solução sem a adoção explícita de informação derivada do problema (SENGUPTA; BASAK; PETER II, 2019).

O PSO foi inspirado pelas leis do comportamento de bandos de pássaros, cardumes de peixes e interações humanas (KENNEDY; EBERHART, 1995). O modelo primitivo do PSO foi usado para estudar e simular graficamente a coreografia de pássaros. O algoritmo emprega uma população de pontos que se movem estocasticamente no espaço de busca. A melhor experiência ou posição alcançada por cada indivíduo é mantida, e depois comunicada à parte ou a toda a população. O esquema de comunicação é determinado por uma rede social fixa ou adaptativa, que desempenha um papel essencial nas propriedades de convergência do algoritmo.

A versão do PSO padrão usa uma população de partículas em que cada partícula representa uma solução candidata do problema tratado. O sistema é inicializado com uma população de soluções aleatórias e busca por soluções melhores, de acordo com alguma função de aptidão (*fitness*), gerando novas soluções pela atualização das posições das partículas através do espaço de busca multidimensional. As partículas acompanham suas próprias posições com melhor aptidão, bem como a melhor posição da sua vizinhança na população (KOTHARI et al., 2012; SENGUPTA; BASAK; PETERS II, 2019), ou seja, cada partícula guarda sua solução que resultou no melhor valor de aptidão e também conhece a solução que obteve o melhor valor de aptidão na sua vizinhança.

Formalmente, cada partícula *i* tem um vetor posição x_i e um vetor velocidade representado por um vetor v_i em um espaço multidimensional. Todas as partículas lembram de seu melhor vetor posição p_i e conhecem o melhor vetor posição global *g*. A cada iteração, o vetor v_i e o vetor x_i são atualizados de acordo com as Equações (3.1) e (3.2), respectivamente (ABRAHAM; GUO; LIU, 2006).

$$v_i = v_i + r_1 c_1 (p_i - x_i) + r_2 c_2 (g - x_i) \tag{3.1}$$

$$x_i = x_i + v_i \tag{3.2}$$

Onde r_1 e r_2 são números aleatórios, que são usados para manter a diversidade da população, sendo gerados por uma distribuição uniforme no intervalo [0, 1] para cada componente da partícula *i*. As constantes de aceleração c_1 e c_2 são constantes positivas chamadas de coeficientes da componente de auto reconhecimento e da componente social, respectivamente. Da Equação (3.1), a partícula decide para onde ir, considerando sua experiência p_i, que é a memória de sua melhor posição passada e a experiência da partícula mais bem sucedida do enxame *g*. Para guiar as partículas efetivamente no espaço de busca, a distância máxima de movimento durante uma iteração pode ser fixada entre as velocidades mínima e máxima [$-v_{min}$, v_{max}] (ABRAHAM; GUO; LIU, 2006).

Shi e Eberhart introduziram o fator de inércia que elimina a necessidade de fixação de velocidade, mas, ainda assim, restringe o comportamento divergente (SHI; EBERHART, 1998; EBERHART; SHI, 2001). O fator de inércia ω controla a atualização da partícula ponderando a contribuição da velocidade anterior, ou seja, basicamente controla quanto da velocidade anterior irá influenciar a nova velocidade. A equação da velocidade, então, foi alterada para:

$$v_i = \omega v_i + r_1 c_1 (p_i - x_i) + r_2 c_2 (g - x_i) \tag{3.3}$$

Estudos empíricos iniciais do PSO com a inércia têm mostrado que o valor de ω é muito importante para garantir um comportamento convergente (EBERHART; SHI, 2000; SHI; EBERHART, 1998). Valores de $\omega > 1$ geram um incremento da velocidade ao longo do tempo, o que leva a um comportamento divergente. Para valores $\omega < 0$, partículas desaceleram até que suas velocidades se tornem zero (dependendo dos valores de c_1 e c_2). Embora os valores de inércia estática tenham sido usados com sucesso, os valores de inércia adaptativa também mostraram levar a um comportamento convergente (SUGANTHAN, 1999; YOSHIDA et al., 1999).

O termo de inércia ajuda a garantir trajetórias convergentes, no entanto, para certos problemas, ainda é possível o controle dos limites dos valores da velocidade para impedir que as velocidades extrapolem. Esse problema pode ser resolvido pela seleção cuidadosa dos parâmetros ω, c_1 e c_2 (BERGH; ENGELBRECHT, 2006).

O pseudocódigo do Algoritmo 2 demonstra as etapas do algoritmo PSO padrão. Os critérios de parada geralmente usados são o número máximo de iterações (o processo de otimização termina após um número fixo de iterações); número de iterações sem melhora (o processo termina após algum número de iterações sem melhora na solução global); e erro mínimo da função objetivo (um erro entre o valor da função objetivo e o melhor valor de aptidão é menor que um limiar pré-fixado).

Algoritmo 2 – Pseudocódigo da meta-heurística PSO padrão

```
entrada: m, IterMax, ω, c1, c2
saída: g
 1: Para i ← 1 até m faça
 2:     Inicializa as posições x_i e a velocidade v_i da partícula i
 3:     p_i ← x_i
 4: Fim
 5: g ← argmin (fitness(p_i))
 6: Para t ← 1 até IterMax faça
 7:     Para i ← 1 até m faça
 8:         v_i ← ωv_i+r_1c_1(p_i-x_i)+r_2c_2(g-x_i)
 9:         x_i ← x_i+ v_i
10:         Se (fitness(x_i) < fitness(p_i) então
11:             p_i← x_i
12:             Se (fitness(x_i) < fitness(g) então
13:                 g ← x_i
14:             Fim
15:         Fim
16:     Fim
17: Fim
18: retorne g
```

No Algoritmo 2, o laço das Linhas 1-4 gera o enxame de partículas inicial, com a definição da posição x_i e do vetor velocidade v_i de cada partícula i (Linha 2) e, na Linha 3, a melhor posição p_i é inicializada com a posição atual x_i. Na Linha 5, a melhor posição dentre todas as partículas é armazenada em g. O processo iterativo do PSO inicia no laço das Linhas 6-17. O laço das Linhas 7-16 itera sobre todas partículas, atualizando as velocidades v_i (Linha 8) e a nova posição x_i (Linha 9) de acordo com as Equações (3.3) e (3.2), respectivamente. Na Linha 10, é verificado se a aptidão (*fitness*) da nova posição x_i é melhor que a aptidão da posição p_i. Se verdadeiro, a partícula p_i é atualizada (Linha 11) e ainda, na Linha 12, verifica-se a aptidão da nova posição x_i em relação à melhor solução (posição) atual g. Caso ocorra melhora, a melhor solução é atualizada na Linha 13. Ao final, a melhor solução g é retornada na Linha 18.

Estudos mais detalhados sobre o PSO e aplicações podem sem encontrados em Elbes et al. (2019), Sengupta, Basak e Peters II (2019), Bonyadi e Michalewicz (2017), Poli, Kennedy e Blackwell (2007), Abraham, Guo e Liu (2006), Bergh e Engelbrecht (2006), Eberhart e Shi (2001), Suganthan (1999) e Shi e Eberhart (1998).

3.3 ITERATED LOCAL SEARCH – ILS

O *Iterated Local Search* (ILS) é uma meta-heurística que aplica iterativamente, buscas locais e perturbações à solução. A ideia principal do ILS é de que a busca local leva o algoritmo a um ótimo local e, assim, é necessário um mecanismo para escapar desses ótimos locais. Então, o algoritmo tenta escapar, desconstruindo a solução corrente com uma perturbação que seja capaz de levar a outras soluções no espaço de busca, juntamente com o procedimento de aceitação, que introduz um mecanismo de intensificação ou diversificação, dependendo do critério adotado (LOURENÇO; MARTIN; STÜTZLE, 2003, 2010).

No ILS, existem quatro componentes bem definidos: a geração de solução inicial, a busca local, a perturbação, que guia a solução para uma nova solução no espaço de soluções, e um procedimento que define um critério de aceitação da nova solução (CHIARANDINI; STÜTZLE, 2002; NETO, 2016). O Algoritmo 3 ilustra o pseudocódigo com esses componentes.

Algoritmo 3 – Pseudocódigo ILS básico

saída: S^*
1: $k \leftarrow 0$
2: $S^k \leftarrow$ *GerarSoluçãoInicial* ()
3: $S^* \leftarrow$ *BuscaLocal* (S^k)
4: $A \leftarrow S^k$
5: **Enquanto** Condição de parada seja satisfeita **faça**
6: $k \leftarrow k + 1$
7: $A' \leftarrow$ *Perturbação* (A)
8: $S^k \leftarrow$ *BuscaLocal* (A')
9: $A \leftarrow$ *CritérioDeAceitação* (S^*, S^k)
10: **Fim**
11: **retorne** S^*

Na Linha 1 do Algoritmo 3, a variável k é inicializada e na Linha 2 uma solução inicial S^k é obtida utilizando alguma heurística ou simplesmente de maneira aleatória (STÜTZLE, 1998). Na Linha 3, a busca local é aplicada à solução S^k retornando uma solução localmente ótima S^*. A melhor solução corrente é armazenada na solução A (Linha 4). O processo iterativo do ILS

ocorre no laço das Linhas 5 a 10, até que um critério de parada seja satisfeito. Na Linha 6, o valor de k é incrementado. O procedimento de perturbação transforma uma solução corrente em outra solução inicial A' (Linha 7) e na Linha 8, é aplicada a busca local à solução A' que se move para uma nova solução localmente ótima S^k. O critério de aceitação, Linha 9, deve decidir qual das soluções ótimas locais previamente encontradas devem ser perturbadas na próxima iteração. O ILS mantém, durante todo o processo, a melhor solução encontrada em S^* e quando o algoritmo atinge uma nova solução localmente ótima, S^* pode ser atualizado. Por fim, o algoritmo retorna S^*.

A intensidade da perturbação da solução torna possível diversificar ou intensificar a busca em uma determinada região do espaço de soluções (STÜTZLE; RUIZ, 2018). De maneira geral, a intensidade da perturbação deve ser de tal forma que não permita que a busca local retorne facilmente à solução inicial (LOURENÇO; MARTIN; STÜTZLE, 2010). Se a perturbação for muito pequena, ela poderá ser desfeita rapidamente pela próxima busca local. Por outro lado, se a perturbação for muito forte, muitos componentes da solução são modificados e grande parte da qualidade e da estrutura da solução candidata atual pode ser perdida (LOURENÇO; MARTIN; STÜTZLE, 2010). Normalmente são aplicados movimentos de perturbação aleatórios, mas uma perturbação adaptativa, alterando a intensidade durante a busca, ou utilizando um histórico de buscas também pode ser empregada.

De acordo com Lourenço, Martin e Stützle (2003), a maioria dos trabalhos publicados sobre ILS não utiliza o histórico de busca tanto na perturbação, quanto na aceitação, operando de maneira dissociado das soluções encontradas anteriormente. Na literatura, o critério comumente utilizado é aceitar a melhor solução visitada, embora trabalhos recentes tenham mostrado de forma decisiva que a agregação de memória, para um histórico de busca, leva a um resultado melhor.

Se for considerado na perturbação e no critério de aceitação um histórico da busca, por exemplo, para ajustar a intensidade da perturbação ou para realizar a escolha no critério de aceitação (STÜTZLE; RUIZ, 2018), os procedimentos correspondentes seriam *Perturbação* (A, *histórico*) e *CritérioDeAceitação* (S^*, S^k, *histórico*), Algoritmo 3, Linhas 7 e 9, respectivamente. Mais detalhes sobre o ILS, incluindo especificidades sobre cada uma das quatro componentes, podem ser encontrados em Lourenço, Martin e Stützle (2010), Stützle, Ruiz (2018) e Neto (2016).

4 ALGORITMO GRASP PROPOSTO PARA O SRFLP

Esse capítulo apresenta o algoritmo GRASP-F proposto para a resolução do SRFLP. O algoritmo foi denominado GRASP-F e é uma extensão do algoritmo GRASP desenvolvido por Cravo e Amaral (2015). Em Cravo e Amaral (2015), ao realizar um movimento de vizinhança, o algoritmo realiza um cálculo completo da função objetivo. Guan e Lin (2016) e Kothari e Ghosh (2013b) apresentaram técnicas para acelerar a avaliação do ganho da função objetivo. Essas técnicas são usadas no algoritmo proposto. Adicionalmente, uma fase de diversificação foi incorporada, na tentativa de melhorar a exploração da vizinhança. A fase de diversificação adotada aqui é baseada em uma memória de frequência usada principalmente na busca tabu (GLOVER; LAGUNA, 1997): uma matriz F de tamanho $n \times n$ é mantida e atualizada, sendo que cada elemento da matriz F (i, p) contém a frequência com que a facilidade i ocupou a posição p nas soluções geradas até o momento. Dessa forma, atribuições mais raras podem ser identificadas e, portanto, favorecidas. Uma versão preliminar dessas ideias, que levaram ao algoritmo desenvolvido, está presente no trabalho de Cravo e Amaral (2017).

4.1 FASE CONSTRUTIVA

O procedimento disponível em Cravo e Amaral (2015) foi utilizado para a construção da solução na fase construtiva. Inicialmente a solução S está vazia, ou seja, ela tem n posições vazias. A lista de candidatos LC contém os possíveis pares de facilidades que podem ocupar a posição livre mais à esquerda p e a posição livre mais à direita q na solução parcial atual. Quando uma solução S está vazia, o custo w_{ij} do par de facilidades i e j, tal que i será colocada na posição livre $p = 1$ e j será colocada na posição livre $q = n$, é calculado pela Equação (4.1).

$$w_{ij}=c_{ij}\left(\frac{l_i+l_j}{2}+\sum_{\substack{k=1\\k\neq i,k\neq j}}^{n} l_k\right), \quad (1 \leq i < j \leq n) \tag{4.1}$$

Nota-se que por definição $L = \sum_{k=1}^{n} l_k$, e a Equação (4.1) pode então ser simplificada para $w_{ij} = c_{ij}\left(L-\frac{l_i+l_j}{2}\right)$, $(1 \leq i < j \leq n)$.

A lista de candidatos inicial contém todos os pares (i, j) tal que $1 \leq i < j \leq n$, ordenada pelos custos w_{ij}. Claramente, o tamanho da LC é $\frac{n(n-1)}{2}$. Os melhores elementos na LC definirão a LCR.

Então, um elemento é selecionado aleatoriamente da LCR e adicionado na solução vazia S nas posições p e q. Agora, como a solução não é mais vazia, e os custos w_{ij} dos pares candidatos são atualizados (recalculados), levando em consideração os elementos já pertencentes à solução parcial S, pela Equação (4.2).

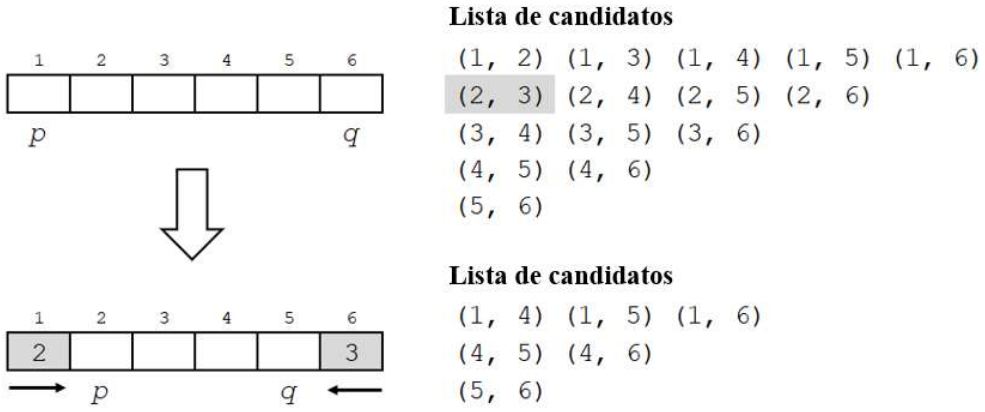
$$w_{ij}=c_{ij}\left(\frac{l_i+l_j}{2}+\sum_{\substack{k=1\\ k\neq i,k\neq j,k\notin S}}^{n} l_k\right)+\sum_{k\in(S'\cup S'')} c_{ki}d_{ki}+c_{kj}d_{kj}\,,\quad \forall i\notin S, j\notin S, i\neq j \tag{4.2}$$

Onde S' é o conjunto dos elementos à esquerda da posição livre mais à esquerda p na solução S; e S'' o conjunto dos elementos à direita da posição livre mais a direta q na solução S. A Equação (4.2) representa o custo w_{ij} em relação a um par candidato (i, j) (colocar a facilidade i na posição p e j na posição q) e implicitamente, considera-se o custo w_{ji} relativo a um par candidato (j, i) (colocar a facilidade i na posição q e j na posição p). No entanto, apenas o candidato que tem o custo menor, dos dois candidatos (i, j) e (j, i), é considerado na LC. A LC é reordenada pelos custos w_{ij}. A Figura 14 ilustra a aplicação das Equações (4.1) e (4.2) para um problema com 6 facilidades.

A fase construtiva (mostrada no Algoritmo 4) recebe como entrada: n (número de facilidades) e o valor de α ($0 \leq \alpha \leq 1$), que define o tamanho da LCR (LCR_{size}) pela Equação (4.3).

$$LCR_{size}=max\{1, \lfloor \alpha \times |\mathrm{LC}| \rfloor\} \tag{4.3}$$

Figura 14 – Construção de uma solução inicial



Fonte: próprio autor.

No exemplo da Figura 14, inicialmente a solução está vazia, sendo $p = 1$ e $q = n$, e a lista de candidatos contém $\binom{n}{2}$ pares de facilidades. O custo de cada par é calculado pela Equação (4.1).

Assumindo que o par (2, 3) foi selecionado para ser adicionado à solução, a facilidade 2 é colocada na posição *p* e a facilidade 3 é colocada na posição *q*, a lista de candidatos é atualizada, removendo os pares que contém *2* ou *3*, *p* é movido para a direita e *q* é movido para a esquerda. Para cada par (*i*, *j*) na lista de candidatos, o custo w_{ij} e w_{ji} são determinados pela Equação (4.2).

Algoritmo 4 – Pseudocódigo da Fase Construtiva proposta

```
entrada: n, α
saída: S
 1: Alocados ← [F, ..., F]
 2: S ← {}
 3: p ← 1, q ← n
 4: LC ← CriarListaCandidatos ()
 5: Enquanto LC ≠ {} faça
 6:    LCR_size ← max {1, ⌊α×|LC|⌋}
 7:    (i, j) ← SelecionarAleatoriamente (LC, LCR_size)
 8:    S[ p ] ← i
 9:    S[ q ] ← j
10:    p ← p+1
11:    q ← q–1
12:    Alocados[ i ] ← V
13:    Alocados[ j ] ← V
14:    RemoverConflitos (i, j, LC)
15:    AtualizarCustos (LC, S)
16:    OrdenarLista (LC)
17: Fim
18: Se mod (n, 2) ≠ 0 então
19:    S[ (n+1)/2 ] ← BuscarNãoAlocado (Alocados)
20: Fim
21: retorne S
```

Na Linha 1, o vetor *Alocados*, que registra se uma facilidade já foi alocada para a solução *S*, é inicializado colocando "*F*" (falso) para todas as facilidades; na Linha 2, a solução *S* é inicializada como vazia; e na Linha 3, os ponteiros *p* e *q* são inicializados de modo que *p* aponta para a esquerda e *q* para a posição mais à direita na solução *S*; na Linha 4, a lista de candidatos LC é criada por meio do procedimento *CriarListaCandidatos* e ordenada pelos custos dos candidatos calculados de acordo com a Equação (4.1); o laço definido nas Linhas 5-17 é executado até que não tenha mais candidato em LC; na Linha 6, o tamanho atual da LCR é calculado (Equação 4.3).

Na Linha 7, o par candidato a ser adicionado à solução é selecionado aleatoriamente de LCR e nas Linhas 8 à 9 o par de facilidades selecionado é adicionado em *S* nas posições livres apontadas por *p* e *q*. Nas Linhas 10 à 11, o ponteiro *p* aponta para a próxima posição livre à direita, enquanto o ponteiro *q* aponta para a próxima posição livre à esquerda. Nas Linhas 12 à 13, as facilidades *i* e *j*, que acabaram de ser adicionadas à solução, são marcadas como alocadas, e na Linha 14, o procedimento *RemoverConflitos* remove quaisquer candidatos (pares)

contendo *i* ou *j* da LC. Na Linha 15, os custos dos candidatos restantes em LC são atualizados (conforme a Equação 4.2) por meio do procedimento *AtualizarCustos*. Na Linha 16, a LC é reordenada, de forma não decrescente, de acordo com os custos pelo procedimento *OrdenarLista*. A Linha 18 verifica se a instância tem uma quantidade ímpar de facilidades, e, se for verdadeiro, a solução deve ser ajustada da seguinte forma: busca-se a facilidade ainda não alocada, utilizando o procedimento *BuscarNãoAlocado,* e coloca-a na posição central da solução *S*. A propriedade adaptativa do procedimento reside no fato de que a cada par de facilidade adicionado na solução *S*, os pares contendo alguma facilidade alocada são removidos do LC e os custos dos pares restantes em LC são atualizados. Os custos são recalculados levando em consideração a solução parcial *S* em construção. No fim do procedimento construtivo, uma solução viável completa *S* é retornada (Linha 21). A solução retornada *S* provavelmente não será otimizada localmente. A fim de melhorar a solução obtida nessa fase, uma busca local será aplicada.

4.1.1 Fases Construtivas Alternativas

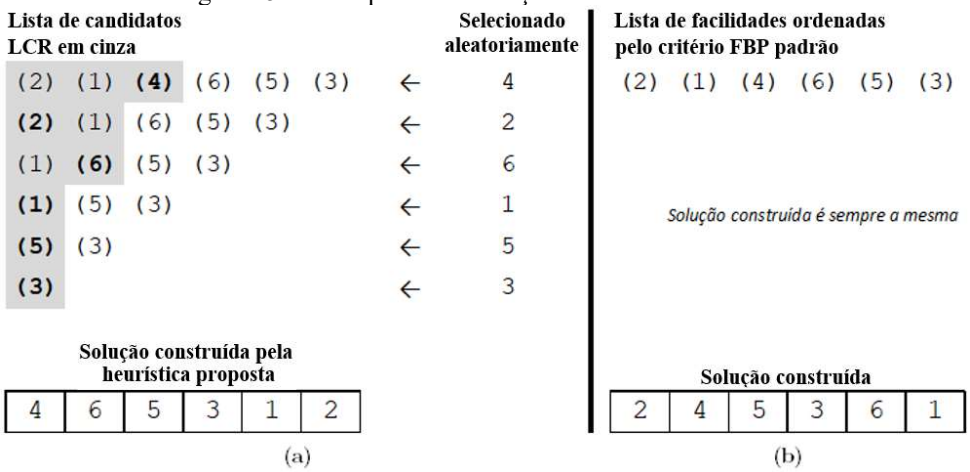
Datta, Amaral e Figueira (2011) apresentam quatro técnicas para a construção de soluções iniciais viáveis para o SRFLP. As técnicas são denominadas *worst pair together* (WPT), *flow-based permutation* (FBP), *length-based permutation* (LBP) e *random initialization* (RND). A WPT e a FBP usam informações de fluxo entre facilidades; LBP usa os comprimentos das facilidades, enquanto RND é uma inicialização puramente aleatória. A fase construtiva proposta neste trabalho, dada pelas Equações (4.1) e (4.2), assemelha-se à técnica WPT, entretanto, apresenta uma maior complexidade no cálculo dos pesos w_{ij} para os pares de facilidades (i, j) que compõem a LC, pois incorpora uma característica de adaptabilidade necessária na meta-heurística GRASP, que não está presente nas inicializações de Datta, Amaral e Figueira (2011).

Para comparar as várias maneiras de construir uma solução inicial viável para o SRFLP, cinco versões do GRASP-F foram criadas, quatro versões com fases construtivas baseadas nas técnicas WPT, LBP, FBP, RND, e uma versão utilizando as Equações (4.1) e (4.2). Lembrando que o GRASP tem uma característica importante, a LCR, cujo tamanho (LCR_{size}) regula a proporção entre o quão guloso e aleatório será o processo de construção. No entanto, as técnicas de construção apresentadas em Datta, Amaral e Figueira (2011) são puramente gulosas (WPT, LBP e FBP) ou puramente aleatória (RND). Uma técnica puramente gulosa construirá a mesma

solução sempre que for utilizada. Portanto, as técnicas WPT, LBP e FBP foram modificadas de modo a permitir o comportamento da LCR.

Considere, por exemplo, a construção FBP, que inicialmente cria uma lista na qual as facilidades são classificadas em ordem não decrescente de seus valores de $q_i = \sum_{j=1,\ j \neq i}^{n} c_{ij} l_j$, $i \in \{1, ..., n\}$, e, em seguida, remove cada facilidade da lista em ordem e a coloca nas extremidades da solução de forma alternada. A adaptação proposta, ao invés de extrair da lista essa facilidade com o menor q_i, será extraída uma facilidade escolhida aleatoriamente entre as LCR_{size} primeiras facilidades com menor q_i. Assim, soluções diferentes podem ser construídas cada vez que o procedimento construtivo é chamado. Para a técnica RND não foi definida uma LCR, porque na técnica não existe uma definição de lista de facilidades candidatas. A Figura 15 ilustra a diferença entre o processo FBP original e a alteração proposta.

Figura 15 – Exemplo de construção usando a técnica FBP



Fonte: próprio autor.

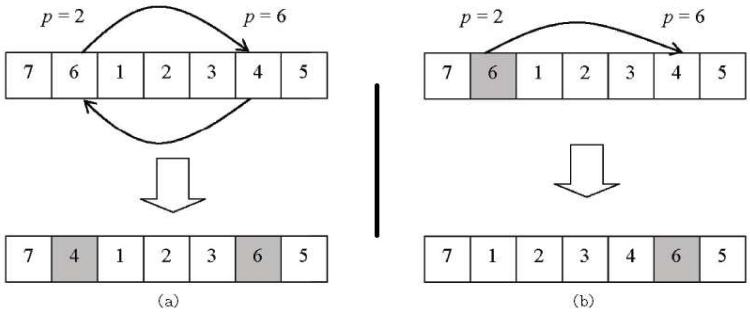
O exemplo da Figura 15(a) considera uma LCR de tamanho definido com α=0,5 na Equação (4.3) para a FBP e a Figura 15(b) mostra um exemplo de construção da solução usando a FBP original.

4.2 FASE DE BUSCA LOCAL

Na presente implementação, na segunda fase do GRASP não é aplicada uma, mas cinco estratégias de busca local, baseadas no movimento de troca 2-opt e de inserção. Os movimentos

de vizinhança 2-opt e inserção são amplamente usados na literatura para o SRFLP (KOTHARI; GHOSH, 2013b; OZCELIK, 2012; PALUBECKIS, 2015). A Figura 16 ilustra os movimentos de vizinhança.

Figura 16 – Movimentos de vizinhança: (a) 2-opt e (b) inserção



Fonte: próprio autor.

No contexto dos problemas de layout e considerado nesse trabalho, o movimento de troca 2-opt, consiste na troca de posição de duas facilidades dispostas no arranjo, o que difere da troca *k*-opt definido no clássico problema do caixeiro viajante (HELSGAUN, 2000). A Figura 16(a) mostra um exemplo de movimento de troca 2-opt com a troca do par de facilidades das posições 2 e 6. Já no exemplo do movimento inserção, Figura 16(b), a facilidade da posição 2 é inserida na posição 6. Com isso, os elementos intermediários são deslocados para liberar a posição 6 para inserção da facilidade da posição 2.

As cinco rotinas de busca local implementadas aqui não são padrão, como em uma rotina que gera uma sequência de movimentos (p, q) e encerra. Assim, o número de iterações realizadas pela rotina é exatamente o tamanho da sequência (p, q) gerada. Um vizinho da solução atual é obtido por um movimento (p, q) que troca as facilidades em posições p e q (movimento 2-opt) ou que insere a facilidade da posição p na posição q (movimento inserção). A rotina avalia uma vizinha S^1 da solução atual S. Se S^1 não melhorar em S, a rotina irá examinar o vizinho S^2, que é obtido aplicando o próximo movimento (p, q) em S, e assim por diante. Quando um vizinho S^k que melhora S é encontrado, este substitui a solução atual S (ou seja, $S \leftarrow S^k$) para que o próximo passo (p, q) seja aplicado a S^k. O Algoritmo 5 fornece um exemplo, em que a sequência de movimentos (p, q) é definida por $p = 1, ..., n$; $q = 1, ..., n$; $q \neq p$ e o movimentos 2-opt são utilizados.

Algoritmo 5 – Exemplo de Rotina de Busca Local para o GRASP-F

entrada: S_0
saída: S
1: $S \leftarrow S_0$
2: $P \leftarrow \{1, 2, 3, ..., n\}$
3: $Q \leftarrow \{1, 2, 3, ..., n\}$
4: **Para cada** $p \in P$ **faça**
5: **Para cada** $q \in Q$ **faça**
6: **Se** $p \neq q$ **então**
7: $S' \leftarrow ObterVizinhoMovimento\_2\text{-}Opt\ (S, p, q)$
8: **Se** $f(S') < f(S)$ **então**
9: $S \leftarrow S'$
10: **Fim**
11: **Fim**
12: **Fim**
13: **Fim**
14: **retorne** S

Três rotinas de busca local baseadas no 2-opt são utilizadas e denominadas:

- 2_*opt_todos_os_pares_k*: seleciona aleatoriamente uma posição k e gera movimentos 2-opt (p, q) na ordem $p = k, ..., n, 1, ..., k\text{-}1$; $q = 1, ..., n$; $q \neq p$. O número de iterações é $n \times (n\text{-}1)$;
- 2_*opt_todos_os_pares*: gera sequencialmente movimentos 2-opt (p, q) com $p = 1, ..., n$; $q = 1, ..., n$; $q \neq p$. Diferentemente da rotina 2_*opt_todos_os_pares_k*, a busca nessa rotina sempre inicia pelo mesmo par ($p=1$, $q=2$). O número de iterações é $n \times (n\text{-}1)$; e
- 2_*opt_trocas_adjacentes*: sequencialmente, gera movimentos restritos a pares de posições consecutivas, ou seja, (p, q) com $p = 1, ..., n\text{-}1$ e $q = p+1$ e então $p = n, ..., 2$ e $q = p\text{-}1$. O número de iterações é $2 \times (n\text{-}1)$.

Para os cálculos dos ganhos na função objetivo, dos movimentos 2-opt e trocas adjacentes, utilizou-se a técnica apresentada em Kothari e Ghosh (2013b). Os autores apresentaram uma expressão para o cálculo do ganho de um movimento (p, q) com complexidade computacional $O(n^2)$ e para o cálculo dos movimentos (p, $q+1$) em $O(n)$, pois sua técnica leva em consideração parte do cálculo já realizado para o movimento (p, q).

Para o movimento de inserção são definidas duas rotinas de busca local e denominadas:

- *inserção_todos_os_pares_k*: seleciona aleatoriamente uma posição k e sequencialmente gera movimentos (p, q) tentando inserir a facilidade da posição p ($p = k, ..., n, 1, ..., k\text{-}1$) na posição $q = 1, ..., n$; com $q \neq p$. O número de iterações é $n \times (n\text{-}1)$;

- *inserção_todos_os_pares*: gera sequencialmente movimentos (p, q) para todas as posições $p = 1, ..., n$ da solução, na tentativa de inserir a facilidade da posição de p nas posições $q = 1, ..., n$ com $q \neq p$. O número de iterações é $n \times (n\text{-}1)$.

Para calcular o ganho da função objetivo nos movimentos de inserção, foi utilizada a técnica desenvolvida em Guan e Lin (2016). A técnica apresenta complexidade computacional $O(n^2)$, no entanto, neste trabalho, é proposta uma melhoria, baseada nas ideias apresentadas em Kothari e Ghosh (2013b), para a avaliação do ganho na função objetivo no movimento (p, q+1), reduzindo a complexidade nesses casos, para $O(n)$. Deste modo, a técnica proposta para a inserção, aproveita o cálculo do ganho na função objetivo já realizado para o par (p, q) definida por Guan e Lin (2016) para calcular o ganho do movimento de inserção para o par (p, q+1), quando não ocorre melhora no par anterior (p, q).

Dado o processo de geração dos pares (p, q) das rotinas que utilizam os movimentos de inserção descrito anteriormente, define-se $\Delta_{pq+1_{(p<q)}}$ como o ganho no valor da função objetivo ao avaliar o movimento (p, q+1) quando $p < q$ e $\Delta_{pq+1_{(p>q)}}$ como o ganho no valor da função objetivo ao avaliar o movimento (p, q+1) quando $p > q$, tendo as expressões mostradas nas Equações (4.4) e (4.5), respectivamente.

Nas Equações (4.4) e (4.5), $S1$ contém as facilidades das posições $i < p$; $S2$ contém as facilidades das posições i tal que $p < i \leq q$; $S3$ é o conjunto das facilidades das posições $i > q$; c é o custo fluxo de materiais dos pares de facilidades (por exemplo c_{ip}, custo do fluxo de materiais entre o par de facilidades das posições i e p); l é o comprimento de uma facilidade (por exemplo, l_p, comprimento da facilidade da posição p); e d é a distância entre duas facilidades no arranjo (por exemplo, d_{pi}, distância entre as facilidades das posições p e i).

$$\begin{aligned}\Delta_{pq+1_{(p<q)}} = & \left[\sum_{i\in S1} c_{ip}D_2 - \sum_{i\in S1}\sum_{j\in S2} c_{ij}l_p - \sum_{i\in S3} c_{pi}D_2 \right. \quad + \\ & \left. \sum_{i\in S2} c_{pi}(D_2 + l_p - 2d_{pi}) + \sum_{i\in S2}\sum_{j\in S3} c_{ij}l_p \right] \quad + \\ & \left[\sum_{i\in S1} (c_{ip}l_{q+1} - c_{iq+1}l_p) + \sum_{i\in S2} (c_{pi}l_{q+1} - c_{iq+1}l_p) + \right. \\ & \left. \sum_{i\in S3} (c_{q+1i}l_p - c_{pi}l_{q+1}) + c_{pq+1}(2(D_2 + l_{q+1}) + l_p - 2d_{pq+1}) \right]\end{aligned} \tag{4.4}$$

Na Equação (4.4), a primeira parte da equação, contida nos primeiros colchetes, é o cálculo do ganho na função objetivo realizada para o par anterior (p, q) com $p < q$ definida em Guan e Lin

(2016) e a segunda parte, contida no segundo colchetes, é o cálculo adicional para a avaliação do ganho da função objetivo para o próximo par (p, q+1) com $p < q$. Assim, pode-se aproveitar o cálculo do ganho do par anterior para o cálculo do próximo par, sendo necessário apenas avaliar a segunda parte da expressão, que pode ser realizado com um esforço computacional de $O(n)$.

$$\begin{aligned}\Delta_{pq+1_{(p>q)}} = & \left[\sum_{i\in S1} c_{ip}D_2 - \sum_{i\in S1}\sum_{j\in S2} c_{ij}l_p - \sum_{i\in S3} c_{pi}D_2 + \right. \\ & \left. \sum_{i\in S2} c_{pi}(D_2 + l_p - 2d_{pi}) + \sum_{i\in S2}\sum_{j\in S3} c_{ij}l_p \right] + \\ & \left[\sum_{i\in S1} (c_{iq}l_p - c_{ip}l_q) + \sum_{i\in S2} (c_{iq}l_p - c_{pi}l_q) + \right. \\ & \left. \sum_{i\in S3} (c_{qi}l_p - c_{pi}l_q) - c_{pq}(2(D_2 - l_q) + l_p - 2d_{pq+1}) \right]\end{aligned} \tag{4.5}$$

Novamente, na Equação (4.5), a expressão contida no primeiro par de colchetes é o cálculo do ganho na função objetivo realizada para o par anterior (p, q) com $p < q$ definida em Guan e Lin (2016), já calculada para o par anterior (p, q) sendo necessário apenas o cálculo da segunda parte da expressão, segundo par de colchetes, que pode ser realizado com um esforço computacional de $O(n)$.

Os detalhamentos dos passos realizados para se determinar as melhorias nos cálculos dos ganhos dos movimentos de inserção, definidos pelas Equações (4.4) e (4.5) são apresentados no Apêndice A.

O Algoritmo 6 mostra o pseudocódigo para o procedimento de busca local, utilizando as cinco rotinas propostas, onde são chamadas as cinco rotinas definidas. No algoritmo, o procedimento de busca local aplica de forma sistemática as cinco estruturas de vizinhança à solução atual S. A solução S é inicializada na Linha 1 e assim, no laço das Linhas 2-12, o algoritmo busca encontrar a solução S_1, que, provavelmente, é uma solução melhor em relação a S (Linha 4). Em seguida, o algoritmo tenta encontrar S_2, uma solução possivelmente melhor que a S_1 (Linha 5), e assim por diante. O laço das Linhas 2-12 é repetido até que nenhuma melhora na solução atual S seja obtida. As cinco rotinas são aplicadas na seguinte ordem: 2*_opt_todos_os_pares_k* (Linha 4), *inserção_todos_os_pares_k* (Linha 5), 2*_opt_trocas_adjacentes* (Linha 6), 2*_opt_todos_os_pares* (Linha 7) e *inserção_todos_os_pares* (Linha 8). No final do laço das Linhas 2-12 a solução S é retornada (Linha 13).

Algoritmo 6 – Procedimento de Busca Local – GRASP-F/SRFLP

entrada: S_0
saída: S
1: $S \leftarrow S_0$
2: **Faça**
3: $melhorou \leftarrow 0$
4: $S_1 \leftarrow 2\_opt\_todos\_os\_pares\_k\ (S)$
5: $S_2 \leftarrow inserção\_todos\_os\_pares\_k\ (S_1)$
6: $S_3 \leftarrow 2\_opt\_trocas\_adjacentes\ (S_2)$
7: $S_4 \leftarrow 2\_opt\_todos\_os\_pares\ (S_3)$
8: $S_5 \leftarrow inserção\_todos\_os\_pares\ (S_4)$
9: **Se** $f(S_5) < f(S)$ **então**
10: $S \leftarrow S_5$, $melhorou \leftarrow 1$
11: **Fim**
12: **Enquanto** $melhorou = 1$
13: **retorne** S

4.3 FASE DE DIVERSIFICAÇÃO

Na tentativa de melhorar a exploração da vizinhança das soluções geradas pelo GRASP, ao algoritmo proposto foi adicionada uma fase de diversificação. Como dito antes, a fase de diversificação adotada aqui é baseada em uma memória de frequência usada principalmente na busca tabu.

Uma versão preliminar dessas ideias, que levaram ao algoritmo desenvolvido, aparece no trabalho de Cravo e Amaral (2017).

4.3.1 Matriz de Frequência

A matriz de frequência F contém $n \times n$ elementos inteiros, onde n é o número de facilidades. Cada linha da matriz F representa uma facilidade e cada coluna da matriz representa uma posição na solução. O valor do elemento (i, p) da matriz armazena o número de vezes que a facilidade i foi alocada na posição p nas soluções anteriores. Assim, a matriz de frequência representa uma memória de longo prazo que servirá para realizar a diversificação. Dessa forma, alocações mais raras podem ser identificadas e, portanto, favorecidas.

A matriz de frequência F é atualizada com base na solução S fornecida pela fase de busca local, em cada iteração do algoritmo GRASP-F.

4.3.2 Diversificação

A diversificação é feita tomando facilidades e alterando suas posições na solução corrente. Primeiramente, um conjunto K é construído contendo K_{size} facilidades escolhidas aleatoriamente entre as n facilidades. Para cada facilidade $i \in K$, tem-se na linha i da matriz F, a frequência com que a facilidade i ocupou cada posição $p = 1, ..., n$ no passado. Em seguida, uma lista contendo as posições classificadas por frequência, em ordem não decrescente, é formada. Assim, as posições mais raras da facilidade i terão prioridade e um parâmetro LCR_{diver} é usado para limitar o tamanho dessa lista de posições de destino. A ideia é similar à LCR na fase de construção do GRASP. A nova posição para cada facilidade $i \in K$ é selecionada aleatoriamente dentre as primeiras LCR_{diver} posições candidatas, e então, a facilidade i é inserida na nova posição selecionada.

4.4 O ALGORITMO GRASP-F PROPOSTO

Com as definições da fase de construção da solução inicial, das estruturas de vizinhanças, da fase de busca local e da fase de diversificação, nesta seção é apresentada a proposta de implementação da meta-heurística GRASP para o SRFLP.

O algoritmo proposto, diferentemente do GRASP-PR de Rubio-Sánchez et al. (2016), mantém uma única solução corrente durante o processo iterativo e intensifica a exploração na vizinhança da solução, utilizando uma busca local composta por cinco rotinas com movimentos de troca 2-opt e inserção, as quais utilizam técnicas para acelerar o cálculo do ganho na função objetivo. Além disso, foi incorporada uma fase de diversificação utilizando uma matriz de frequência que a cada iteração do GRASP alterna entre a construção de uma solução inicial e a realização da diversificação na solução corrente. O Algoritmo 7 apresenta o pseudocódigo para o GRASP-F. O procedimento recebe como entrada os seguintes parâmetros:

- n: tamanho do problema;
- α: um valor no intervalo [0, 1] que define LCR_{size}, ou seja, quantos dos melhores elementos da LC serão parte da LCR na fase de construção do GRASP pela Equação (4.3);

- *MaxIterWI*: o valor que define o número de iterações decorridas sem melhoria na melhor solução encontrada S^*. Assim, o valor de *MaxIterWI* define quando ocorre a construção de uma nova solução;
- β: um valor no intervalo [0, 1] que define K_{size}, pela fórmula: $K_{size} = \lfloor \beta \times n \rfloor$. Em outras palavras, quantas facilidades serão realocadas na etapa de diversificação, e
- γ: também um valor no intervalo [0, 1], o qual determina LCR_{diver} (o tamanho da lista de posições destino no procedimento de diversificação), calculado como: $LCR_{diver} = \lfloor \gamma \times n \rfloor$.

Algoritmo 7 – Algoritmo GRASP-F proposto

entrada: *n, α, MaxIterWI, β, γ*
saída: S^*
1: $f^* \leftarrow \infty$, $K_{size} \leftarrow \lfloor \beta \times n \rfloor$, $LCR_{diver} \leftarrow \lfloor \gamma \times n \rfloor$
2: *Iter* ← 0
3: *UltimaIterSemMelhora* ← 0
4: $S \leftarrow$ *Procedimento_Construção* (*n, α*)
5: **Enquanto** (*Critério de parada não satisfeito*) **faca**
6: **Se** *Iter – UltimaIterSemMelhora* > *MaxIterWI* **então**
7: $S \leftarrow$ *Procedimento_Construção* (*n, α*)
8: *UltimaIterSemMelhora* ← *Iter*
9: **Senão Se** *Iter* > 0 **então**
10: *Diversificação* (*F*, *S*, K_{size}, LCR_{diver})
11: **Fim**
12: $S \leftarrow$ *Busca_Local* (*S*)
13: *Atualizar_Matriz_Frequencia* (*F*, *S*)
14: **Se** $f(S) < f^*$ **então**
15: $f^* \leftarrow f(S)$
16: $S^* \leftarrow S$
17: *UltimaIterSemMelhora* ← *Iter*
18: **Fim**
19: *Iter* ← *Iter* + 1
20: **Fim**
21: **retorne** S^*

As Linhas de 1 a 4 inicializam as variáveis locais, incluindo a primeira solução inicial (Linha 4). O laço das Linhas 5-20 é repetido até que o critério de parada seja satisfeito. Na Linha 6, o algoritmo verifica se o número de iterações sem melhora na melhor solução foi alcançado (*MaxIterWI*). Se o número máximo de iterações foi alcançado, o algoritmo constrói uma nova solução *S* (Linha 7) e *UltimaIterSemMelhora* é atualizado para o valor da iteração corrente *Iter* (Linha 8). Caso contrário, para *Iter* > 0, o procedimento de diversificação é aplicado à solução corrente *S* (Linha 10). Na Linha 12, o procedimento de busca local é aplicado à solução *S* (Seção 4.2). Na Linha 13, a matriz de frequência *F* é atualizada usando a solução corrente *S*. A matriz de frequência é usada na fase de diversificação do algoritmo (Linha 10). Na Linha 14, é verificado se a nova solução é melhor do que a melhor solução encontrada até então. Se

verdadeiro, a melhor solução encontrada S^* e a variável *UltimaIterSemMelhora* são atualizadas. Portanto, sempre que o número de iterações sem melhoria de S^* é atingido, o algoritmo alterna para a construção de uma nova solução inicial S (Linha 7) e, em seguida, aplica o procedimento de busca local para a solução S (Linha 12). Note que quando há uma melhoria em S^* (Linha 14), a variável *UltimaIterSemMelhora* é atualizada (Linha 17) e, em seguida, o algoritmo aplica o procedimento de diversificação na solução S (Linha 10), em vez de gerar uma nova solução pela fase de construção.

O GRASP-F tem como critério de parada um tempo máximo de execução. Assim, implicitamente, esse parâmetro também está sendo recebido pelo procedimento definido no Algoritmo 7.

4.5 EXPERIMENTOS COMPUTACIONAIS PARA O GRASP-F

Nos experimentos, foi utilizado um computador desktop com um processador Intel i7 - 7700k 4,2 GHz com 16GB de memória. A calibração dos parâmetros dos algoritmos GRASP-F foi realizada por meio de um algoritmo PSO. O GRASP-F e o PSO foram implementados em linguagem de programação C e compilados no compilador GCC 5.1.0 x64 com as *flags* de otimização -O3 e -march. Nos experimentos para o SRFLP, foram utilizadas as 93 instâncias da literatura listadas a seguir:

- 3 instâncias de Amaral e Letchford (2013), cada uma com tamanho $n = 110$;
- 20 instâncias de Rubio-Sánchez et al. (2016), denominadas *Anjos-large*, com cinco instâncias para cada $n \in \{200, 300, 400, 500\}$;
- 20 instâncias de Rubio-Sánchez et al. (2016), denominadas *Sko-large*, com cinco instâncias para cada $n \in \{200, 300, 400, 500\}$;
- 20 instâncias de Palubeckis (2015), com uma instância para cada $n \in \{110, 120, ..., 290, 300\}$;
- 30 instâncias Palubeckis (2017), com uma instância para cada $n \in \{310, 320, ..., 490, 500\} \cup \{550, 600, ..., 950, 1000\}$.

4.5.1 Calibração dos Parâmetros do GRASP-F

Essa subseção apresenta a calibração dos parâmetros do GRASP-F e das versões do GRASP-F com as fases construtivas alternativas. Os resultados obtidos na calibração dão subsídios às escolhas tanto da fase de construção, quanto dos parâmetros que são utilizados na avaliação de desempenho da meta-heurística GRASP-F proposta. A calibração utiliza a meta-heurística PSO, como pode ser visto no Apêndice B.

O GRASP-F tem um importante parâmetro denominado α, que determinada o tamanho da lista de candidatos restrita (LCR_{size}). Um outro parâmetro é *MaxIterWI*, que determina o número de iterações que ocorrem sem melhoria na melhor solução encontrada que, quando excedido, leva o algoritmo à fase de construção. Outros parâmetros são necessários para a fase de diversificação: β, que determina o número K_{size} de facilidades que serão realocadas na fase de diversificação, e γ, que determina o tamanho LCR_{diver} da lista de posições de destino para uma facilidade que será realocada para uma nova posição na fase de diversificação. Assim, para definir os intervalos dos possíveis valores dos parâmetros para a calibração, são definidos os componentes do vetor posição x_i como sendo os parâmetros do GRASP-F e listados a seguir: x_{i1}: o valor de α, $0{,}05 \leq x_{i1} \leq 0{,}9$; x_{i2}: o valor de β, $0{,}05 \leq x_{i2} \leq 0{,}9$; x_{i3}: o valor de *MaxIterWI*, $5 \leq x_{i3} \leq 90$; e x_{i4}: o valor de γ, $0{,}05 \leq x_{i4} \leq 0{,}9$.

O PSO de calibração foi executado para um subconjunto contendo 4 das 20 instâncias de Palubeckis (2015). As instâncias usadas foram n = {150, 200, 250, 300} e os parâmetros usados no PSO foram: $m = 10$ (dez partículas) e $IterMax = 36$ (total de iterações do PSO). Para cada partícula criada pelo PSO, os tempos máximos de execução do GRASP-F foram definidos em 30 segundos, mas algumas pequenas variações podem ocorrer dependendo do tempo gasto pelo GRASP-F em uma iteração. Assim, para cada instância, o PSO executou um tempo total de 3 horas. Os parâmetros obtidos pelo PSO para as versões do GRASP-F com fases construtivas diferentes são mostrados na Tabela 1.

Na Tabela 1, a primeira coluna mostra a denominação, dada a combinação do GRASP-F com uma dada fase construtiva. A segunda coluna mostra o nome da instância utilizada, seguido do nome dado ao conjunto de parâmetros em cada linha e os valores encontrados pelo PSO para cada versão do GRASP-F.

Os conjuntos de parâmetros encontrados pelo PSO são diferentes para cada instância. Assim, para determinar o conjunto de parâmetros que fornece o melhor desempenho para as várias

versões do GRASP-F, cada versão do algoritmo foi executada com cada conjunto de parâmetros, sendo que foi utilizado o mesmo subconjunto de instâncias e tempo de 100s para cada execução. O parâmetro α da versão do GRASP-F com a fase construtiva RND foi removido da tabela, pois não é utilizado por essa versão do GRASP-F. Os resultados encontrados para cada conjunto de parâmetros são mostrados na Tabela 2.

A Tabela 2 é estruturada da seguinte forma: a primeira coluna mostra as denominações dadas às versões do GRASP-F com as fases construtivas, seguida dos nomes dos conjuntos de parâmetros, na terceira coluna, *Média* (v^*), tem-se a média de v^* (obtidos em Palubeckis, 2015) para o subconjunto de instâncias usado na calibração. A coluna *Média* mostra a média das soluções encontradas para a versão do GRASP-F para o subconjunto de instâncias e, por fim, os valores da diferença entre as médias, calculada como *Dif = Média – Média* (v^*). A qualidade do conjunto de parâmetros é medida pelo valor da coluna *Dif*. Quanto menor o valor, melhor o parâmetro.

Tabela 1 – Calibração das versões do GRASP-F/SRFLP com o PSO

Algoritmo	Instância	Parâmetro	α	β	γ	*MaxIterWI*
GRASP-F WPT	p150	P1	0,0500	0,8182	0,0500	44
	p200	P2	0,0500	0,0500	0,0500	5
	p250	P3	0,9000	0,3925	0,9000	90
	p300	P4	0,9000	0,4050	0,9000	90
GRASP-F FBP	p150	P1	0,1297	0,6537	0,9000	61
	p200	P2	0,1578	0,6683	0,7403	79
	p250	P3	0,0500	0,1287	0,0500	5
	p300	P4	0,2686	0,3880	0,9000	80
GRASP-F LBP	p150	P1	0,3854	0,0821	0,0500	5
	p200	P2	0,0500	0,0500	0,0500	5
	p250	P3	0,1884	0,0738	0,4540	70
	p300	P4	0,0500	0,0500	0,0500	14
GRASP-F RND	p150	P1	-	0,0898	0,4267	7
	p200	P2	-	0,1918	0,0500	5
	p250	P3	-	0,0500	0,5207	60
	p300	P4	-	0,5562	0,5939	5
GRASP-F Proposto	p150	P1	0,5545	0,2623	0,2865	6
	p200	P2	0,0500	0,0500	0,0500	5
	p250	P3	0,9000	0,1773	0,9000	51
	p300	P4	0,3515	0,0690	0,2729	53

Tabela 2 – Desempenhos das versões do GRASP-F/SRFLP na calibração

Algoritmo	Parâmetro	*Média*(*v**)	*Média*	*Dif*
GRASP-F WPT	P1	47142766,9	47161361,4	18594,5
	P2	47142766,9	47145035,4	2268,5
	P3	47142766,9	47144779,1	**2012,3**
	P4	47142766,9	47145552,4	2785,5
GRASP-F FBP	P1	47142766,9	47144259,9	**1493,0**
	P2	47142766,9	47148473,1	5706,3
	P3	47142766,9	47148206,6	5439,8
	P4	47142766,9	47156951,4	14184,5
GRASP-F LBP	P1	47142766,9	47149803,4	7036,5
	P2	47142766,9	47148578,4	5811,5
	P3	47142766,9	47144220,9	1454,0
	P4	47142766,9	47143260,2	**493,3**
GRASP-F RND	P1	47142766,9	47145897,6	3130,8
	P2	47142766,9	47145786,4	**3019,5**
	P3	47142766,9	47165623,1	22856,3
	P4	47142766,9	47151844,9	9078,0
GRASP-F Proposto	P1	47142766,9	47147227,4	4460,5
	P2	47142766,9	47145524,6	2757,8
	P3	47142766,9	47144369,9	**1603,0**
	P4	47142766,9	47161005,1	18238,3

Como pode ser visto na Tabela 2, o melhor conjunto de parâmetros para a versão do GRASP-F com a fase de construção do WPT é o P3 com α=0,9000, β=0,3925, γ=0,9000 e *MaxIterWI* = 90; para a versão com a fase construtiva FBP, o melhor conjunto de parâmetros é o P1 com α=0,1297, β=0,6537, γ=0,9000 e *MaxIterWI* = 61; para a versão GRASP-F com a fase de construção LBP, o melhor conjunto é P4 com α=0,0500, β=0,0500, γ=0,0500 e *MaxIterWI* = 14. Para GRASP-F + RND, o melhor conjunto de parâmetros é P2 com β=0,1918, γ=0,0500 e *MaxIterWI* = 5; e para a versão GRASP-F proposto, o melhor conjunto de parâmetros é P3 com α=0,9000, β=0,1773, γ=0,9000 e *MaxIterWI* = 51.

Na próxima subseção, são apresentados os resultados computacionais para cada versão do algoritmo GRASP-F, usando as melhores configurações de parâmetros encontrados.

4.5.2 Comparação do GRASP-F com diferentes Fases Construtivas

As diversas versões do GRASP-F foram comparadas usando os melhores conjuntos de parâmetros definidos na subseção anterior. Para determinar a melhor versão do GRASP-F, os algoritmos foram executados dez vezes para cada instância, com o tempo de 600s por execução. A Tabela 3 apresenta os resultados encontrados com as diferentes fases de construção.

A Tabela 3 está estruturada da seguinte forma: a primeira coluna mostra a versão do algoritmo, seguido pelo nome das instâncias, e a coluna *v** mostra a melhor solução conhecida na literatura (PALUBECKIS, 2015). Na quarta coluna aparecem as melhores soluções encontradas pelo GRASP-F, na quinta coluna, têm-se os valores médios das soluções em dez execuções dos algoritmos, e por último, o desvio percentual da média, calculado como $Desv = \frac{(Média\text{-}v^*)}{v^*} \times 100$. A última linha apresenta a média geral sobre todas as colunas.

Na Tabela 3, a versão do GRASP-F proposto obteve o melhor desempenho, com a melhor média geral do valor de *Desv* entre as versões testadas (0,000235). Assim, a fase de construção adotada é a proposta nesse trabalho, ou seja, a que utiliza para cálculo dos pares de facilidades as Equações (4.1) e (4.2) e os melhores parâmetros desta versão, cujo conjunto foi denominado P3 com $\alpha = 0{,}9000$, $\beta = 0{,}1773$, $\gamma = 0{,}9000$ e $MaxIterWI = 51$. Os experimentos computacionais para o GRASP-F são apresentados nas subseções seguintes.

Tabela 3 – Resultados do GRASP-F/SRFLP para cada fase construtiva

Algoritmo	Instância	v*	*Melhor*	*Média*	*Desv*
GRASP-F WPT	p150	10624389,5	10624389,5	10624389,5	0
	p200	27482649,5	27482649,5	27482649,5	0
	p250	54526293,5	54526293,5	54526380,9	0,000160
	p300	95937735,0	95937769,0	95938642,3	0,000946
	Média	47142766,9	47142775,4	47143015,6	**0,000277**
GRASP-F FBP	p150	10624389,5	10624389,5	10624389,5	0
	p200	27482649,5	27482649,5	27482650,0	0,000002
	p250	54526293,5	54526293,5	54528072,7	0,003263
	p300	95937735,0	95937757,0	95939379,9	0,001715
	Média	47142766,9	47142772,4	47143623,0	**0,001245**
GRASP-F LBP	p150	10624389,5	10624389,5	10624389,5	0
	p200	27482649,5	27482649,5	27482649,5	0
	p250	54526293,5	54526293,5	54526585,7	0,000536
	p300	95937735,0	95937830,0	95938716,7	0,001023
	Média	47142766,9	47142790,6	47143085,4	**0,000390**
GRASP-F RND	p150	10624389,5	10624389,5	10624389,5	0
	p200	27482649,5	27482649,5	27482654,0	0,000016
	p250	54526293,5	54526303,5	54528303,0	0,003685
	p300	95937735,0	95938557,0	95941770,7	0,004207
	Média	47142766,9	47142974,9	47144279,3	**0,001977**
GRASP-F Proposto	p150	10624389,5	10624389,5	10624389,5	0
	p200	27482649,5	27482649,5	27482649,5	0
	p250	54526293,5	54526293,5	54526575,0	0,000516
	p300	95937735,0	95937745,0	95938142,6	0,000425
	Média	47142766,9	47142769,4	47142939,2	**0,000235**

4.5.3 Critério de avaliação do desempenho

O desempenho do GRASP-F é avaliado pela comparação das soluções encontradas com as melhores soluções conhecidas para diversas instâncias de problemas de grande porte disponíveis na literatura.

Para cada instância, o GRASP-F foi executado 10 (dez) vezes. Cada rodada leva um tempo pequeno e limitado, definidos conforme os tempos utilizados em Palubeckis (2017). Os tempos foram fixados como segue:

- Para as instâncias de Amaral e Letchford (2013), o tempo limite foi de 100s;
- Para as instâncias de Palubeckis (2015), o tempo de execução foi de 600s;
- Para as instâncias de Palubeckis (2017), os tempos limites foram: 1200s para $n \in \{310, ...400\}$, 1800s para $n \in \{410, ..., 500\}$ e 3600s para $n \geq 550$; e
- Para as instâncias *Sko-large* e *Anjos-large* de Rubio-Sánchez et al. (2016), os tempos limites foram: 600s para $n \in \{200, 300\}$, 1200s para $n = 400$ e 1800s para $n = 500$.

Nos resultados apresentados na próxima subseção, dada a melhor solução encontrada após as 10 execuções (*Melhor*) e a média dos valores das 10 soluções (*Média*), são calculadas as diferenças entre *Melhor* e a melhor solução da literatura v^*, $Dif_{melhor} = Melhor - v^*$, e a diferença entre *Média* e v^*, $Dif_{média} = Média - v^*$. Além do desvio-padrão, denotado por *DP*.

É interessante observar se o algoritmo converge de forma consistente para uma solução de alta qualidade em cada uma das 10 execuções. Assim, duas medidas são calculadas:

- A notação *Desv*, mostra o desvio percentual da média em relação ao valor da melhor solução conhecida (v^*), dado por $Desv = 100 \times \frac{Média - v^*}{v^*}$; e
- *DPr*, que apresenta o desvio-padrão relativo calculado como $DPr = 100 \times \frac{DP}{Média}$.

4.5.4 Experimentos com instâncias de Amaral e Letchford (2013) com $n = 100$

A Tabela 4 apresenta os resultados dos experimentos com o GRASP-F para as instâncias de Amaral e Letchford (2013), as quais tem tamanho $n = 100$ e são comparados com os melhores valores conhecidos disponíveis em Ozcelik (2012) obtidos pelo algoritmo HGA (*Hybrid Genetic Algorithm*). A Tabela 4 está estruturada da seguinte forma: a primeira coluna mostra

os nomes das instâncias; na segunda coluna têm-se os tamanhos das instâncias; na terceira coluna são mostrados os valores das melhores soluções conhecidas na literatura; a quarta coluna mostra os valores das melhores soluções encontradas pelo GRASP-F; na quinta coluna são apresentados os valores das médias das soluções; e, por último, os desvios-padrão. A última linha de cada tabela mostra as médias dos valores, calculados sobre as instâncias que aparecem na tabela e os valores das melhores soluções são destacados em negrito.

Tabela 4 – Resultados do GRASP-F/SRFLP instâncias de Amaral e Letchford (2013)

Instância	n	$v^{*(a)}$	GRASP-F		
			Dif_{melhor}	$Dif_{média}$	DP
SRFLP-110-1	110	**144296664,5**	0	0	0
SRFLP-110-2	110	**86050037,0**	0	0	0
SRFLP-110-3	110	**2234743,5**	0	0	0
Média		77527148,3	0	0	0

[(a)] O melhor valor conhecido é dado por Ozcelik (2012).
Os melhores valores são destacados em negrito.

A Tabela 5 apresenta as medidas de desempenho do algoritmo. A primeira coluna da tabela mostra os nomes das instâncias, seguido dos desvios percentuais do valor da solução média em relação à melhor solução conhecida e o desvio-padrão relativo ao valor da solução média. Na última coluna, $T_{médio}$, tem-se o tempo médio de execução (em segundos) nas 10 execuções do algoritmo e, na última linha, as médias de todas as colunas da tabela.

Tabela 5 – Desempenho do GRASP-F/SRFLP instâncias de Amaral e Letchford (2013)

Instância	GRASP-F		
	$Desv$	DPr	$T_{médio}$
SRFLP-110-1	0	0	100
SRFLP-110-2	0	0	100
SRFLP-110-3	0	0	100
Média	0	0	100

A Tabela 5 mostra que, para cada instância, as soluções médias são iguais às melhores soluções conhecidas, ou seja, os valores conhecidos foram alcançados em todas as execuções. Para esse conjunto de instâncias, os tempos médios foram iguais aos tempos limites definidos na Subseção 4.5.3. Pode-se notar que o algoritmo alcançou os melhores valores conhecidos na literatura para todas as instâncias em um tempo limitado em 100s.

4.5.5 Experimentos com instâncias *Anjos-large* com $200 \leq n \leq 500$

As melhores soluções conhecidas para as vinte instâncias *Anjos-large* ($200 \leq n \leq 500$) foram obtidas por Rubio-Sánchez et al. (2016). Os autores rodaram três algoritmos denominados GENALGO_R, SS-2P_R e GRASP-PR. Cada algoritmo foi executado por uma hora para cada instância. O algoritmo GRASP-PR proposto por Rubio-Sánchez et al. (2016) encontrou os melhores valores para todas as instâncias, exceto para a instância Anjos_200_02, a qual o melhor valor conhecido foi encontrado pelo algoritmo GENALGO_R.

A Tabela 6 mostra os resultados do GRASP-F para as instâncias denominadas *Anjos-large* (RUBIO-SÁNCHEZ et al., 2016) e está estruturada como a Tabela 4. Os resultados mostram que quando consideradas as instâncias *Anjos-large*, o GRASP-F melhorou os resultados conhecidos para quatorze das vinte instâncias (em destaque na tabela). Dentre as quatorze instâncias que tiveram as soluções melhoradas, uma instância tem tamanho $n = 200$, três têm tamanhos $n = 300$, cinco têm tamanhos $n = 400$, e outras cinco têm tamanhos $n = 500$. Ou seja, as melhorias foram obtidas para as maiores instâncias nesse conjunto. Para as três instâncias Anjos_200_01, Anjos_200_03 e Anjos_300_04, o GRASP-F alcançou os melhores valores conhecidos. Para as três instâncias restantes, o GRASP-F encontrou soluções muito próximas às melhores.

Tabela 6 – Resultados do GRASP-F/SRFLP instâncias *Anjos-large*

Instância	n	$v^{*(a)}$	GRASP-F		
			Dif_{melhor}	$Dif_{média}$	DP
Anjos_200_01	200	**305461818,0**	**0**	0	0,0
Anjos_200_02	200	178807197,5	**-369,0**	-265,3	165,1
Anjos_200_03	200	**61891275,0**	**0**	97,7	80,0
Anjos_200_04	200	**127735691,0**	10,0	399,3	217,0
Anjos_200_05	200	**89057121,5**	9,0	106,8	78,6
Anjos_300_01	300	1549663657,0	**-240385,0**	-104694,0	82741,5
Anjos_300_02	300	955572066,5	**-33764,0**	-32600,1	2517,3
Anjos_300_03	300	**308257630,5**	15,0	659,4	291,8
Anjos_300_04	300	**602873168,5**	**0**	3052,2	5028,8
Anjos_300_05	300	466160264,0	**-8969,0**	-372,7	14585,2
Anjos_400_01	400	5000752142,5	**-935340,5**	-796813,5	316544,1
Anjos_400_02	400	2910276759,0	**-11263,0**	160851,0	543398,2
Anjos_400_03	400	921216455,0	**-355146,0**	-269331,9	201835,1
Anjos_400_04	400	1806061379,0	**-422430,0**	-421812,0	422,3
Anjos_400_05	400	1402779472,5	**-944157,5**	-887219,5	77798,7
Anjos_500_01	500	12300409839,0	**-9713589,0**	-5838543,0	3059297,0
Anjos_500_02	500	7493120635,5	**-1420084,5**	-1080470,5	563560,7
Anjos_500_03	500	2479333773,0	**-985617,0**	-515811,0	316731,9
Anjos_500_04	500	4285937468,5	**-4830208,5**	-4809557,5	52659,7
Anjos_500_05	500	3678038066,5	**-2668837,5**	-1403371,5	972374,1
Média		2346170294,0	-1128506,3	-799784,8	310516,4

[(a)] O melhor valor conhecido foi obtido pelo GRASP-PR (RUBIO-SÁNCHEZ et al., 2016), exceto Anjos_200_02, obtido pelo GENALGO_R (RUBIO-SÁNCHEZ et al., 2016).
Os melhores valores são destacados em negrito.

A Tabela 7 mostra as medidas de desempenho do GRASP-F para as instâncias *Anjos-large* e está estruturada como a Tabela 5.

Tabela 7 – Desempenho do GRASP-F/SRFLP instâncias *Anjos-large*

Instância	GRASP-F		
	Desv	*DPr*	$T_{médio}$
Anjos_200_01	0,0000	0,0000	600,1
Anjos_200_02	-0,0001	0,0001	600,1
Anjos_200_03	0,0002	0,0001	600,1
Anjos_200_04	0,0003	0,0002	600,1
Anjos_200_05	0,0001	0,0001	600,1
Anjos_300_01	-0,0068	0,0053	600,6
Anjos_300_02	-0,0034	0,0003	600,5
Anjos_300_03	0,0002	0,0001	600,6
Anjos_300_04	0,0005	0,0008	600,4
Anjos_300_05	-0,0001	0,0031	600,7
Anjos_400_01	-0,0159	0,0063	1201,3
Anjos_400_02	0,0055	0,0187	1201,3
Anjos_400_03	-0,0292	0,0219	1201,4
Anjos_400_04	-0,0234	0,0000	1201,3
Anjos_400_05	-0,0633	0,0055	1201,0
Anjos_500_01	-0,0475	0,0249	1802,4
Anjos_500_02	-0,0144	0,0075	1802,5
Anjos_500_03	-0,0208	0,0128	1802,4
Anjos_500_04	-0,1123	0,0012	1802,7
Anjos_500_05	-0,0382	0,0264	1802,7
Média	-0,0184	0,0068	1051,1

Como pode ser visto, o GRASP-F apresenta uma convergência consistente para boas soluções nesse conjunto de instâncias, com desvios percentuais muito pequenos em relação às melhores soluções conhecidas e desvios-padrão relativos também muito baixos. Para esse conjunto de instâncias, os tempos médios, $T_{médio}$, foram muito próximos aos tempos limites definidos na Subseção 4.5.3, tendo pequenas variações dependendo do custo computacional de cada iteração do algoritmo.

4.5.6 Experimentos com instâncias *Sko-large* com $200 \leq n \leq 500$

Os resultados do GRASP-F para as instâncias *Sko-large* ($200 \leq n \leq 500$) são mostrados na Tabela 8. Os melhores valores conhecidos para essas instâncias foram obtidos pelo algoritmo GRASP-PR de Rubio-Sánchez et al. (2016), com um tempo limite de uma hora para cada instância. A tabela está estruturada conforme Tabela 4.

Tabela 8 – Resultados do GRASP-F/SRFLP instâncias *Sko-large*

Instância	*n*	$v^{*(a)}$	GRASP-F		
			Dif_{melhor}	$Dif_{média}$	*DP*
Sko_200_01	200	3231379,0	**-335,0**	308,0	425,4
Sko_200_02	200	**7758927,0**	7,0	23,5	11,9
Sko_200_03	200	**12739043,0**	**0**	6,2	6,7
Sko_200_04	200	**20260531,0**	**0**	9,6	24,9
Sko_200_05	200	**26871976,5**	14,0	26,7	14,2
Sko_300_01	300	**11251960,0**	48,0	751,8	511,4
Sko_300_02	300	28993831,0	**-1977,0**	-286,9	1627,0
Sko_300_03	300	48800510,5	**-9485,0**	-3807,3	8006,2
Sko_300_04	300	71194203,5	**-8944,0**	-7770,7	1040,1
Sko_300_05	300	86792128,5	**-2585,0**	547,9	5439,5
Sko_400_01	400	26718485,0	**-9486,0**	-2119,9	7503,6
Sko_400_02	400	67996107,0	**-45434,0**	-35826,6	9840,7
Sko_400_03	400	115942726,0	**-60792,0**	-46022,8	12338,4
Sko_400_04	400	163099026,0	**-166006,0**	-117014,9	50983,2
Sko_400_05	400	227778641,5	**-145623,0**	-126718,0	40726,1
Sko_500_01	500	52951975,0	**-78584,0**	-63732,7	17681,2
Sko_500_02	500	127428477,0	**-64203,0**	-41589,5	30742,8
Sko_500_03	500	231041993,5	**-122535,0**	-78868,8	51658,2
Sko_500_04	500	340390652,0	**-115670,0**	-42430,6	66653,2
Sko_500_05	500	445738609,0	**-39132,0**	-23325,2	42601,8
Média		105849059,1	-43536,1	-29392,0	17391,8

(a) O melhor valor conhecido foi obtido pelo GRASP-PR (RUBIO-SÁNCHEZ et al., 2016). Os melhores valores são destacados em negrito.

Como pode ser visto na Tabela 8, o GRASP-F encontrou novas soluções, melhores que os resultados disponíveis até então, para quinze das vinte instâncias. As instâncias que tiveram seus melhores valores conhecidos melhorados são as de maior tamanho nesse conjunto (uma instância com $n = 200$, quatro instâncias com $n = 300$, cinco instâncias com $n = 400$ e cinco com $n = 500$). Para as duas instâncias, Sko_200_03 e Sko_200_04, o GRASP-F alcançou os valores das melhores soluções conhecidas em pelo menos uma execução. Para as três instâncias restantes, o GRASP-F encontrou soluções ligeiramente piores.

A Tabela 9 mostra as medidas de desempenho do GRSAP-F para esse conjunto de instâncias e está estruturada conforme a Tabela 5.

Os valores dos desvios percentuais das soluções médias obtidas são muito próximos dos melhores valores conhecidos e com desvios-padrão relativos muito pequenos (Tabela 9). Assim, o GRASP-F converge consistentemente para soluções muito boas. Também para esse conjunto de instâncias, os tempos médios, $T_{médio}$, foram muito próximos aos tempos limites definidos na Subseção 4.5.3.

Tabela 9 – Desempenho do GRASP-F/SRFLP instâncias *Sko-large*

Instância	GRASP-F		
	Desv	*DPr*	$T_{médio}$
Sko_200_01	0,0095	0,0132	600,1
Sko_200_02	0,0003	0,0002	600,1
Sko_200_03	0,0000	0,0001	600,1
Sko_200_04	0,0000	0,0001	600,1
Sko_200_05	0,0001	0,0001	600,1
Sko_300_01	0,0067	0,0045	600,7
Sko_300_02	-0,0010	0,0056	600,5
Sko_300_03	-0,0078	0,0164	600,3
Sko_300_04	-0,0109	0,0015	600,4
Sko_300_05	0,0006	0,0063	600,5
Sko_400_01	-0,0079	0,0281	1202,5
Sko_400_02	-0,0527	0,0145	1201,1
Sko_400_03	-0,0397	0,0106	1201,3
Sko_400_04	-0,0718	0,0313	1201,4
Sko_400_05	-0,0557	0,0179	1201,1
Sko_500_01	-0,1205	0,0334	1803,5
Sko_500_02	-0,0326	0,0241	1802,5
Sko_500_03	-0,0341	0,0224	1802,9
Sko_500_04	-0,0125	0,0196	1801,5
Sko_500_05	-0,0052	0,0096	1802,9
Média	-0,0218	0,0130	1051,2

4.5.7 Experimentos com instâncias com $110 \leq n \leq 300$ de Palubeckis (2015)

A Tabela 10 mostra os resultados que o GRASP-F obteve para o conjunto de instâncias com $110 \leq n \leq 300$ de Palubeckis (2015). Os melhores resultados, v^*, conhecidos para essas instâncias foram obtidos por Palubeckis (2015) utilizando o algoritmo denominado VNS-LS3. O algoritmo VNS-LS3 foi executado por Palubeckis (2015) um longo tempo, limitado para cada instância em 5 horas.

Pode-se notar, na Tabela 10, que o GRASP-F alcançou os melhores valores conhecidos para dezesseis das vinte instâncias, sendo dez instâncias com $110 \leq n \leq 200$ e uma instância para cada n em {210, 230, 240, 250, 260, 270}. Em particular, para onze das treze instâncias com n em {110, ..., 230}, o algoritmo alcançou o melhor resultado conhecido em todas as 10 execuções. Para quatro instâncias, as soluções encontradas são ligeiramente piores.

As medidas de desempenho do GRASP-F para as instâncias Palubeckis (2015) são mostradas na Tabela 11, que está estruturada como a Tabela 5.

Nota-se que o algoritmo mantém uma convergência consistente para boas soluções nesse conjunto de instâncias. Os desvios percentuais das médias dos valores são muito pequenos. Por exemplo, o desvio máximo observado é de 0,0030% (para a instância p290), o qual é muito pequeno, assim como os desvios-padrão relativos. Para as instâncias de Palubeckis (2015), os

tempos médios, $T_{médio}$, foram muito próximos aos tempos limites definidos na Subseção 4.5.3, as pequenas variações são explicadas pela variação do tempo em cada iteração do algoritmo.

Tabela 10 – Resultados do GRASP-F/SRFLP instâncias de Palubeckis (2015)

Instância	n	$v^{*(a)}$	GRASP-F		
			Dif_{melhor}	$Dif_{média}$	DP
p110	110	**4435868,0**	**0**	0	0
p120	120	**6282721,0**	**0**	0	0
p130	130	**7880929,5**	**0**	0	0
p140	140	**9257162,0**	**0**	0	0
p150	150	**10624389,5**	**0**	0	0
p160	160	**14873277,0**	**0**	0	0
p170	170	**16630187,0**	**0**	0	0
p180	180	**18746031,5**	**0**	0,2	0
p190	190	**24453272,0**	**0**	0	0
p200	200	**27482649,5**	**0**	0	0
p210	210	**29512000,5**	**0**	0	0
p220	220	**37351850,5**	7,0	87,5	69,9
p230	230	**46744886,0**	**0**	0	0
p240	240	**46717781,0**	**0**	16,3	49,5
p250	250	**54526293,5**	**0**	281,5	825,3
p260	260	**63300360,5**	**0**	269,4	491,3
p270	270	**68960438,5**	**0**	3,7	6,6
p280	280	**73845821,0**	6,0	1558,4	4884,2
p290	290	**86255267,5**	79,0	2585,6	5604,1
p300	300	**95937735,0**	10,0	407,6	474,3
Média		37190946,1	5,1	260,5	620,3

[a] O melhor valor conhecido foi obtido pelo algoritmo VNS-LS3 com tempo limite de 5 horas para cada instância, como descrito por Palubeckis (2015).
Os melhores valores são destacados em negrito.

Tabela 11 – Desempenho do GRASP-F/SRFLP instâncias Palubeckis (2015)

Instância	GRASP-F		
	$Desv$	DPr	$T_{médio}$
p110	0,00000	0,00000	600,0
p120	0,00000	0,00000	600,0
p130	0,00000	0,00000	600,0
p140	0,00000	0,00000	600,0
p150	0,00000	0,00000	600,0
p160	0,00000	0,00000	600,0
p170	0,00000	0,00000	600,1
p180	0,00000	0,00000	600,1
p190	0,00000	0,00000	600,1
p200	0,00000	0,00000	600,1
p210	0,00000	0,00000	600,2
p220	0,00020	0,00020	600,1
p230	0,00000	0,00000	600,2
p240	0,00000	0,00010	600,2
p250	0,00050	0,00150	600,2
p260	0,00040	0,00080	600,3
p270	0,00000	0,00000	600,2
p280	0,00210	0,00660	600,4
p290	0,00300	0,00650	600,5
p300	0,00040	0,00050	600,4
Média	0,00033	0,00081	600,2

4.5.8 Experimentos com instâncias com $310 \leq n \leq 1000$ de Palubeckis (2017)

A Tabela 12 mostra a comparação do GRASP-F com os algoritmos VNS-LS3, MSA_0 e MSA_0.5 (PALUBECKIS, 2017) com o conjunto de instâncias Palubeckis (2017) com $310 \leq n \leq 1000$ facilidades. Os melhores valores conhecidos (v^*) para essas instâncias foram obtidas por Palubeckis (2017) ao executar, por um longo período de tempo, o algoritmo MSA_0 (o tempo limite para uma execução foi de 10 horas para $n \leq 500$ e 20 horas para $n > 500$). A primeira coluna da tabela mostra os nomes das instâncias, a segunda coluna apresenta os tamanhos das instâncias, na coluna v^* têm-se os melhores valores conhecidos; e, para cada algoritmo, a melhor solução é mostrada na coluna Dif_{Melhor} como a diferença entre as melhores soluções encontradas e os valores de v^*, a coluna $Dif_{Média}$ sendo a diferença entre os valores médios das soluções e os valores das melhores soluções conhecidas e a última coluna mostra os valores dos desvios-padrão. Os tempos limites adotados nos testes realizados para esse conjunto de instâncias são os mesmos adotados por Palubeckis (2017) na execução dos algoritmos MSA_0, MSA_0.5 e VNS-LS3.

Na Tabela 12, ao observar os valores das soluções encontradas, pode-se observar que o algoritmo proposto não produz resultados significativamente ruins. Embora os dois melhores algoritmos para essas instâncias sejam as variantes MSA, o GRASP-F produziu soluções com valores muito próximos dos melhores valores conhecidos. Além disso, o GRASP-F tem um desempenho melhor que o VNS-LS3, que é o pior dos quatro algoritmos para essas instâncias, tanto ao considerar os valores médios quanto as medidas dos desempenhos apresentados na

Tabela 13.

A

Tabela 13 apresenta as medidas de desempenho para a avaliação do GRASP-F em comparação com os algoritmos MSA_0, MSA_0.5 e VNS-LS3. A primeira coluna da tabela mostra os nomes das instâncias e para cada algoritmo tem-se os desvios percentuais da média e os desvios-padrão relativos. Para o GRASP-F, na última coluna, é mostrado o valor médio do tempo de execução para cada instância. Para esse conjunto com as maiores instâncias, novamente os tempos médios, $T_{médio}$, foram muito próximos aos tempos limites definidos na Subseção 4.5.3.

Tabela 12 – Resultados do GRASP-F/SRFLP instâncias Palubeckis (2017)

Instância	*n*	$v^{*(a)}$	MSA_0		MSA_0.5		VNS-LS3		GRASP-F	
			Dif_{Melhor} ($Dif_{Média}$)	*DP*	Dif_{Melhor} ($Dif_{Média}$)	*DP*	Dif_{Melhor} ($Dif_{Média}$)	*DP*	Dif_{Melhor} ($Dif_{Média}$)	*DP*
p310	310	**105754955,0**	**0**(41,1)	100,9	**0**(68,2)	136,4	**0**(13353,4)	17669,7	38(265,7)	154,1
p320	320	**119522881,5**	**0**(104,3)	197,1	**0**(185,5)	242,4	**0**(13811,6)	13338,1	82(6690,4)	6253,4
p330	330	**124891823,5**	**0**(0)	0	**0**(0)	0	**0**(13784,4)	28689,2	**0**(93,1)	55,7
p340	340	**129796777,5**	**0**(8,2)	16,2	**0**(18,8)	46,8	**0**(18129,7)	12708,7	123(1053,2)	1398,5
p350	350	**149594388,0**	**0**(1,6)	2,9	**0**(0,8)	2,4	**0**(850,6)	1217,9	**0**(102,6)	211,4
p360	360	**141122187,0**	**0**(15,2)	16,4	**0**(945,0)	1747,4	7552(18499,0)	10736,4	39(5100,1)	2645,9
p370	370	**174663159,5**	**0**(0)	0	**0**(0)	0	**0**(27322,7)	18539,3	823(3309,5)	1127,2
p380	380	**189452403,5**	**0**(6106,4)	7581,8	5(10075,2)	6727,4	**0**(55065,2)	40208,0	**0**(16870,3)	11163,9
p390	390	**208688557,0**	**0**(1,8)	2,7	**0**(1,6)	2,5	**0**(25119,8)	58074,3	538(1080,4)	751,6
p400	400	**213768810,5**	**0**(137,8)	405,4	**0**(372,6)	574,0	**0**(86822,1)	70000,1	33(13703,1)	42742,9
p410	410	**243494269,0**	**0**(130,7)	250,0	**0**(315,3)	302,2	771(139188,6)	58223,4	59(648,3)	513,3
p420	420	**270756527,5**	**0**(12,2)	9,7	**0**(8)	11,0	**0**(61250,8)	67133,8	**0**(2676,4)	8442,1
p430	430	**286334521,5**	50(10169,3)	6545,3	5426(11591,7)	5945,7	**0**(51128,8)	40579,3	1370(16985,3)	21256,5
p440	440	**301067264,5**	**0**(11,8)	17,8	**0**(22,2)	23,5	**0**(51841,0)	69322,9	390(6918,4)	5664,0
p450	450	**324488485,0**	**0**(2505,0)	6301,6	**0**(678,6)	1837,9	**0**(58852,1)	43175,5	38(16571,7)	12051,6
p460	460	**314884659,0**	**0**(23,5)	31,0	**0**(1506,2)	2287,2	19046(121696,6)	81326,7	795(19113,9)	10072,2
p470	470	**379529990,0**	43(5275,6)	3748,4	**0**(4997,4)	3641,0	4571(83188,3)	72517,9	4892(26679,9)	12178,9
p480	480	**366821075,0**	5(33,8)	26,0	**0**(453,6)	887,2	64279(128487,8)	54662,9	138(4996,6)	4685,3
p490	490	**413901954,5**	24(120,5)	108,6	**0**(218,7)	360,9	18216(178850,7)	139882,6	18993(26961,0)	15932,2
p500	500	**465570835,5**	12(7897,4)	9294,1	**0**(12228,2)	8777,8	27797(171775,2)	71813,7	7464(90740,4)	74119,0
p550	550	**587090450,5**	**0**(495,4)	1371,7	12(1784,9)	3760,3	**0**(156073,4)	109625,5	240(58122,0)	44196,6
p600	600	**801567664,5**	**27**(15066,6)	9027,7	5696(21370,7)	9781,3	64091(205450,0)	128615,1	2429(83102,9)	50244,9
p650	650	**927512834,0**	52(16701,7)	23277,0	**0**(9438,4)	14634,2	111697(507097,6)	238047,4	6194(73398,6)	77300,3
p700	700	**1158462340,0**	**110**(31815,3)	44511,8	123(58104,8)	60841,9	220828(701776)	253314,3	146(284907,0)	110033,5
p750	750	**1438408861,0**	**84**(8082,4)	7095,2	124(19131,5)	12840,1	717965(984477,5)	212670,7	21854,5(201204,5)	159346,4
p800	800	**1861593391,0**	**198**(55572,0)	27313,4	6433(74295,5)	36803,1	610509(1208417,9)	471036,3	129230(350929,0)	146034,3
p850	850	**2126675923,0**	**0**(80540,5)	92376,0	18(125397,4)	126021,2	884039(1832777,7)	596218,2	204860(523746,0)	394424,7
p900	900	**2600124305,0**	**1993**(86203,3)	51133,4	214(110839,4)	62645,7	1221813(2025800,6)	570647,6	296255(467825,0)	171719,1
p950	950	**2993153593,0**	**0**(42993,6)	78621,7	18932(192666,8)	119746,0	1640934(2979178,8)	782731,3	260439,5(571265,5)	391499,6
p1000	1000	**3426647267,0**	**0**(120817,5)	197536,2	110592(282866,5)	173755,0	3524866(4200597,8)	546123,7	163102(1288480,0)	448047,1
Média		761511405,1	86,6(16362,8)	18897,3	4919,2(31319,4)	21.812,7	304632,5(537355,5)	162628,3	37352,2(138784,7)	74142,2

[(a)] O melhor valor conhecido foi obtido pelo algoritmo MSA_0 executado para cada instância com um tempo limite de 10 horas para $n \leq 500$ e 20 horas para $n > 500$, como descrito por Palubeckis (2017).

Os melhores valores são destacados em negrito.

Tabela 13 – Desempenho do GRASP-F/SRFLP instâncias Palubeckis (2017)

Instância	MSA_0		MSA_0.5		VNS-LS3		GRASP-F		
	Desv	*DPr*	*Desv*	*DPr*	*Desv*	*DPr*	*Desv*	*DPr*	$T_{médio}$
p310	0,0000	0,0001	0,0001	0,0001	0,0126	0,0167	0,0003	0,0001	1200,6
p320	0,0001	0,0002	0,0002	0,0002	0,0116	0,0112	0,0056	0,0052	1200,9
p330	0,0000	0,0000	0,0000	0,0000	0,0110	0,0230	0,0001	0,0000	1200,7
p340	0,0000	0,0000	0,0000	0,0000	0,0140	0,0098	0,0008	0,0011	1200,6
p350	0,0000	0,0000	0,0000	0,0000	0,0006	0,0008	0,0001	0,0001	1200,5
p360	0,0000	0,0000	0,0007	0,0012	0,0131	0,0076	0,0036	0,0019	1200,7
p370	0,0000	0,0000	0,0000	0,0000	0,0156	0,0106	0,0019	0,0006	1201,2
p380	0,0032	0,0040	0,0053	0,0036	0,0291	0,0212	0,0089	0,0059	1201,2
p390	0,0000	0,0000	0,0000	0,0000	0,0120	0,0278	0,0005	0,0004	1201,3
p400	0,0001	0,0002	0,0002	0,0003	0,0406	0,0327	0,0064	0,0200	1200,9
p410	0,0001	0,0001	0,0001	0,0001	0,0572	0,0239	0,0003	0,0002	1801,0
p420	0,0000	0,0000	0,0000	0,0000	0,0226	0,0248	0,0010	0,0031	1801,6
p430	0,0036	0,0023	0,0040	0,0021	0,0179	0,0142	0,0059	0,0074	1801,6
p440	0,0000	0,0000	0,0000	0,0000	0,0172	0,0230	0,0023	0,0019	1801,5
p450	0,0008	0,0019	0,0002	0,0006	0,0181	0,0133	0,0051	0,0037	1801,2
p460	0,0000	0,0000	0,0005	0,0007	0,0386	0,0258	0,0061	0,0032	1801,3
p470	0,0014	0,0010	0,0013	0,0010	0,0219	0,0191	0,0070	0,0032	1802,7
p480	0,0000	0,0000	0,0001	0,0002	0,0350	0,0149	0,0014	0,0013	1802,3
p490	0,0000	0,0000	0,0001	0,0001	0,0432	0,0338	0,0065	0,0038	1802,6
p500	0,0017	0,0020	0,0026	0,0019	0,0369	0,0154	0,0195	0,0159	1802,9
p550	0,0001	0,0002	0,0003	0,0006	0,0266	0,0187	0,0099	0,0075	3603,6
p600	0,0019	0,0011	0,0027	0,0012	0,0256	0,0160	0,0104	0,0063	3604,8
p650	0,0018	0,0025	0,0010	0,0016	0,0547	0,0257	0,0079	0,0083	3604,3
p700	0,0027	0,0038	0,0050	0,0053	0,0606	0,0219	0,0246	0,0095	3606,4
p750	0,0006	0,0005	0,0013	0,0009	0,0684	0,0148	0,0140	0,0111	3608,8
p800	0,0030	0,0015	0,0040	0,0020	0,0649	0,0253	0,0189	0,0078	3619,8
p850	0,0038	0,0043	0,0059	0,0059	0,0862	0,0280	0,0246	0,0185	3613,1
p900	0,0033	0,0020	0,0043	0,0024	0,0779	0,0219	0,0180	0,0066	3618,6
p950	0,0014	0,0026	0,0064	0,0040	0,0995	0,0261	0,0191	0,0131	3629,8
p1000	0,0035	0,0058	0,0083	0,0051	0,1226	0,0159	0,0376	0,0131	3633,8
Média	0,0011	0,0012	0,0018	0,0014	0,0385	0,0195	0,0089	0,0060	2205,7

Como apresentado na

Tabela 13, o maior desvio percentual da média do GRASP-F é muito pequeno, 0,0376% para a instância p1000. Além disso, os desvios-padrão relativos são muito baixos (o máximo observado foi 0,0200% para a instância p400). Portanto, para o conjunto com as maiores instâncias para o problema SRFLP, o algoritmo apresentou uma convergência consistente para boas soluções. Assim, pode-se considerar que o GRASP-F é uma abordagem alternativa interessante para o SRFLP, o qual não faz uso das técnicas desenvolvidas em Palubeckis (2017).

4.6 CONSIDERAÇÕES FINAIS

O algoritmo GRASP-F proposto foi executado dentro de um tempo pequeno e limitado, para várias instâncias do SRFLP, com tamanhos variando de 110 a 1000 facilidades. Os testes mostraram que o algoritmo possui uma convergência consistente para soluções de alta qualidade, mantendo um bom desempenho à medida que os tamanhos das instâncias aumentaram. O GRASP-F encontrou soluções com valores muito próximos ou iguais aos melhores valores conhecidos. Além disso, o GRASP-F melhorou os melhores valores conhecidos em 29 das 93 instâncias consideradas nos experimentos computacionais. Portanto, o algoritmo GRASP-F proposto aqui pode ser considerado uma boa alternativa para obter soluções de alta qualidade em tempos computacionais muito pequenos para problemas SRFLP considerados de grande porte.

Com os resultados do trabalho descritos neste capítulo, foi realizada a publicação em periódico de grande relevância e o artigo disponível em Cravo e Amaral (2019). Até a data da publicação do artigo, havia apenas um trabalho anterior que tratava instâncias com tamanho $n > 500$.

5 ALGORITMO PSO PROPOSTO PARA O DRLP

Neste capítulo é apresentado o algoritmo para resolver o DRLP denominado PSO-DRLP. Salienta-se que na literatura somente Guan et al. (2020) resolveram o DRLP com abordagem com estratégias heurísticas, gerando bons resultados. O presente estudo propõe-se o desenvolvimento de um método heurístico, porém com a inovação de não utilizar técnicas de decomposição do problema, isto é, não considerar o DRLP como dois subproblemas, como as abordagens até então propostas, mas, sim, considerar na resolução do DRLP pelo PSO-DRLP tanto o posicionamento das máquinas, quanto suas posições no mesmo processo iterativo.

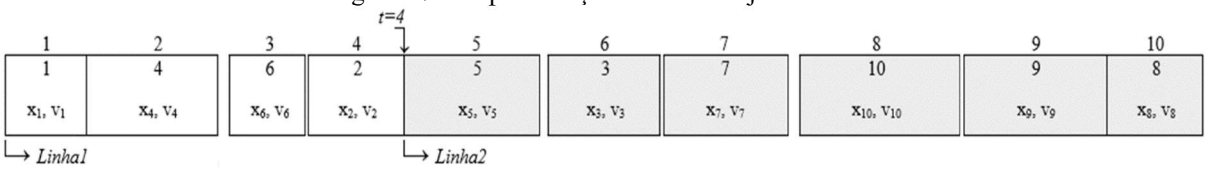
O PSO-DRLP apresenta similaridade com o DBA (GUAN et al., 2020), o qual também utiliza a meta-heurística PSO para a resolução das posições das máquinas ao longo do eixo-*x*. Entretanto, existem diferenças importantes nas duas abordagens. O PSO-DRLP é baseado na meta-heurística PSO, em que o enxame de partículas é organizado em subpopulações e o processo iterativo alterna entre duas fases de otimização. Na primeira fase, busca-se encontrar as melhores soluções em cada subpopulação. Enquanto que, na segunda fase, a busca considera as melhores soluções de cada subpopulação como um enxame de partículas elite, com o objetivo de encontrar a melhor solução do enxame global. Além disso, para melhorar a convergência do PSO-DRLP, a cada atualização das partículas pelo PSO (nova solução encontrada) é aplicado um procedimento de busca local. Enquanto que no DBA, o PSO é utilizado como resolvedor das posições absolutas das máquinas, dada a disposição encontrada na primeira fase do DBA. Outra diferença está no uso do procedimento de mutação. No PSO-DRLP não existe tal procedimento, que é empregado no DBA como uma perturbação da solução corrente discreta, executada a cada iteração do algoritmo.

5.1 GERAÇÃO DO ENXAME DE PARTÍCULAS

No algoritmo proposto, uma partícula encapsula um número inteiro t e um arranjo com n máquinas. Cada máquina contém sua identificação, sua posição x (a abscissa do ponto central de carga e descarga de material) e velocidade v. O número inteiro t define o ponto de quebra do arranjo para representar as duas linhas do DRLP. Assim, em uma linha tem-se t máquinas e, na outra linha, as (n-t) máquinas restantes. A aptidão de cada partícula é calculada pela função

objetivo, conforme Equação (2.3). A Figura 17, exemplifica a representação da solução com um arranjo de $n = 10$ máquinas e o ponto de quebra $t = 4$.

Figura 17 – Representação de um arranjo no DRLP



Fonte: próprio autor

O enxame de partículas é formado por subpopulações de partículas com o mesmo valor para t (ponto de quebra do arranjo de máquinas). O ponto de quebra t é limitado pelo intervalo [t_{min}, t_{max}] e os valores de t não são redefinidos no processo iterativo do PSO-DRLP, ou seja, uma vez definido o t na inicialização da partícula, o valor não será mais alterado.

O processo de criação do enxame inicial de partículas é mostrado no pseudocódigo do Algoritmo 8. O algoritmo recebe como parâmetros o valor m_s, que define o total de partículas em cada subpopulação; o n_s, valor que define a quantidade de subpopulações que irá compor o enxame de partículas; o valor n, que é o número de máquinas do problema; e, por fim, os valores de t_{min} e t_{max}, que definem os limitantes para os valores de t. O procedimento retorna P, um conjunto (enxame) contendo todas as partículas geradas; *Pbest*, que representa um conjunto contendo a melhor situação de cada partícula $p \in P$, inicialmente *Pbest* é igual a P; e *Gbest*, um conjunto com as melhores partículas de cada subpopulação do enxame.

Algoritmo 8 – Gerar Enxame de Partículas

entrada: m_s, n_s, n, t_{min}, t_{max}
saída: P, *Pbest*, *Gbest*
1: $t \leftarrow t_{min}$ $P \leftarrow \{\}$, $Pbest \leftarrow \{\}$, $Gbest \leftarrow \{\}$
2: **Para** $s \leftarrow 1$ **até** n_s **faça**
3: **Para** $i \leftarrow 1$ **até** m_s **faça**
4: $p \leftarrow GerarParticulaAleatoriamente\ (t, n)$
5: $P^s \leftarrow P^s \cup \{p\}$
6: $Pbest^s \leftarrow Pbest^s \cup \{p\}$
7: **Se** $i = 1$ **ou** $f(p) < f(Gbest^s)$ **então**
8: $Gbest^s \leftarrow p$
9: **Fim**
10: **Fim**
11: $t \leftarrow t + 1$
12: **Se** $t > t_{max}$ **então**
13: $t \leftarrow t_{min}$
14: **Fim**
15: **Fim**
16: **retorne** P, *Pbest, Gbest*

No Algoritmo 8, o valor de t é inicializado com t_{min} e os conjuntos P, *Pbest* e *Gbest* são inicializados vazios. No laço das Linhas 2-15, são geradas as subpopulações P^s que compõem o enxame de partículas P. No laço das Linhas 3-10, as partículas são geradas e adicionadas à subpopulação P^s do enxame. Na Linha 4, a variável p recebe a partícula gerada pelo procedimento *GerarParticulaAleatoriamente*. A partícula possui o ponto de quebra t; o arranjo de máquinas, gerado de forma aleatória; as posições x e as velocidades v de cada máquina do arranjo. Inicialmente, as posições das máquinas são calculadas considerando a ordenação do arranjo. Na Linha 5, a partícula é adicionada à subpopulação P^s do enxame P e também no enxame $Pbest^s$ (Linha 6). Na Linha 7, é verificado se a nova partícula é melhor (f é a função objetivo definida pela Equação 2.3) que a melhor partícula da subpopulação P^s, denominada $Gbest^s$. Caso afirmativo, a partícula $Gbest^s$ é atualizada (Linha 8). Na Linha 11, o valor de t é incrementado de uma unidade e a Linha 12 verifica se o limite para t foi excedido, reiniciando para t_{min}, se necessário. Ao final, o procedimento retorna P, *Pbest* e o *Gbest* (Linha 16).

O enxame inicial de partículas é criado composto por n_s subpopulações. Vale ressaltar que, se $n_s = t_{max} - t_{min} + 1$, cada subpopulação está associada a um valor de t, ou seja, as subpopulações se distinguem por terem soluções com pontos de quebra (valores de t) distintos. Entretanto, a quantidade de subpopulações n_s poderá ser maior que $t_{max} - t_{min} + 1$ e, assim, mais de uma subpopulação pode conter partículas com o mesmo valor de t. Na implementação proposta, $n_s \geq t_{max} - t_{min} + 1$. Após essa etapa, o PSO-DRLP realizará a busca por soluções através das atualizações das velocidades e posições de cada partícula contida no enxame P fazendo uso das informações de *Pbest* e *Gbest*.

5.2 ATUALIZAÇÃO DA VELOCIDADE E POSIÇÃO DAS PARTÍCULAS

O vetor velocidade é atualizado pela Equação (3.3), na qual é definido o fator de inércia ω, não sendo então utilizados os limitantes $[v_{min}, v_{max}]$ para a velocidade nas partículas.

No processo de atualização do vetor posição de cada partícula do PSO, as posições das máquinas são recalculadas, definindo assim, onde cada máquina será alocada ao longo do corredor, e, implicitamente, são definidas as distâncias dos pares (i, j) de máquinas no arranjo. Após a atualização das posições, um procedimento verifica se existem sobreposições de máquinas nas duas linhas do arranjo, utilizando um procedimento que percorre os pares de

máquinas (i, j; $i < j$) em cada linha do arranjo verificando se as máquinas i e j estão sobrepostas. Se as máquinas i e j estão sobrepostas, a posição de j é redefinida para um ponto adjacente a i, ou seja, $x_i = x_j + 0{,}5 \times (l_i + l_j)$, removendo assim a sobreposição.

5.3 BUSCA LOCAL

No processo de atualização das partículas, as posições das máquinas ao longo do eixo-x são redefinidas e sobreposições são corrigidas, caso ocorram. Consequentemente, novas distâncias entre as máquinas são determinadas, gerando assim, um layout viável para o DRLP. Contudo, a disposição das máquinas no arranjo não é afetada. Deste modo, para intensificar a busca por novas soluções, foi adicionada ao processo iterativo do PSO uma busca local composta de duas fases. As fases da busca local alternam a utilização de movimentos de inserção e troca 2-opt. Esses movimentos são amplamente utilizados em procedimentos heurísticos propostos para os diversos problemas de layout de facilidades, na tentativa de intensificar a busca no espaço de soluções (OZCELIK, 2012; AMARAL, 2012; KOTHARI; GHOSH, 2013b; PALUBECKIS, 2015, 2017; CELLIN, 2017; MURRAY; SMITH; ZHANG, 2013; CRAVO; AMARAL; 2019).

Na primeira etapa da busca local são geradas sequencialmente os pares de posições (r, s), na tentativa de inserir a máquina da posição $r = 1, ..., t$, na nova posição $s = 1, ..., t$ com $r \neq s$. Em seguida, são gerados os pares (r, s) para as posições $r = t + 1, ..., n$, na tentativa de inserir a máquina da posição r nas posições $s = t + 1, ..., n$; com $r \neq s$. Caso a nova solução vizinha melhore o valor da função objetivo corrente, a solução corrente é atualizada e a busca continua para a próxima solução vizinha, realizando o movimento de inserção para o próximo par (r, s). Para manter o ponto de quebra t inalterado na solução, nessa primeira fase, não há inserção de máquinas de uma linha em outra do arranjo, permanecendo assim, a partícula com o mesmo valor de t da sua subpopulação no enxame de partículas.

Após o término da primeira etapa, a solução é submetida a uma segunda etapa da busca local, em que são geradas soluções vizinhas realizando movimentos de troca 2-opt. Para isso, são percorridas de forma sequencial as posições $p = 1, ..., n$ e $q = 1, ..., n$; $p \neq q$, na tentativa de obter uma solução vizinha melhor que a solução corrente. Caso a nova solução vizinha tenha o valor da função objetivo melhor que a solução corrente, a solução corrente é atualizada e a busca continua para a próxima solução vizinha, realizando o movimento 2-opt para o próximo par (p, q). Nessa rotina, mesmo sendo realizados movimentos entre máquinas em linha opostas,

não há alteração no ponto de quebra *t*, pois a quantidade de máquinas em ambas as linhas não é alterada. O Algoritmo 9 apresenta o pseudocódigo da busca local proposta. O procedimento recebe como parâmetros S_0, que representa a solução corrente do PSO-DRLP; o número de máquinas do problema, *n*; e o valor de *t* da solução S_0. Ao final, retorna uma solução ótima local *S*.

Algoritmo 9 – Busca Local para o PSO-DRLP

```
entrada: S0, n, t
saída: S
 1: S ← S0
 2: Faça
 3:    melhorou ← 0
         /* FASE 1 */
 4:    Para r ← 1 até t faça
 5:       Para s ←1 até t faça
 6:          Se r ≠ s então
 7:             S' ← ObterVizinhoMovimentoInserção (S, r, s)
 8:             Se f(S') < f(S) então
 9:                S ← S'
10:                melhorou ← 1
11:             Fim
12:          Fim
13:       Fim
14:    Fim
15:    Para r ← t + 1 até n faça
16:       Para s ← t + 1 até n faça
17:          Se r ≠ s então
18:             S' ← ObterVizinhoMovimentoInserção (S, r, s)
19:             Se f(S') < f(S) então
20:                S ← S'
21:                melhorou ← 1
22:             Fim
23:          Fim
24:       Fim
25:    Fim
         /* FASE 2 */
26:    Para p ← 1 até n faça
27:       Para q ← 1 até n faça
28:          Se p ≠ q então
29:             S' ← ObterVizinhoMovimento2-opt (S, p, q)
30:             Se f(S') < f(S) então
31:                S ← S'
32:                melhorou ← 1
33:             Fim
34:          Fim
35:       Fim
36:    Fim
37: Enquanto melhorou = 1
38: retorne S
```

Na Linha 1 do Algoritmo 9, a solução *S* é inicializada com a solução S_0. No laço das Linhas 2-37, o algoritmo repete, enquanto uma solução vizinha melhor for encontrada. Na Linha 3, a *flag melhorou* é inicializada com zero. A primeira parte da busca local ocorre nas Linhas 4 à 25,

onde são geradas e avaliadas soluções vizinhas utilizando os movimentos de inserção. Nos laços das Linhas 4-14 e 5-13 são gerados os pares de posições referentes ao arranjo da Linha 1 do layout. O procedimento *ObterVizinhoMovimentoInserção*, Linha 7, retorna uma solução vizinha de *S* que é atribuída à *S′*. Na Linha 8, é verificado se a solução *S′* é melhor que a solução corrente *S*. Caso afirmativo, a solução *S* é atualizada (Linha 9), e a *flag melhorou* recebe o valor 1 (Linha 10). De forma semelhante, nos laços das Linhas 15-16 são gerados os pares de posições referente ao arranjo da Linha 2, para serem avaliados. Na Linha 18, uma solução vizinha *S′* é obtida pelo movimento de inserção, e, na Linha 19, é verificado se a nova solução *S′* é melhor que a solução corrente *S*. Se verdadeiro, a solução *S* é atualizada na Linha 20 e a *flag melhorou* recebe o valor 1 (Linha 21).

A segunda parte da busca local, em que são utilizados os movimentos 2-opt, ocorre nas Linhas 26 à 36. Para cada valor de *p*, o laço das Linhas 27-35 gera os valores das posições *q*. Na Linha 29, uma solução vizinha *S′* é obtida pelo procedimento *ObterVizinhoMovimento2-opt*, com a aplicação do movimento 2-opt, considerando as posições *p* e *q*. Na Linha 30, é verificado se a nova solução *S′* é melhor que a solução corrente *S*. Caso afirmativo, a solução corrente *S* é atualizada e a *flag melhorou* recebe o valor 1 (Linhas 31 à 32). Ao final do procedimento, o algoritmo retorna à solução *S* (Linha 38).

Vale destacar dois pontos interessantes na busca local proposta. O primeiro é que, cada fase da busca local não é executada como uma rotina de busca local padrão, em que uma lista de pares (*p*, *q*) é gerada e a busca é realizada sobre a lista, até que não seja encontrada uma solução vizinha melhor. Como pode ser observado no Algoritmo 9, ambas as etapas realizam um número exato de movimentos, sem reinício, o que difere de uma busca local padrão, sendo que na primeira etapa são aplicados movimentos de inserção, e, na segunda etapa, movimentos 2-opt. Após finalizar a segunda etapa, caso tenha ocorrido alguma melhora, o processo é reiniciado com a solução corrente.

O segundo ponto é que as rotinas da busca local realizam trocas ou movimentos de inserção considerando a posição absoluta das facilidades. Assim, mesmo que a ordenação das máquinas no arranjo seja mantida, pois apenas as posições absolutas são afetadas pela atualização da partícula no PSO (Equação 3.2), um movimento de inserção ou 2-opt, que antes não havia melhorado a solução corrente, agora, com as novas posições absolutas das máquinas, poderá melhorar e, se assim for, ocorrerá uma alteração na disposição das máquinas no arranjo, possibilitando então uma intensificação do enxame de partícula.

5.4 O ALGORITMO PSO PROPOSTO

Como dito antes, o algoritmo PSO-DRLP é composto por duas fases. Inicialmente são gerados os exames *P* e o correspondente *Pbest*, que armazena as melhores posições no espaço de busca de cada partícula de *P*. Além do conjunto *Gbest*, com as melhores partículas de cada subpopulação, a melhor solução (partícula) global, g_{best}, obtida. Na primeira fase do PSO-DRLP, para cada subpopulação, são realizadas as atualizações das partículas (velocidade e posição das máquinas). A cada atualização de uma partícula *p*, é realizada a verificação e ajuste, se necessário, para eliminar possíveis sobreposições de máquinas no arranjo. Em seguida, como a partícula *p* contém uma nova solução, esta é submetida à busca local. Ao final de cada iteração, é verificado se a partícula *p* melhorou em relação à sua melhor posição e também em relação à melhor solução da sua subpopulação. A cada iteração, a melhor partícula global, g_{best}, é mantida atualizada.

A ideia da primeira fase é que o PSO otimize cada subpopulação individualmente, não realizando troca de informação entre as subpopulações. Assim, cada partícula leva em consideração sua melhor posição e a melhor partícula da subpopulação a qual pertence. Na segunda fase, o PSO irá otimizar as partículas pertencentes ao conjunto *Gbest*, que formam uma população elite com os melhores de cada subpopulação, introduzindo, assim, uma comunicação entre as subpopulações, sendo utilizadas as informações das partículas em *Gbest* e também a melhor partícula global g_{best}. O PSO-DRLP é mostrado no pseudocódigo apresentado no Algoritmo 10. O Algoritmo 10, descrito a seguir, recebe como entrada os seguintes parâmetros:

- n: o tamanho do problema tratado;
- m_s, número de partículas para cada subpopulação;
- n_s, a quantidade de subpopulações;
- T_{max}, tempo máximo de execução do algoritmo;
- $tempo_{\omega}$, tempo de referência para o decremento do fator de inércia ω;
- ω_{max}, valor máximo inicial para o fator de inércia ω;
- ω_{min}, valor mínimo final para o fator de inércia ω;
- c_1: coeficiente da componente de auto reconhecimento;
- c_2: coeficiente da componente social; e
- k: valor inteiro para a definição do intervalo $[t_{min}, t_{max}]$.

Algoritmo 10 – Algoritmo PSO-DRLP proposto

entrada: n, m_s, n_s, T_{max}, $tempo_{\omega}$, ω_{max}, ω_{min}, c_1, c_2, k
saída: g_{best}

1: $t_{min} \leftarrow max\ \{1; \left\lfloor \frac{n}{2} \right\rfloor\}$, $t_{max} \leftarrow min\ \{n; \left\lfloor \frac{n}{2} \right\rfloor + k\}$
2: $[P, Pbest, Gbest] \leftarrow GerarEnxamePartículas\ (m_t, n_s, n, t_{min}, t_{max})$
3: $\omega \leftarrow \omega_{max}$, $tempo_{base} \leftarrow tempo_{corrente}$
4: $g_{best} \leftarrow \underset{1 \le s \le n_s}{argmin}\ \{Gbest^s\}$
5: **Enquanto** $tempo_{corrente} < T_{max}$ **faça** /* FASE 1 */
6: $melhorou \leftarrow V$
7: **Enquanto** $melhorou = V$ **E** $tempo_{corrente} < T_{max}$ **faça**
8: $melhorou \leftarrow F$
9: **Para** $s \leftarrow 1$ **até** n_s **faça**
10: **Para cada** $p_i \in P^s$ **faça**
11: $p_{i,v} \leftarrow \omega p_v + r_1 c_1 (Pbest^s_{i,x} - p_{i,x}) + r_2 c_2 (Gbest^s_x - p_{i,x})$
12: $p_{i,x} \leftarrow p_{i,x} + p_{i,v}$
13: $p_i \leftarrow VerificarSobreposiçãoAjustar\ (p_i)$
14: $p_i \leftarrow BuscaLocal\ (p_i, n, t)$
15: **Se** $f(p_i) < f(Pbest^s_i)$ **então**
16: $Pbest^t_i \leftarrow p_i$
17: **Se** $f(p_i) < f(Gbest^s)$ **então**
18: $Gbest^s \leftarrow p_i$
19: **Se** $f(p_i) < f(g_{best})$ **então**
20: $g_{best} \leftarrow p_i$, $melhorou \leftarrow V$
21: **Fim**
22: **Fim**
23: **Fim**
24: **Fim**
25: **Fim**
26: **Fim**
27: $P^{Gbest} \leftarrow Gbest$; $Pbest^{Gbest} \leftarrow Gbest$ /* FASE 2 */
28: $melhorou \leftarrow V$
29: **Enquanto** $melhorou = V$ **E** $tempo_{corrente} < T_{max}$ **faça**
30: $melhorou \leftarrow F$
31: **Para cada** $p_i \in P^{Gbest}$ **faça**
32: $p_{i,v} \leftarrow \omega p_{i,v} + r_1 c_1 (Pbest^{Gbest}_{i,x} - p_{i,x}) + r_2 c_2 (g_{best,x} - p_{i,x})$
33: $p_{i,x} \leftarrow p_{i,x} + p_{i,v}$
34: $p_i \leftarrow VerificarSobreposiçãoAjustar\ (p_i)$
35: $p_i \leftarrow BuscaLocal\ (p_i, n, t)$
36: **Se** $f(p_i) < f(Pbest^{Gbest}_i)$ **então**
37: $Pbest^{Gbest}_i \leftarrow p_i$
38: **Se** $f(p_i) < f(g_{best})$ **então**
39: $g_{best} \leftarrow p_i$, $melhorou \leftarrow V$
40: **Fim**
41: **Fim**
42: **Fim**
43: **Fim**
44: $Gbest \leftarrow Pbest^{Gbest}$
45: $tempo_{decorrido} \leftarrow tempo_{corrente} - tempo_{base}$
46: $\omega \leftarrow \omega_{max} - (\omega_{max} - \omega_{min}) \times (tempo_{decorrido}/tempo_{\omega})$
47: **Se** $(tempo_{decorrido} \ge tempo_{\omega})$ **então**
48: $tempo_{base} \leftarrow tempo_{corrente}$
49: $\omega \leftarrow \omega_{max}$
50: **Fim**
51: **Fim**
52: **retorne** g_{best}

Na Linha 1, são definidos os valores mínimos e máximo, t_{min} e t_{max}, utilizados para os pontos de quebra dos arranjos. Na Linha 2, são inicializados o enxame de partículas inicial *P*, *Pbest* e *Gbest*. Na Linha 3 é inicializado o fator de inércia ω e a variável $tempo_{base}$ com o tempo atual. A variável $tempo_{base}$ armazena o tempo atual e é utilizado para a atualização do valor de ω na Linha 46. Na Linha 4, a melhor solução encontrada pelo PSO-DRLP, g_{best}, é inicializada com a melhor solução do conjunto *Gbest*. No laço das Linhas 5-51 ocorre o processo iterativo do PSO-DRLP.

Na Linha 6, a *flag melhorou* é iniciada com *F*. A primeira fase do PSO-DRLP é definida no laço das Linhas 7-26. Nessa primeira fase, são percorridas todas as subpopulações, o laço das Linhas 9-25, identificadas pelo índice *s* superior nos conjuntos *P*, *Pbest* e *Gbest*. No laço das Linhas 10-24, as partículas pertencentes a cada subpopulação P^s são atualizadas. As velocidades ($p_{i,v}$) das máquinas de cada partícula p_i são recalculadas na Linha 11 e, na Linha 12, as posições ($p_{i,x}$) são atualizadas. Após a atualização, é verificado se ocorreu alguma sobreposição de máquinas no arranjo e, se necessário, as posições são ajustadas (Linha 13). Na Linha 14 a partícula p_i é atualizada com a solução retornada da busca local. Na Linha 15, é verificado se a nova partícula p_i melhora em relação à sua melhor posição até então, $Pbest_i^s$. Se verdadeiro, $Pbest_i^s$. é atualizada na Linha 16. Na Linha 17 é verificado se a partícula corrente p_i é melhor em relação à melhor partícula, $Gbest^s$, da subpopulação *s*. Caso verdadeiro, a partícula $Gbest^s$ é atualizada na Linha 18. Na Linha 19, a partícula p_i é comparada com a melhor solução corrente g_{best}. Se verdadeiro, a solução g_{best} e *flag melhorou* são atualizadas na Linha 20. Ao final da primeira fase, que termina quando não houver melhoria na solução g_{best} ou o tempo de execução tenha se esgotado, tem-se a melhor solução do PSO-DRLP, g_{best} e as melhores soluções de cada subpopulação *Gbest*, que serão utilizadas na segunda fase do PSO-DRLP.

A segunda fase do PSO-DRLP inicia na Linha 27 com a definição dos enxames P^{Gbest} e $Pbest^{Gbest}$, que são inicializados com as partículas contidas no conjunto *Gbest*. Na Linha 28, a *flag melhorou* é reiniciada com *V*. A partir desse ponto, no laço das Linhas 29-43, tem-se a segunda fase do PSO-DRLP. No laço das Linhas 31-42, são atualizadas as partículas em P^{Gbest}, $Pbest^{Gbest}$ e a melhor solução global g_{best}, utilizando o princípio da meta-heurística PSO. Na Linha 32, o vetor velocidade *v* de cada partícula é atualizada e, na Linha 33, o vetor posições *x* é recalculado. Na Linha 34, verifica se ocorrem sobreposições no novo arranjo, e, se necessário, efetua a correção. Na Linha 35, a partícula é submetida à busca local. Na Linha 36 é verificado se a solução atual melhora em relação à sua melhor posição $Pbest_i^{Gbest}$. Se verdadeiro, atualiza a partícula $Pbest_i^{Gbest}$ (Linha 37). Na Linha 38, é verificado se a partícula corrente é melhor que

a melhor solução global do PSO-DRLP. Se verdadeiro, g_{best} e a *flag melhorou* são atualizadas na Linha 39. A final da segunda fase, que é encerrada quando não ocorrer melhoria na solução g_{best} ou pelo esgotamento do tempo de execução, atualiza-se o conjunto *Gbest* com as soluções obtidas pela segunda fase do PSO-DRLP (Linha 44), que serão utilizadas na primeira fase do PSO-DRLP na próxima iteração.

Na Linha 45, obtém-se o tempo decorrido. O valor de ω é recalculado, levando em consideração a variação do tempo de execução da atualização de todas as partículas ($tempo_{decorrido}$) e o tempo de referência $tempo_{\omega}$ (Linha 46). Na Linha 47, o tempo decorrido é verificado em relação ao limite de referência $tempo_{\omega}$. Se verdadeiro, as variáveis $tempo_{base}$ e ω são reiniciadas, respectivamente, nas Linhas 48 e 49. Ao final, o algoritmo PSO-DRLP retorna a melhor solução global *g* encontrada (Linha 52).

O uso da informação do tempo decorrido e o tempo de referência $tempo_{\omega}$ é utilizado para manter o padrão de decrescimento de ω, independentemente do tempo máximo de execução do algoritmo (T_{max}).

5.5 EXPERIMENTOS COMPUTACIONAIS PARA O PSO-DRLP

Para a realização dos experimentos computacionais, foi utilizado um laptop com processador Intel i7-7500U 2,7 GHz com 8GB de memória com sistema operacional MS Windows 10. O PSO-DRLP foi implementado em linguagem de programação C e compilados no compilador GCC 5.1.0 x64 com as *flags* de otimização -O3.

Nos experimentos foram utilizadas trinta e nove instâncias com matriz de fluxos simétricas e folgas implícitas, sendo seis instâncias para cada $n = \{11, 12, 13\}$ de Amaral (2020); uma com $n = 15$ (AMARAL, 2006); duas com $n = 16$ (AMARAL, 2013a); uma com $n = 17$ e uma com $n = 18$ de (AMARAL, 2008); cinco instâncias com $n = 30$ (ANJOS; VANNELLI, 2008); sete com $n = 40$ (AMARAL, 2020); e quatro instâncias com tamanhos $n = \{60, 70, 75, 80\}$ de Anjos, Kennings e Vannelli (2005). Também são consideradas doze instâncias com matriz de fluxos assimétricos e folgas explícitas, sendo sete para cada $n = \{15, 20, 25, 30, 35, 40, 45\}$ e cinco com $n = 50$ de Murray, Smith e Zhang (2013). Tais instâncias foram consideradas na presente análise por serem as mais complexas e utilizadas nas comparações dos estudos mais recentes (AMARAL, 2020, GUAN et al., 2020).

5.5.1 Calibração dos Parâmetros do PSO-DRLP

No PSO-DRLP tem-se os seguintes parâmetros que precisam ser definidos: m_s, número de partículas do enxame; n_s, número de subpopulações do exame de partículas; $tempo_\omega$, tempo de referência para o decremento do fator de inércia ω; ω_{max} e $\omega_{min,}$ que formam o intervalo para o fator de inércia ω; e as constantes c_1 e c_2. Além disso, os parâmetros n e k são fixos, sendo n definido pelo tamanho da instância e $k = 4$ (AMARAL, 2020).

Para determinar os parâmetros do PSO-DRLP, utilizou-se a meta-heurística PSO de calibração (APÊNDICE B). Assim, para o PSO de calibração, foi definido o seguinte vetor de parâmetros e seus limites: m_s, $5 \leq x_{i1} \leq 100$; n_s, $5 \leq x_{i2} \leq 100$; $tempo_\omega$, $30 \leq x_{i3} \leq 120$; ω_{max}, $0{,}4 \leq x_{i4} \leq 0{,}9$; ω_{min}, $0{,}4 \leq x_{i5} \leq \omega_{max}$; c_1, $0 \leq x_{i6} \leq 2$; e c_2, $0 \leq x_{i7} \leq 2$ para cada partícula i do PSO de calibração, sendo $T_{max} = tempo_\omega$.

O PSO de calibração foi executado para um subconjunto contendo quatro instâncias de diferentes complexidades, sendo duas instâncias com matriz de fluxos simétricas e folgas implícitas (N30_03 e DRLP_02) e duas instâncias com matriz de fluxos assimétricas e folgas explícitas (Murray243 e Murray322). Os parâmetros usados no PSO de calibração foram: m = 10 (dez partículas) e $IterMax$ = 36 (total de iterações do PSO de calibração). Os resultados da calibração são mostrados na Tabela 14.

Tabela 14 – Parâmetros encontrados para o PSO-DRLP

Instância	Parâmetro	$tempo_\omega$	m_s	n_s	ω_{max}	ω_{min}	c_1	c_2
N30_02	P1	79	76	86	0,708602	0,510998	1,251112	0,050853
DRLP_02	P2	91	48	99	0,405524	0,404195	1,565725	1,133444
Murray243	P3	52	28	75	0,483970	0,475621	0,873352	0,754753
Murray322	P4	92	75	59	0,643440	0,499749	0,977237	0,266861

Como pode ser visto na Tabela 14, para cada instância utilizada na calibração, o PSO de calibração encontrou um conjunto de parâmetros. Assim, para determinar o melhor conjunto, o PSO-DRLP foi executado utilizando cada conjuntos de parâmetros para as 4 instâncias uma única vez. Em cada execução, foi definido um tempo limite T_{max} = 120. As médias dos resultados são mostrados na Tabela 15. A tabela é estruturada da seguinte forma: a primeira coluna apresenta o nome do conjunto de parâmetros e a segunda coluna mostra a média das

soluções encontradas pelo PSO-DRLP, para cada parâmetro. A qualidade do conjunto de parâmetros é medida pelo valor da coluna *Média*. Quanto menor o valor, melhor o parâmetro.

Tabela 15 – Desempenho do PSO-DRLP para cada conjunto de parâmetros

Parâmetro	*Média*
P1	1476081,8
P2	1476405,9
P3	1476734,5
P4	**1475791,7**

Pelos resultados da Tabela 15, o conjunto de parâmetros denominado P4 (em negrito) apresentou o melhor desempenho, tendo o menor valor médio dentre os demais (1475791,7). Assim, o conjunto de parâmetros denominado P4 ($tempo_{\omega}$ = 92; m_s = 75; n_s = 59; ω_{max} = 0,643440; ω_{min} = 0,499749; c_1 = 0,977237; c_2 = 0,266861) foi utilizado nos experimentos computacionais para o PSO-DRLP. Ressalta-se que a média para os quatro conjuntos de parâmetros testados apresenta-se com valores muito próximos, o que evidencia a robustez do algoritmo proposto.

5.5.2 Critério de avaliação do desempenho

Para a avaliação do PSO-DRLP, foram realizadas dez execuções para cada instância. A melhor solução obtida pelo PSO-DRLP é denotada por *Melhor*. A média dos valores das soluções encontradas pelo PSO-DRLP é denominada *Média*. Também são apresentados o desvio-padrão (*DP*) e a medida de desempenho *Desv*. O desvio percentual relativo (*Desv*) é calculado como $Desv = 100 \times \frac{Média - v^*}{v^*}$, onde v^* é a melhor solução da literatura. O tempo limite de execução do PSO-DRLP, T_{max}, é informado em segundos.

5.5.3 Experimentos para instâncias com matriz de fluxo simétrica e folgas implícitas

A Tabela 16 apresenta a comparação dos resultados dos experimentos com o PSO-DRLP para as instâncias com tamanhos $11 \leq n \leq 18$ máquinas. Essas instâncias apresentam uma matriz de fluxos simétrica e os valores das folgas implícitos. Para as instâncias com tamanhos $11 \leq n \leq 16$, os melhores valores conhecidos são ótimos, sendo $11 \leq n \leq 13$ obtidos em Amaral (2018),

n = 15 em Secchin e Amaral (2018) e n = 16 em Chae e Regan (2020). Para a instância Am17, o melhor valor conhecido foi obtido pelas heurísticas em Amaral (2020). O valor ótimo para a instância Am18 não é conhecido, sendo o valor disponível (5372,5), obtido pelo PSO-DRLP com T_{max} = 3600s.

A Tabela 16 está estruturada da seguinte forma: a primeira coluna mostra os nomes das instâncias; a segunda coluna, os tamanhos das instâncias; a terceira coluna apresenta os melhores valores conhecidos; na quarta coluna são mostradas as diferenças entre as melhores soluções obtidas pelo PSO-DRLP nas 10 execuções e as melhores soluções conhecidas, que são dados por Dif_{Melhor} = ($Melhor - v^*$), seguidas das diferenças entre valores médios das soluções obtidas e a melhor solução conhecida, definidas como Dif_{Media} = ($Média - v^*$); a sexta coluna mostra os valores de *Desv*; os desvios-padrão são mostrados na coluna *DP*; e a coluna $T_{Médio}$ apresenta a média dos tempos (T_{max}) de execução utilizados para cada instância, em segundos. A última linha da tabela apresenta a média sobre todas as colunas da tabela.

Tabela 16 – Resultados para instâncias de pequeno porte com $11 \leq n \leq 15$ máquinas

Instância	n	$v^{*(a)}$	Dif_{Melhor}	$Dif_{Média}$	*Desv*	*DP*	$T_{Médio}$
Am11a	11	**5559,0**	**0**	0	0,000	0,0	5,4
Am11b	11	**3655,5**	**0**	0	0,000	0,0	5,2
Am11c	11	**3832,5**	**0**	0	0,000	0,0	5,2
Am11d	11	**906,5**	**0**	0	0,000	0,0	5,3
Am11e	11	**578,0**	**0**	0	0,000	0,0	5,3
Am11f	11	**825,5**	**0**	0	0,000	0,0	5,2
Am12a	12	**1493,0**	**0**	0	0,000	0,0	5,4
Am12b	12	**1606,5**	**0**	0	0,000	0,0	5,2
Am12c	12	**2012,5**	**0**	0	0,000	0,0	5,4
Am12d	12	**1107,0**	**0**	0	0,000	0,0	5,4
Am12e	12	**1066,0**	**0**	0	0,000	0,0	5,2
Am12f	12	**997,5**	**0**	0	0,005	0,2	5,3
Am13a	13	**2456,5**	**0**	0	0,000	0,0	5,5
Am13b	13	**2864,0**	**0**	0	0,000	0,0	5,3
Am13c	13	**4136,0**	**0**	0	0,000	0,0	5,4
Am13d	13	**6164,5**	**0**	0	0,000	0,0	5,3
Am13e	13	**6502,5**	**0**	0	0,000	0,0	5,6
Am13f	13	**7699,5**	**0**	0	0,000	0,0	5,8
Am15	15	**3195,0**	**0**	0,6	0,019	0,5	5,0
P16a	16	**7365,5**	**0**	0	0,000	0,0	5,3
P16b	16	**5870,5**	**0**	0,3	0,005	0,9	5,2
Am17	17	**4655,0**	**0**	0	0,000	0,0	5,6
Am18	18	**5372,5**	**0**	2,4	0,045	2,1	5,5
Média		3506,4	0	0,1	0,003	0,2	5,3

[a] Valor ótimo para $11 \leq n \leq 13$ (AMARAL, 2018), para n = 15 (SECCHIN; AMARAL, 2018) e n = 16 (CHAE; REGAN, 2020). O melhor valor conhecido para Am17 foi obtido por Amaral (2020) e o melhor valor conhecido para Am18 foi obtido pelo PSO-DRLP com *Tmax*=3600s. Os melhores valores são destacados em negrito.

Para o conjunto de testes da Tabela 16, o PSO-DRLP encontrou as melhores soluções da literatura para todas as instâncias. O algoritmo apresenta robustez, como pode ser notado pela baixa média dos valores de desvios percentuais e de desvio-padrão, apresentados

respectivamente nas colunas *Desv* e *DP*, a um custo computacional extremamente baixo, conforme indicado na coluna $T_{médio}$.

A Tabela 17 apresenta os resultados para as instâncias do DRLP com $30 \leq n \leq 40$ máquinas. Os resultados são comparados com os melhores resultados disponíveis até a presente data. Os valores em v^* foram obtidos por Amaral (2020) a partir de um computador Intel Core i7-3770 CPU 3,40 GHZ com 8 GB de RAM e usando o solver CPLEX 12.7.1.0 para resolver o modelo de programação linear. A tabela está estruturada como a Tabela 16.

Para tornar a comparação entre computadores com processadores diferentes mais justa, os tempos de execução do PSO-DRLP foram ajustados de acordo com os parâmetros *CPU Mark* (CPUBENCHMARK, 2020). Assim, os tempos limites de execução, T_{max}, utilizados pelo PSO-DRLP são iguais aos tempos do algoritmo *Heuristica2+LP* (AMARAL, 2020) multiplicados pelo fator 1,7184. Os tempos apresentados são as médias dos tempos efetivos de execução do PSO-DRLP, podendo sofrer variações dependendo do tempo de execução de cada iteração do algoritmo.

Tabela 17 – Comparação dos resultados para as instâncias de médio porte com $30 \leq n \leq 40$ máquinas

Instância	n	$v^{*(a)}$	Dif_{melhor}	$Dif_{média}$	*Desv*	*DP*	$T_{médio}$
N30_01	30	**4115,0**	**0**	0	0,000	0	3413,9
N30_02	30	**10771,0**	**0**	0	0,000	0	2031,9
N30_03	30	**22692,0**	**0**	2,1	0,009	2,7	1665,1
N30_04	30	**28390,0**	**0**	1,5	0,005	1,4	1676,0
N30_05	30	**57393,5**	**0**	0,2	0,000	0,6	1463,1
DRLP_01	40	99525,5	**-17,5**	**-0,6**	-0,001	11,0	5436,6
DRLP_02	40	300973,5	**-6,5**	17,4	0,006	22,0	5322,6
DRLP_03	40	**416257,0**	3	21,1	0,005	12,0	5404,6
DRLP_04	40	**207510,0**	7	20,5	0,010	9,6	5602,0
DRLP_05	40	**193748,0**	**0**	20,6	0,011	16,1	5371,5
DRLP_06	40	**1881277,0**	84	135,8	0,007	47,0	5282,0
DRLP_07	40	545239,0	**-140**	138,1	0,025	197,3	5196,1
Média		313991,0	-5,8	29,7	0,006	26,6	3988,8

[a] O melhor valor conhecido foi obtido em Amaral (2020).
Os valores mínimos estão destacados em negrito.

Para as instâncias da Tabela 17, o PSO-DRLP alcançou as melhores soluções da literatura para todas as instâncias com $n = 30$, sendo que para as instâncias N30_01 e N30_02, o PSO-DRLP encontrou os melhores valores conhecidos nas 10 execuções, como pode ser observado na coluna *DP*. Para as instâncias com $n = 40$, o PSO-DRLP alcançou uma solução conhecida (DRLP_05) e obteve soluções melhores para três instâncias (DRLP_01, DRLP_02 e DRLP_07).

O PSO-DRLP é capaz de gerar soluções de alta qualidade, conforme os baixos valores dos desvios percentuais relativos. Além disso, pelos baixos valores dos desvios-padrão obtidos, o algoritmo apresenta uma convergência consistente para esse conjunto de instâncias. Mesmo não tendo encontrado as melhores soluções para as instâncias DRLP_03, DRLP_04 e DRLP_06, as soluções encontradas para essas três instâncias estão bem próximas dos melhores valores conhecidos. De modo geral, o PSO-DRLP apresentou valor menor de média geral das melhores soluções obtidas em relação às soluções de referência.

A Tabela 18 apresenta os resultados encontrados para as instâncias do PSO-DRLP com $60 \leq n \leq 80$ máquinas que, até a presente data, são as maiores instâncias consideradas na literatura para o DRLP. Essas instâncias também possuem a matriz de fluxos simétrica e os valores implícitos das folgas. As soluções do PSO-DRLP são comparadas com os melhores valores da literatura, obtidos pelo DBA (GUAN et al., 2020). A Tabela 18 está estruturada conforme a Tabela 16.

Os tempos de execução do PSO-DRLP foram ajustados de acordo com os parâmetros de comparação entre processadores (CPUBENCHMARK, 2020). Deste modo, os tempos utilizados em Guan et al. (2020) foram multiplicados pelo fator 1,2907 e utilizados como tempos de execução do PSO-DRLP.

Tabela 18 – Comparação dos resultados para instâncias com $60 \leq n \leq 80$ máquinas

Instância	n	$v^{*(a)}$	Dif_{melhor}	$Dif_{média}$	$Desv$	DP	$T_{médio}$
AKV60_01	60	739233,0	**-34,0**	162,6	0,022	93,0	395,3
AKV70_01	70	**764511,0**	286,0	505,7	0,066	136,6	783,1
AKV75_01	75	**1197240,0**	368,5	645,2	0,054	181,5	1037,4
AKV80_01	80	**1034735,5**	250,0	456,0	0,044	143,5	1385,0
Média		933929,9	217,6	442,3	0,047	138,6	900,2

[(a)] O melhor valor conhecido foi obtido em Guan et al. (2020).
Os valores mínimos estão destacados em negrito.

Para os testes mostrados na Tabela 18, o PSO-DRLP conseguiu melhorar o melhor valor conhecido de uma instância, AKV60_01. Para as outras instâncias o PSO-DRLP obteve valores muito próximos dos melhores disponíveis, tendo o maior desvio percentual de 0,066%, para a instância AKV70_01. Os baixos valores dos desvios-padrão mostram que o PSO-DRLP mantém uma capacidade de convergência consistente para esse conjunto de testes com as maiores instâncias tratadas neste trabalho.

5.5.4 Experimentos para instâncias com matriz de fluxo assimétrica e folgas explícitas

Nesse conjunto de testes, foram consideradas nove instâncias apresentadas em Murray, Smith e Zhang (2013). As instâncias têm tamanhos de $15 \leq n \leq 50$ máquinas com matriz de fluxos assimétrica e folgas explícitas. Para as sete instâncias com $15 \leq n \leq 45$ máquinas, os melhores resultados foram obtidos pelo DBA (GUAN et al., 2020) e para as seis instâncias com $n = 50$ máquinas, os melhores resultados foram encontrados pelo algoritmo *Heuristic3+LP* (AMARAL, 2020). A Tabela 19 apresenta a comparação dos resultados encontrados pelo PSO-DRLP com as melhores soluções da literatura e tem a mesma estrutura da Tabela 16. Os tempos computacionais utilizados pelo PSO-DRLP foram ajustados como descritos nas subseções anteriores.

Tabela 19 – Comparação dos resultados para instâncias com matriz de fluxos assimétricas e folgas explícitas

Instância	*n*	*v*[(a)]*	Dif_{melhor}	$Dif_{média}$	*Desv*	*DP*	$T_{médio}$
Murray155	15	**105578,4**	0,5	3,5	0,003	2,5	8,0
Murray206	20	**311751,1**	0,8	7,5	0,002	13,1	25,7
Murray218	25	**624653,3**	8,1	352,6	0,056	299,4	65,7
Murray231	30	**918759,6**	799,7	1387,2	0,151	543,9	145,6
Murray243	35	**1257631,8**	566,1	2415,3	0,192	869,7	316,4
Murray256	40	**2118158,1**	2121,2	4199,7	0,198	1287,8	461,1
Murray309	45	**2628585,5**	127,0	3195	0,122	1993,3	432,5
Murray322	50	**4018716,2**	4738,1	7314	0,182	1636,4	251,6
Murray324	50	4786649,1	**-2740,6**	8262,2	0,173	4351,8	254,0
Murray325	50	**4006421,4**	5470,1	8973,3	0,224	2713,7	252,8
Murray327	50	4237417,9	**-4610,4**	3296,5	0,078	3325,9	255,8
Murray328	50	**5273616,5**	1655	5370,7	0,102	2564,0	253,2
Média		2523994,9	678	3731,5	0,124	1633,5	226,9

[(a)] O melhor valor conhecido para $15 \leq n \leq 45$ estão disponíveis em Guan et al. (2020) e para as instâncias com $n = 50$ em Amaral et al. (2020).
Os valores mínimos estão destacados em negrito.

No geral, para esse conjunto de instâncias, o PSO-DRLP obteve soluções muito próximas dos melhores valores encontrados na literatura, como pode ser visto na coluna *Desv*, sendo 0,224% o maior desvio relativo para a instância Murray325, em comparação com a melhor solução obtida em Amaral (2020). Evidencia-se que o PSO-DRLP melhorou os melhores valores da literatura para duas instâncias com $n = 50$, Murray324 e Murray327.

5.6 CONSIDERAÇÕES FINAIS

Este capítulo apresentou uma abordagem com estratégias heurísticas para solucionar o problema de layout em duas linhas (DRLP), baseada na meta-heurística PSO. O PSO-DRLP

proposto é constituído de duas fases e com o enxame de partículas organizadas em subpopulações. O algoritmo proposto foi avaliado utilizando diversas instâncias do DRLP. Os testes realizados variam de instâncias com n = {11, 12, 13, 15, 16, 17, 18} a instâncias consideradas de grande porte com até n = 80 máquinas.

Os resultados obtidos mostraram que o PSO-DRLP possui uma convergência consistente para soluções de alta qualidade com baixos valores dos desvios relativos e desvio-padrão, mesmo com o aumento dos tamanhos das instâncias. No geral, a média das melhores soluções encontradas pelo algoritmo proposto em relação aos melhores valores de soluções encontradas na literatura possui um desvio de apenas 0,02%, sendo que o PSO-DRLP melhorou a melhor solução em 6 instâncias, alcançou os mesmos valores para 29 instâncias e para as 16 instâncias restantes, que o PSO-DRLP não obteve sucesso, os valores encontrados estão bem próximos dos melhores valores conhecidos, sendo o maior desvio percentual da média de 0,224% para a instância Murray325. De modo geral, pelos resultados obtidos, o PSO-DRLP pode ser considerado uma boa estratégia para a resolução do DRLP, sendo capaz de obter soluções de alta qualidade em tempos computacionais relativamente baixos.

O estudo realizado e descrito neste capítulo foi publicado e está disponível em Cravo, Bissoli e Amaral (2021).

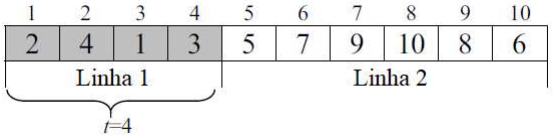
6 ALGORITMO ILS PROPOSTO PARA O PROP

Este capítulo apresenta com detalhes a proposta de solução para o PROP. O algoritmo proposto, denominando AILS, diferentemente do ILS padrão, executa iterações alternando de forma adaptativa entre duas fases usando diferentes critérios de aceitação. Além disso, o AILS ajusta a intensidade da perturbação em cada fase.

6.1 REPRESENTAÇÃO DA SOLUÇÃO E GERAÇÃO DA SOLUÇÃO INICIAL

Uma solução para o PROP é representada por um arranjo e um número inteiro *t*, que define o ponto de quebra do arranjo como em Amaral (2013a). Assim, em uma linha existem *t* facilidades e na outra linha as (*n*-*t*) facilidades restantes. A Figura 18 exemplifica a representação de uma solução PROP com $n = 10$ facilidades e $t = 4$.

Figura 18 – Representação de uma solução do PROP com $n = 10$ facilidades e $t = 4$



Fonte: próprio autor.

O Algoritmo 11 gera uma solução inicial viável com as facilidades em cada linha do layout distribuídas aleatoriamente.

Algoritmo 11 - Gerar Solução Inicial PROP – AILS

entrada: n, t
saída: S
1: $L_1 \leftarrow \{1, ..., t\}$, $L_2 \leftarrow \{t+1, ..., n\}$
2: $S_1 \leftarrow \emptyset$; $S_2 \leftarrow \emptyset$
3: **Enquanto** $L_1 \neq \emptyset$ **faça**
4: $i \leftarrow ObterFacilidadeAleatoriamente(L_1)$
5: $S_1 \leftarrow S_1 + \{i\}$ */*Adiciona i à direita das facilidades em S_1 */*
6: **Fim**
7: **Enquanto** $L_2 \neq \emptyset$ **faça**
8: $i \leftarrow ObterFacilidadeAleatoriamente(L_2)$
9: $S_2 \leftarrow S_2 + \{i\}$ */*Adiciona i à direita das facilidades em S_2 */*
10: **Fim**
11: $S \leftarrow (S_1, S_2)$
12: **retorne** S

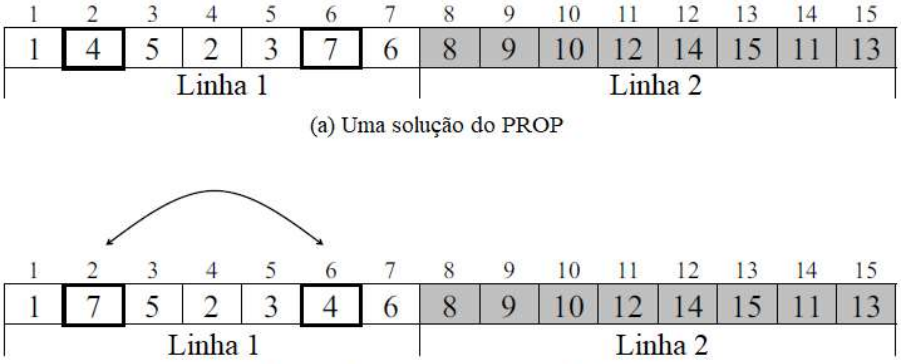
Na Linha 1, as listas L_1 e L_2 são inicializadas com as facilidades de 1, …, t e (t+1), ..., n, respectivamente. Na Linha 2, os arranjos da linha 1 e da linha 2 do layout são inicializados

vazios. No laço das Linhas 3-6 as facilidades são selecionadas de forma aleatória da L_1, uma a uma, e adicionadas à direita da sequência na linha 1 do layout (S_1). O mesmo procedimento é executado, no laço das Linhas 7-10, pegando as facilidades L_2 e adicionando à sequência da linha 2 (S_2). O algoritmo devolve uma solução inicial $S = (S_1, S_2)$.

6.2 PROCEDIMENTO DE BUSCA LOCAL

A busca local usada no algoritmo AILS proposto aplica iterativamente um movimento 2-opt a uma solução para gerar uma solução vizinha. O movimento 2-opt é amplamente utilizado em procedimentos heurísticos propostos para vários problemas de layout (AMARAL, 2020; CRAVO; AMARAL, 2019; GUAN; LIN 2016; GUAN et al., 2020; KOTHARI; GHOSH, 2013b). Um movimento 2-opt (p, q) troca as facilidades das posições p e q de um arranjo, como mostrado na Figura 19.

Figura 19 – Exemplo de aplicação do movimento de troca 2-opt no PROP



(a) Uma solução do PROP

(b) A solução em (a) após uma troca 2-opt

Fonte: próprio autor.

A Figura 19 exemplifica a aplicação do movimento 2-opt em um arranjo do PROP. A Figura 19(a) apresenta uma solução do PROP e a Figura 19(b) a solução após as facilidades das posições $p = 2$ e $q = 6$ serem trocadas. Observe que nesses exemplos, $p, q \in N_1$. Assim, os movimentos não mudam a disposição das facilidades na linha 2 e não será necessário avaliar a segunda soma em Equação (2.6).

Uma característica importante do PROP é que cada instalação é fixada em sua linha pré-alocada. Assim, a busca local não realiza movimentos envolvendo instalações de linhas diferentes, ou seja, a geração de movimentos (p, q) considera cada linha individualmente.

Como pode ser visto na Equação (2.6), as duas primeiras somas representam subproblemas SRFLP e a terceira soma representa o custo dos pares das facilidades (i, j) com i pertencendo à linha 1 e j pertencendo à linha 2. Assim, observa-se que:

- Para qualquer movimento (p, q) com p, $q \in N_1$, não há mudança no arranjo da linha 2. Assim, o segundo somatório não precisa ser avaliado e o ganho na função objetivo será calculado como o ganho do subproblema SRFLP representado pela linha 1 mais o custo definido pela terceira soma na Equação (2.6); e
- Para qualquer movimento (p, q) com p, $q \in N_2$, não há mudança no arranjo da linha 1. Assim, o primeiro somatório não precisa ser avaliado e o ganho na função objetivo será calculado como o ganho do subproblema SRFLP representado pela linha 2 mais o custo definido pela terceira soma na Equação (2.6).

Além do descrito acima, o cálculo do ganho na função objetivo no movimento 2-opt é realizado de modo que a avaliação do custo total do arranjo em cada linha não seja necessária, ou seja, o cálculo do ganho para o subproblema SRFLP em uma linha usa as mesmas técnicas para SRFLP de Kothari e Ghosh (2013b), pelo qual o cálculo realizado para determinar o ganho na função objetivo para o movimento (p, q) é empregado para calcular o ganho para o movimento (p, q+1), assim, reduzindo a complexidade dos cálculos de $O(n^2)$ a $O(n)$.

O procedimento de busca local consiste em aplicar movimentos 2-opt à solução atual até que não seja possível melhorar o valor da função objetivo, ou seja, até que um mínimo local seja alcançado. A busca local varre todos os pares (p, q) disponíveis em uma lista L_{movs}. A lista contém os possíveis pares para os movimentos 2-opt para a linha 1 (p = 1, …, t-1; $q = p + 1$, …, t) e para a linha 2 ($p = t + 1$, …, n -1; $q = p + 1$, ..., n).

O Algoritmo 12 mostra o pseudocódigo da busca local proposta. O algoritmo recebe como entrada os seguintes parâmetros: S_0, uma solução inicial; t, número de facilidades na primeira linha; e n, tamanho do problema.

Na Linha 1, a solução atual S é inicializada. Na Linha 2, a lista L_{movs} é inicializada com todos os movimentos 2-opt válidos. Nas Linhas 3 a 4, as variáveis de controle são inicializadas, sendo *cont* o contador do número de movimentos que foram aplicados à solução atual S e i, o índice de um movimento na lista L_{movs}. O laço das Linhas 5-21 é repetido enquanto houver melhorias na função objetivo. No laço das Linhas 7-20, cada movimento em L_{movs} é aplicado à S em sequência. Para cada movimento 2-opt, obtido na Linha 8, uma solução vizinha S' é gerada, aplicando o movimento à S (Linha 9). Na Linha 10, é verificado se a nova solução S' é melhor

que a solução atual *S*. Se verdadeiro, a solução *S* é atualizada na Linha 11, a *flag melhorou* é definida como um (Linha 12) e *cont* é definida como zero (Linha 13). Na Linha 15, *cont* é incrementada e, nas Linhas 16 a 19, o índice *i* é incrementado ciclicamente (ou seja, *i* é definido como um, se *i* for maior do que $|L_{movs}|$). O algoritmo termina quando nenhuma solução vizinha melhor é encontrada e então devolve a solução ótima local *S*.

Algoritmo 12 – Busca Local – AILS

```
entrada: S0, t, n
saída: S
 1:  S ← S0
 2: Lmovs ← {(1, 2), (1, 3), ..., (t-1, t), (t+1, t+2), ..., (n-1, n)}
 3: cont ← 1
 4: i ← 1
 5: Faça
 6:    melhorou ← 0
 7:    Enquanto cont ≤ | Lmovs | faça
 8:       (p, q) ← Lmovs[i]
 9:       S' ← 2OptMov (S, p, q)
10:       Se f(S') < f(S) então
11:          S ← S'
12:          melhorou ← 1
13:          cont ← 0
14:       Fim
15:       cont ← cont + 1
16:       i ← i + 1
17:       Se i > | Lmovs | então
18:          i ← 1
19:       Fim
20:    Fim
21: Enquanto melhorou = 1
22: retorne S
```

6.3 PROCEDIMENTO DE PERTURBAÇÃO

O procedimento de perturbação no AILS gera uma nova solução $\hat{S}$ mudando a posição das facilidades em cada linha na solução corrente *S*. Assim, dadas as listas $L_1 = (1, ..., t)$ e $L_2 = (t+1, \ldots, n)$ para $r = \{1, 2\}$, um par (p, q) é selecionado aleatoriamente e removido da lista L_r, e, então, uma das duas sub-rotinas é aplicada ao arranjo π^r: (*Rotina* 1) a instalação na posição *p* é removida e inserida na posição *q,* e, (*Rotina* 2) as posições das instalações no intervalo (p, q) são invertidas. Como um exemplo, suponha o layout da Figura 18 com arranjos $\pi^1 = (2, 4, 1, 3)$ e $\pi^2 = (5, 7, 9, 10, 8, 6)$. Inicialmente, $L_1 = (1, 2, 3, 4)$ e $L_2 = (5, 6,7, 8, 9, 10)$. Supondo que para π^1, o par $(p = 4; q = 1)$ de L_1 é selecionado, então, a lista L_1 é atualizada para $L_1 = (2, 3)$. Se *Rotina* 1 for aplicada, o arranjo resultante π^1 será (3, 2, 4, 1). Por outro lado, se a *Rotina* 2 for

aplicada, o arranjo resultante π^1 será (3, 1, 4, 2). A escolha de qual rotina usar é baseada em um parâmetro de probabilidade P. O número de vezes que uma rotina é aplicada ao arranjo π^r depende do parâmetro I_r (r = {1, 2}).

O Algoritmo 13 mostra o pseudocódigo do procedimento de perturbação proposto. Ele recebe como parâmetros: S, a solução a ser perturbada; I_1 e I_2, intensidades de perturbação para cada linha; e P, parâmetro de probabilidade, usado para escolher uma rotina de perturbação. O algoritmo retorna uma solução perturbada $\hat{S}$.

Algoritmo 13 – Procedimento de Perturbação – AILS

```
entrada: S, I1, I2, P
saída: Ŝ
 1: Ŝ ← S
 2: L1 ← {1, ..., t}, L2 ← {t+1, ..., n}
 3: Enquanto I1 > 1 faça
 4:    (p, q) ← Obter_Aleatoriamente(L1)
 5:    Se rand() < P então
 6:       Rotina1 (Ŝ, p, q)
 7:    Senão
 8:       Rotina2 (Ŝ, p, q)
 9:    Fim
10:    I1 ← I1 - 2
11: Fim
12: Enquanto I2 > 1 faça
13:    (p, q) ← Obter_Aleatoriamente(L2)
14:    Se rand() < P então
15:       Rotina1 (Ŝ, p, q)
16:    Senão
17:       Rotina2 (Ŝ, p, q)
18:    Fim
19:    I2 ← I2 - 2
20: Fim
21: retorne Ŝ
```

Na linha 1, $\hat{S}$ é definido como S; na linha 2, as listas L_1 e L_2 são inicializadas; no laço das Linhas 3-11, os movimentos são aplicados ao arranjo na linha 1 da solução $\hat{S}$. Assim, na Linha 4, um par (p, q) é obtido de L_1 em relação ao arranjo da linha 1 da solução $\hat{S}$; na Linha 5, a rotina de perturbação é selecionada aleatoriamente, a função *rand*() gera um valor aleatório no intervalo [0, 1). O tipo de perturbação é feito de acordo com a sub-rotina escolhida, sendo a *Rotina*1 com probabilidade P e a *Rotina*2 com probabilidade 1-P. Observe que $P = 1$ indica que apenas a *Rotina*1 é usada, $P = 0$ indica que apenas a *Rotina*2 é executada e, se $0 < P < 1$, então movimentos de ambos os tipos podem ser empregados. Dependendo da escolha, é executada a inserção da facilidade da posição p na posição q ou a inversão das facilidades no intervalo de posições (p, q). Como em cada iteração um par (p, q) é removido de L_1, o valor de I_1 é decrementado em duas unidades, conforme mostrado na Linha 10.

No laço das Linhas 12-20, os movimentos são aplicados ao arranjo da linha 2 da solução $\hat{S}$, de forma semelhante ao que foi descrito para a linha 1 (Linhas 3 a 11). Em seguida, $\hat{S}$ é retornado na Linha 21. Os valores I_1 e I_2 recebidos como uma entrada pelo procedimento de perturbação são calculados usando a Equação (6.1).

$$I_r = \lfloor \alpha \times |N_r| \rfloor \tag{6.1}$$

Onde α define o percentual do número de facilidades de $|N_r|$ no arranjo da linha r do layout.

O valor da intensidade da perturbação (I_r) varia durante o processo de busca de acordo com o valor de α. Dado os limites α_{min} e α_{max}, inicialmente $\alpha = \alpha_{min}$ e durante o processo iterativo do AILS, no momento em que o k-ésimo ótimo local (S^k) é encontrado, α é ajustado pela multiplicação do seu valor corrente α' pela razão entre o custo de S^k e o custo da melhor entre as (k-1) soluções ótimas locais anteriores, ou seja, $min\ \{f(S^1), \ldots, f(S^{k-1})\}$, como expressado na Equação (6.2).

$$\alpha = \alpha' \times \frac{f(S^k)}{min\{f(S^1), \ldots, f(S^{k-1})\}} \tag{6.2}$$

Após a atualização do valor de α, se o valor sair do intervalo $[\alpha_{min}, \alpha_{max}]$, o valor é reinicializado com $\alpha = \alpha_{min}$. Na Equação (6.2), f é a função de custo do PROP (Equação 2.6). O ajuste aumentará o valor de α se a solução S^k for pior (ou diminuirá o valor de α se a solução S^k for melhor) do que $min\{f(S^1), \ldots, f(S^{k-1})\}$. A ideia é aumentar a intensidade da perturbação a ser realizada na próxima iteração do AILS, quando uma solução melhor não for gerada, ou diminuir a intensidade da perturbação, caso contrário.

Note que se uma perturbação muito grande parecer necessária, ela seria alcançada realizando várias pequenas perturbações usando $\alpha = \alpha_{min}$. Isso está de acordo com o que foi sugerido em Lourenço, Martin e Stützle (2003).

6.4 CRITÉRIO DE ACEITAÇÃO DO AILS

O critério de aceitação determina qual solução se tornará a próxima solução atual, e, portanto, será perturbada. O critério de aceitação influencia muito o nível de intensificação / diversificação da pesquisa.

No algoritmo AILS proposto existem duas fases e cada fase usa um critério de aceitação diferente. Na Fase 1, para intensificar a busca, a solução S a ser perturbada a seguir é a melhor solução encontrada até o momento. Assim, no momento em que a k -ésima solução ótima local S^k é encontrada, o critério de aceitação é dado pela Equação (6.3).

$$S = min\{f(S^1), \ldots, f(S^k)\} \tag{6.3}$$

Onde f é a função custo do PROP definida pela Equação (2.6).

Na Fase 2 do AILS, a fim de diversificar a busca, o critério de aceitação utilizado escolhe o último ótimo local encontrado, embora possa não ser a melhor solução visitada até o momento. Assim, no momento em que a k-ésima solução ótima local S^k é encontrada, este critério de aceitação é dado pela Equação (6.4).

$$S = S^k \tag{6.4}$$

6.5 ALGORITMO AILS PROPOSTO

Ao definir o procedimento para construir uma solução inicial, a busca local, o mecanismo de perturbação e os critérios de aceitação para as duas fases, o algoritmo AILS proposto é determinado. O pseudocódigo pode ser visto no Algoritmo 14.

O AILS recebe como entrada os seguintes parâmetros: n, tamanho do problema; t, número de instalações na primeira linha; T_{Lim}, limite de tempo de execução; *IterWI*, o número máximo de iterações sem melhoria da solução na Fase 1; $ITER_{max}$, número máximo de iterações na Fase 2; α_{min} e α_{max}, os valores percentuais mínimo e máximo, respectivamente, para o cálculo da intensidade I_r (r = 1, 2) da perturbação (Equação 6.1); e P, parâmetro de probabilidade.

Na Linha 1, a solução inicial S^0 é criada; na Linha 2, k é definido como 1 e, na Linha 3, a busca local (Algoritmo 12) é aplicada à S^0 para obter a solução S^k. Na Linha 4, a melhor solução encontrada até agora, S^*, é definida com S^k; na Linha 5, são inicializados os valores de α_1 e α_2, usados para determinar a intensidade da perturbação nas Fases 1 e 2 do AILS, respectivamente. O laço nas Linhas 6-39 define o processo iterativo do AILS. Na Linha 7, a variável *iter_melhor* começa com o valor da iteração atual k.

Algoritmo 14 –Algoritmo AILS proposto – PROP

entrada: n, t, T_{Lim}, $IterWI$, $ITER_{max}$, α_{min}, α_{max}, P
saída: S^*
1: $S^0 \leftarrow GerarSoluçãoInicialPROP\ (n,\ t)$
2: $k \leftarrow 1$
3: $S^k \leftarrow BuscaLocal\ (S^0, t, n)$
4: $S^* \leftarrow S^k$ /* Inicializa a melhor solução encontrada */
5: $\alpha_1 \leftarrow \alpha_{min}$, $\alpha_2 \leftarrow \alpha_{min}$
6: **Faça**
/* Fase 1 */
7: $iter\_melhor \leftarrow k$
8: **Faça**
9: $A \leftarrow S^*$ /* Critério de aceitação intensificação */
10: $k \leftarrow k + 1$
11: $I_1 \leftarrow \lfloor \alpha_1 \times (t) \rfloor$, $I_2 \leftarrow \lfloor \alpha_1 \times (n\text{-}t) \rfloor$
12: $\hat{S} \leftarrow Perturbação\ (A, I_1, I_2, P)$
13: $S^k \leftarrow BuscaLocal\ (\hat{S}, t, n)$
14: $\alpha_1 \leftarrow \alpha_1 \times \frac{f(S^k)}{f(S^*)}$
15: **Se** $\alpha_1 < \alpha_{min}$ **OU** $\alpha_1 > \alpha_{max}$ **então**
16: $\alpha_1 \leftarrow \alpha_{min}$
17: **Fim**
18: **Se** $f(S^k) < f(S^*)$ **então**
19: $S^* \leftarrow S^k$ /* Atualiza a melhor solução */
20: $iter\_melhor \leftarrow k$
21: **Fim**
22: **Enquanto** k - $Iter\_melhor \neq IterWI$ **E** $tempo\_decorrido \leq T_{Lim}$
/* Fase 2 */
23: $iter\_diver \leftarrow 0$
24: **Faça**
25: $A \leftarrow S^k$ /* Critério de aceitação diversificação */
26: $iter\_diver \leftarrow iter\_diver + 1$
27: $k \leftarrow k + 1$
28: $I_1 \leftarrow \lfloor \alpha_1 \times (t) \rfloor$, $I_2 \leftarrow \lfloor \alpha_1 \times (n\text{-}t) \rfloor$
29: $\hat{S} \leftarrow Perturbação\ (A, I_1, I_2, P)$
30: $S^k \leftarrow BuscaLocal\ (\hat{S}, t, n)$
31: $\alpha_2 \leftarrow \alpha_2 \times \frac{f(S^k)}{f(S^*)}$
32: **Se** $\alpha_2 < \alpha_{min}$ **OU** $\alpha_2 > \alpha_{max}$ **então**
33: $\alpha_2 \leftarrow \alpha_{min}$
34: **Fim**
35: **Se** $f(S^k) < f(S^*)$ **então**
36: $S^* \leftarrow S^k$ /* Atualiza a melhor solução */
37: **Fim**
38: **Enquanto** $iter\_diver \neq ITERmax$ **OU** $tempo\_decorrido \leq T_{Lim}$
39: **Enquanto** $tempo\_decorrido \leq T_{Lim}$
40: **retorne** S^*

O laço nas Linhas 8-22 define a Fase 1 do AILS. Na Linha 9, um critério de aceitação que reforça a intensificação é utilizado (Equação 6.3), e a solução aceita é denotada por A; na Linha 10, k é incrementado; na Linha 11, as intensidades I_1 e I_2 são calculadas para o procedimento de perturbação; na Linha 12, a perturbação é aplicada à solução A, com intensidade I_1 e I_2 para os arranjos da linha 1 e linha 2 do layout, respectivamente, retornando uma solução perturbada $\hat{S}$, que é submetida à busca local na Linha 13. A busca local retorna uma nova solução S^k; na

Linha 14, o valor de α_1 é atualizado de acordo com a Equação (6.2); e nas Linhas 15 a 17, se o valor de α_1 estiver fora do intervalo [α_{min}, α_{max}], α_1 é redefinido para α_{min}.

Na Linha 18, é verificado se a nova solução S^k é melhor que a melhor solução atual S^*. Se verdadeiro, então S^* é atualizado (Linha 19) e *iter_melhor* é definido com o valor de k (Linha 20). A Fase 1 termina quando as iterações do *IterWI* tiverem decorrido, sem que uma solução melhor tenha sido encontrada ou o tempo de execução tenha ultrapassado T_{Lim} (Linha 22).

A Fase 2 do algoritmo começa na Linha 23 com a inicialização da variável *iter_diver*, que controla o número de iterações na Fase 2; na Linha 25 é utilizado um critério de aceitação que incentiva a diversificação (Equação 6.4) e, assim, a solução aceita A é determinada; nas Linhas 26 e 27, as variáveis *iter_diver* e k são incrementadas, respectivamente; na Linha 28, as intensidades I_1 e I_2 para o procedimento de perturbação são definidas; na Linha 29, a perturbação é aplicada à solução A, com intensidade I_1 e I_2 para os arranjos da linha 1 e linha 2 do layout, respectivamente, retornando uma solução perturbada $\hat{S}$; na Linha 30, a busca local é aplicada à solução $\hat{S}$, retornando a nova solução S^k; nas Linhas 31 a 34, o valor de α_2 é atualizado e seus limites são verificados, como na Fase 1 do AILS. O custo da nova solução S^k é comparado ao melhor custo obtido até então (Linha 35). Se for melhor, a solução S^* é atualizada (Linha 36). A Fase 2 termina quando seu número máximo de iterações *ITERmax* é atingido ou o tempo de execução excede T_{Lim}. No final, AILS termina quando o tempo máximo de execução T_{Lim} é atingido (Linha 39), retornando a melhor solução encontrada S^* na Linha 40.

6.6 EXPERIMENTOS COMPUTACIONAIS

Os experimentos computacionais foram realizados em um computador desktop com processador Intel Core i7-7700k com 4,2 GHz e 16GB de RAM e os algoritmos propostos foram implementados na linguagem C e compilados usando GCC 5.1.0 x64 com as *flags* de otimização -O3 e -march.

Para avaliar o efeito do procedimento de perturbação, foram consideradas três variantes do AILS, que diferem no valor do parâmetro de probabilidade P: AILS0 tem $P = 0,5$; AILS1 tem $P = 1$; e AILS2 tem $P = 0$.

6.6.1 Calibração dos parâmetros

A escolha dos parâmetros do algoritmo foi realizada usando a meta-heurística PSO descrita no Apêndice B. No processo iterativo do PSO, soluções (partículas) representadas por um vetor são geradas. Cada componente desse vetor é um parâmetro do algoritmo que está sendo calibrado. O PSO executa o algoritmo a ser calibrado como uma sub-rotina. O PSO informa um conjunto de parâmetros para a sub-rotina e obtém a saída da sub-rotina, que é a aptidão da partícula junto com alguma estatística de execução. O valor de aptidão é usado para determinar a melhor partícula no processo iterativo do PSO. No final, a partícula contendo a melhor aptidão é retornada contendo seus valores de parâmetros associados (CRAVO; AMARAL, 2019).

Na calibração, um conjunto de dados, contendo cinco instâncias com n = 100 facilidades, denominado sko100 (ANJOS; YEN, 2009) foi utilizado. Os resultados do PSO são usados para escolher os parâmetros dos algoritmos propostos. Os parâmetros obtidos são mostrados na Tabela 20. O valor de t foi definido como $t = \frac{n}{2}$ na calibração.

Tabela 20 – Parâmetros encontrados para as variantes do AILS

Algoritmo	*IterWI*	*ITERmax*	α_{min}	α_{max}
AILS0	60	71	0,1822	0,4236
AILS1	164	71	0,1342	0,1845
AILS2	38	24	0,0409	0,6097

6.6.2 Critério de avaliação do desempenho

Para avaliar a eficiência do algoritmo proposto, cada variante do AILS foi executada 20 vezes para cada instância usada nos experimentos computacionais, assim como em Maadi, Javidnia e Jamshidi (2017). O melhor valor da solução encontrada é denotado por *Melhor,* o valor médio da solução é *Média* e *DP* indica o desvio-padrão do valor da solução. Dada a melhor solução conhecida (v*) para uma determinada instância, são calculados:

- D_{melhor}, a diferença da melhor solução encontrada em relação ao valor da melhor solução conhecida, $D_{melhor} = Melhor - v$*; e
- $D_{média}$, a diferença do valor médio das soluções para o valor da melhor solução conhecida: $D_{média} = Média - v$*;

6.6.3 Avaliação das três variantes do AILS para instâncias com tamanhos $30 \leq n \leq 70$

Nesta subseção, as variantes do algoritmo AILS são comparados com o algoritmo PSA de Maadi, Javidnia e Jamshidi (2017). Para isso, serão considerados cinco conjuntos de instâncias originalmente propostos para o SRFLP e usados por Maadi, Javidnia e Jamshidi (2017), relacionados a seguir:

- N30, conjunto com cinco instâncias de tamanhos n = 30 facilidades (ANJOS; VANNELLI, 2008);
- N40, conjunto com cinco instâncias com n = 40 facilidades (HUNGERLÄNDER; RENDL, 2013);
- sko56, conjunto com cinco instâncias com tamanhos n = 56 facilidades (ANJOS; YEN, 2009);
- AKV_60, conjunto com cinco instâncias com n = 60 facilidades (ANJOS; KENNINGS; VANNELLI, 2005); e
- AKV_70, com cinco instâncias com tamanhos n = 70 facilidades (ANJOS; KENNINGS; VANNELLI, 2005).

Para cada instância dos cinco conjuntos, os algoritmos foram executados com cada $t \in \{\frac{n}{2}, \frac{n}{3}, \frac{n}{4}, \frac{n}{5}\}$, resultando no total de 100 instâncias do PROP.

Os limites dos tempos de execução das versões do algoritmo AILS foram definidos de acordo com os tempos de execução usados em Maadi, Javidnia e Jamshidi (2017). Para que a comparação entre computadores com diferentes processadores seja mais justa, os limites de tempo de execução dos algoritmos propostos foram ajustados usando as informações de desempenho dos processadores (CPUBENCHMARK, 2021). O processador usado em Maadi, Javidnia e Jamshidi (2017) tem uma classificação de single thread de 1539 e o processador usado nos testes, 2775. Assim, o limite de tempo T_{Lim} usado pelos algoritmos AILS para uma instância com característica (n, t) é a média dos tempos usados em Maadi, Javidnia e Jamshidi (2017) para o grupo de instâncias com essa característica (n, t), ajustado pelo fator $(\frac{1539}{2775})$ = 0,555 e o resultado, arredondado para um valor inteiro. Os limites de tempo de execução,

ajustados para cada grupo de cinco instâncias com tamanhos $n \in \{30, 40, 56, 60, 70\}$ e $t \in \{\frac{n}{2}, \frac{n}{3}, \frac{n}{4}, \frac{n}{5}\}$, são mostrados na Tabela 21.

Tabela 21 – Limites de tempo (T_{Lim}), em segundos, usados nos algoritmos AILS

Conjunto	Número de Instâncias	n	$\frac{n}{2}$	$\frac{n}{3}$	$\frac{n}{4}$	$\frac{n}{5}$
N30	5	30	2	2	2	2
N40	5	40	3	4	4	4
sko56	5	56	8	10	11	17
AKV_60	5	60	15	17	21	25
AKV_70	5	70	23	28	30	40

Na Tabela 21, por exemplo, para cada uma das 5 instâncias com $n = 70$ e $t = \frac{n}{3}$ o valor de $T_{Lim} =$ 28 segundos.

Os resultados para as instâncias com $30 \leq n \leq 70$ e $t = \frac{n}{2}$ são apresentados na Tabela 22. A primeira coluna da tabela mostra os nomes das instâncias; a segunda coluna, os valores das melhores soluções obtidas pelo algoritmo PSA (MAADI; JAVIDNIA; JAMSHIDI, 2017) e as diferenças dos valores médio das soluções e as melhores soluções conhecidas ($D_{média}$) do PSA, entre parênteses. Para cada variante do AILS a tabela mostra $D_{melhor}(D_{média})$ seguido do desvio-padrão, *DP*. A última linha da tabela mostra os resultados das médias de todas as instâncias da tabela. Os melhoramentos identificados são mostrados em negrito.

Como pode ser visto na Tabela 22, todas as três variantes do AILS melhoraram os melhores valores conhecidos para doze das vinte e cinco instancias: uma instância com $n = 40$ (N40_02), todas as cinco instâncias com $n = 56$, três instâncias com $n = 60$ (AKV_60_02, AKV_60_03 e AKV_60_05) e três instâncias com $n = 70$ (AKV_70_01, AKV_70_02 e AKV_70_04). Para as treze instâncias restantes, o melhor valor conhecido da literatura foi encontrado.

Considerando os valores médios das soluções, os algoritmos AILS apresentaram resultados melhores que o algoritmo PSA, com exceção do AILS1, que obteve piores valores para AKV_70_02 e AKV_70_05 em relação aos valores obtidos pelo algoritmo PSA. Observa-se também, a partir dos pequenos valores dos desvios-padrão, que os algoritmos AILS são robustos.

Tabela 22 – Resultados encontrados pelos algoritmos AILS0, AILS1 e AILS2 para instâncias com $30 \leq n \leq 70$ facilidades e $t = n/2$

Instância	PSA	AILS0		AILS1		AILS2	
	$v^{*(a)}(D_{média})$	$D_{melhor}(D_{média})$	*DP*	$D_{melhor}(D_{média})$	*DP*	$D_{melhor}(D_{média})$	*DP*
N30_01	**4174**,0(1,96)	**0**(0)	0	**0**(0)	0	**0**(0)	0
N30_02	**11154**,5(6,47)	**0**(0)	0	**0**(0,85)	3,8	**0**(0)	0
N30_03	**23127**,0(16,19)	**0**(0)	0	**0**(0)	0	**0**(0)	0
N30_04	**32651**,5(29,06)	**0**(0)	0	**0**(0)	0	**0**(0)	0
N30_05	**60353**,0(25,35)	**0**(0)	0	**0**(0)	0	**0**(0)	0
N40_01	**55526**,5(217,11)	**0**(0)	0	**0**(11,00)	11,3	**0**(0)	0
N40_02	50399,0(146,66)	**-1,0**(-1,00)	0	**-1,0**(48,60)	62,3	**-1,0**(-1,00)	0
N40_03	**42118**,5(160,89)	**0**(0)	0	**0**(0)	0	**0**(0)	0
N40_04	**40998**,0(209,91)	**0**(0)	0	**0**(0)	0	**0**(0)	0
N40_05	**52562**,0(211,82)	**0**(0)	0	**0**(64,75)	90,5	**0**(0)	0
sko56_01	32292,0(57,16)	**-106,0**(-105,90)	0,3	**-106,0**(-104,45)	6,9	**-106,0**(-105,90)	0,3
sko56_02	259500,0(677,30)	**-151,0**(-151,00)	0	**-151,0**(186,00)	691,5	**-151,0**(-151,00)	0
sko56_03	85881,0(182,07)	**-168,0**(-157,70)	31,7	**-168,0**(-68,90)	107,4	**-168,0**(-162,75)	23,5
sko56_04	158939,0(311,52)	**-253,0**(-183,75)	49,1	**-253,0**(-191,25)	104,2	**-253,0**(-173,45)	48,7
sko56_05	299429,5(601,85)	**-616,0**(-616,00)	0	-616,0(-338,80)	232,2	**-616,0**(-616,00)	0
AKV_60_01	**772202**,0(625,48)	**0**(0)	0	**0**(243,75)	183,5	**0**(18,75)	83,9
AKV_60_02	430384,0(404,56)	**-4,0**(-4,00)	0	**-4,0**(130,55)	152,6	**-4,0**(-4,00)	0
AKV_60_03	331140,5(248,36)	**-37,0**(15,80)	129	**-37,0**(33,40)	144,5	**-37,0**(30,35)	201
AKV_60_04	**201052**,0(180,95)	**0**(75,95)	80,2	**0**(109,55)	61,9	**0**(97,80)	97
AKV_60_05	165099,0(143,64)	**-3,0**(-3,00)	0	**-3,0**(4,80)	24,0	**-3,0**(-3,00)	0
AKV_70_01	779563,0(709,40)	**-433,0**(-433,00)	0	**-433,0**(-433,00)	0	**-433,0**(-254,80)	366
AKV_70_02	738304,0(730,92)	**-385,0**(-190,25)	486,0	**-385,0**(998,30)	473,1	**-385,0**(154,70)	617
AKV_70_03	**764463**,5(680,37)	**0**(594,30)	399,0	**0**(551,85)	415,5	**0**(470,00)	430
AKV_70_04	491217,0(481,39)	**-189,0**(-189,00)	0	**-189,0**(-139,95)	55,6	**-189,0**(-189,00)	0
AKV_70_05	**2187780**,5(1881,49)	**0**(0)	0	**0**(7637,00)	5130,0	**0**(0)	0
Média	322812,44(545,17)	-93,8(-53,94)	47	-93,8(349,76)	318,0	-93,8(-35,57)	74,7

[(a)] O melhor valor conhecido obtido por Maadi et al. (2017).
Os melhores valores são destacados em negrito.

A Tabela 23 apresenta a comparação dos resultados encontrados com as melhores soluções conhecidas para as instâncias com $30 \leq n \leq 70$ e $t = \frac{n}{3}$. A tabela está estruturada conforme Tabela 22.

Para as instâncias da Tabela 23, os algoritmos propostos melhoraram os melhores valores conhecidos para treze instâncias de vinte e cinco instâncias: duas instâncias com n = 40 (N40_01 e N40_04); todas as cinco instâncias com n = 56; três instâncias com n = 60 (AKV_60_03, AKV_60_04 e AKV_60_05); e três instâncias com n = 70 (AKV_70_01 a AKV_70_03). Para as doze instâncias restantes, os melhores valores conhecidos foram obtidos.

Além disso, o algoritmo AILS0 obteve os valores melhores ou iguais aos melhores valores conhecidos em todas as rodadas (*DP* = 0) para vinte e três instâncias; e ambos, AILS1 e AILS2 para vinte instâncias.

Para esse conjunto de instâncias, considerando os valores médios das soluções, o AILS0 melhorou os valores médios das soluções do algoritmo PSA para todas as instâncias. O algoritmo AILS1 não obteve o valor médio melhor que o algoritmo PSA apenas para AKV_60_02; e o algoritmo AILS2 apenas para AKV_60_04.

Tabela 23 – Resultados encontrados pelos algoritmos AILS0, AILS1 e AILS2 para instâncias com $30 \leq n \leq 70$ facilidades e $t = n/3$

Instância	PSA	AILS0		AILS1		AILS2	
	$v^{*(a)}(D_{média})$	$D_{melhor}(D_{média})$	*DP*	$D_{melhor}(D_{média})$	*DP*	$D_{melhor}(D_{média})$	*DP*
N30_01	**5310**(1,79)	**0**(0)	0	**0**(0)	0	**0**(0)	0
N30_02	**14894**,5(6,14)	**0**(0)	0	**0**(0)	0	**0**(0)	0
N30_03	**27306**(13,68)	**0**(0)	0	**0**(0)	0	**0**(0)	0
N30_04	**44498**,5(18,73)	**0**(0)	0	**0**(0)	0	**0**(0)	0
N30_05	**68998**(25,05)	**0**(0)	0	**0**(0)	0	**0**(0)	0
N40_01	83103,5(41,63)	**-4,0**(-4,00)	0	**-4,0**(-4,00)	0	**-4,0**(-4,00)	0
N40_02	**63212**(25,54)	**0**(0)	0	**0**(0)	0	**0**(0)	0
N40_03	**46444**,5(23,13)	**0**(0)	0	**0**(0)	0	**0**(0)	0
N40_04	45142(23,61)	**-1,0**(-1,00)	0	**-1,0**(-1,00)	0	**-1,0**(-1,00)	0
N40_05	**58492**(24,04)	**0**(0)	0	**0**(16,50)	25,9	**0**(0)	0
sko56_01	42946(56,17)	**-44,0**(-44,00)	0	**-44,0**(-44,00)	0	**-44,0**(-44,00)	0
sko56_02	309277(598,76)	**-740,0**(-740,00)	0	**-740,0**(-25,80)	978,5	**-740,0**(-731,95)	36
sko56_03	107469(228,48)	**-29,0**(-29,00)	0	**-29,0**(-29,00)	0	**-29,0**(-29,00)	0
sko56_04	200421(224,67)	**-155,0**(-155,00)	0	**-155,0**(-155,00)	0	**-155,0**(-155,00)	0
sko56_05	374202,5(640,63)	**-208,0**(-206,80)	1,0	**-208,0**(-207,20)	1,0	**-208,0**(-207,10)	1,0
AKV_60_01	**996191**(100,62)	**0**(0)	0	**0**(0)	0	**0**(0)	0
AKV_60_02	**494862**(60,37)	**0**(0)	0	**0**(2654,25)	1572,2	**0**(0)	0
AKV_60_03	411379,5(47,31)	**-108,0**(-108,00)	0	**-108,0**(-108,00)	0	**-108,0**(-104,50)	15,7
AKV_60_04	267511(34,24)	**-8,0**(-7,70)	0,9	**-8,0**(-2,50)	24,6	**-8,0**(91,50)	97,5
AKV_60_05	187572(24,57)	**-38,0**(-38,00)	0	**-38,0**(-38,00)	0	**-38,0**(-38,00)	0
AKV_70_01	963209(111,73)	**-98,0**(-98,00)	0	**-98,0**(-98,00)	0	**-98,0**(-98,00)	0
AKV_70_02	961762(115,41)	**-16,0**(-16,00)	0	**-16,0**(-16,00)	0	**-16,0**(-16,00)	0
AKV_70_03	970083,5(162,97)	**-58,0**(-58,00)	0	**-58,0**(-58,00)	0	**-58,0**(-58,00)	0
AKV_70_04	**625518**(75,69)	**0**(0)	0	**0**(0)	0	**0**(0)	0
AKV_70_05	**2789182**,5(304,02)	**0**(0)	0	**0**(0)	0	**0**(193,25)	167,2
Média	406359,48(224,41)	-60,3(-60,22)	0,1	-60,3(75,37)	104,1	-60,3(-48,07)	12,7

(a) O melhor valor conhecido obtido por Maadi et al. (2017).
Os melhores valores são destacados em negrito.

Os resultados encontrados pelos algoritmos propostos, para instâncias com tamanhos $30 \leq n \leq 70$ e $t = \frac{n}{4}$, são apresentados e comparados com as melhores soluções conhecidas na Tabela 24. A tabela está estruturada conforme Tabela 22.

Como pode ser visto na Tabela 24, os algoritmos propostos melhoraram as soluções de referência para doze instâncias das vinte e cinco, sendo uma instância com $n = 40$ (N40_01), todas as cinco instâncias com $n = 56$, duas instâncias com $n = 60$ (AKV_60_03 e AKV_60_05) e quatro instâncias com $n = 70$ (AKV_70_01, AKV_70_02, AKV_70_03 e AKV_70_05). Para as treze restantes que os algoritmos não melhoraram, os melhores valores presentes na literatura foram alcançados. Note também que, para vinte e três instâncias, ambos, AILS0 e AILS1, obtiveram $DP = 0$ e o AILS2 obteve o mesmo desempenho para vinte e uma instâncias.

Para esse conjunto de testes, os valores médios das soluções encontradas pelo AILS0 e AILS2 foram melhores que os da literatura para todas as instâncias. O valor médio da solução encontrado pelo AILS1 foi pior do que na literatura apenas para a instância AKV_60_02.

Tabela 24 – Resultados encontrados pelos algoritmos AILS0, AILS1 e AILS2 para instâncias com $30 \leq n \leq 70$ facilidades e $t = n/4$

Instância	PSA	AILS0		AILS1		AILS2	
	$v^{*(a)}(D_{média})$	$D_{melhor}(D_{média})$	*DP*	$D_{melhor}(D_{média})$	*DP*	$D_{melhor}(D_{média})$	*DP*
N30_01	**6791**(16,00)	**0**(0)	0	**0**(0)	0	**0**(0)	0
N30_02	**18928**,5(49,95)	**0**(0)	0	**0**(0)	0	**0**(0)	0
N30_03	**34523**(108,82)	**0**(0)	0	**0**(0)	0	**0**(0)	0
N30_04	**52710**,5(109,06)	**0**(0)	0	**0**(0)	0	**0**(0)	0
N30_05	**89548**(260,76)	**0**(0)	0	**0**(0)	0	**0**(0)	0
N40_01	98512,5(40,78)	**-8,0**(-8,00)	0	**-8,0**(-8,00)	0	**-8,0**(-8,00)	0
N40_02	**73752**(25,59)	**0**(0)	0	**0**(0)	0	**0**(0)	0
N40_03	**65280**,5(24,09)	**0**(0)	0	**0**(0)	0	**0**(0)	0
N40_04	**63314**(32,23)	**0**(0)	0	**0**(0)	0	**0**(0)	0
N40_05	**74006**(28,27)	**0**(0)	0	**0**(0)	0	**0**(0)	0
sko56_01	49475(45,37)	**-32,0**(-29,00)	6,2	**-32,0**(-18,50)	12,6	**-32,0**(-31,80)	0,4
sko56_02	362393(337,39)	**-51,0**(-51,00)	0	**-51,0**(-51,00)	0	**-51,0**(-51,00)	0
sko56_03	128843(115,44)	**-31,0**(-31,00)	0	**-31,0**(-31,00)	0	**-31,0**(-31,00)	0
sko56_04	239048(225,90)	**-36,0**(-36,00)	0	**-36,0**(-36,00)	0	**-36,0**(-36,00)	0
sko56_05	467565,5(370,31)	**-89,0**(-89,00)	0	**-89,0**(-89,00)	0	**-89,0**(-88,30)	2,2
AKV_60_01	**1203171**(258,68)	**0**(0)	0	**0**(0)	0	**0**(0)	0
AKV_60_02	**557293**(123,72)	**0**(0)	0	**0**(3007,40)	6171,1	**0**(0)	0
AKV_60_03	517645,5(155,81)	**-20,0**(-20,00)	0	**-20,0**(-20,00)	0	**-20,0**(-20,00)	0
AKV_60_04	**315892**(94,45)	**0**(0)	0	**0**(0)	0	**0**(0)	0
AKV_60_05	245884(65,65)	**-7,0**(-7,00)	0	**-7,0**(-7,00)	0	**-7,0**(-7,00)	0
AKV_70_01	1076667(176,57)	**-105,0**(-105,00)	0	**-105,0**(-105,00)	0	**-105,0**(-103,75)	5,6
AKV_70_02	1107427(153,93)	**-34,0**(-34,00)	0	**-34,0**(-34,00)	0	**-34,0**(-34,00)	0
AKV_70_03	1163514,5(230,38)	**-444,0**(-443,70)	1,3	**-444,0**(-444,00)	0	**-444,0**(-281,55)	150,8
AKV_70_04	**751412**(114,97)	**0**(0)	0	**0**(0)	0	**0**(0)	0
AKV_70_05	3294390,5(553,46)	**-5,0**(-5,00)	0	**-5,0**(-5,00)	0	**-5,0**(-5,00)	0
Média	482319,48(419,73)	-34,5(-34,35)	0,3	-34,5(86,36)	247,3	-34,5(-27,90)	6,4

(a) O melhor valor conhecido obtido por Maadi et al. (2017).
Os melhores valores são destacados em negrito.

Por último, a Tabela 25 apresenta os resultados encontrados pelos algoritmos propostos para testes com instâncias de tamanho $30 \leq n \leq 70$ e $t = \frac{n}{5}$.

Os algoritmos propostos melhoraram o melhor valor conhecido da instância N40_05, as cinco instâncias com $n = 56$, duas instâncias com $n = 60$ (AKV_60_04 e AKV_60_05) e quatro instâncias com $n = 70$ (AKV_70_02 a AKV_70_05), totalizando doze melhorias em relação aos valores de referência. Para as treze instâncias restantes, os algoritmos encontraram soluções com os mesmos valores da literatura.

Considerando as médias das soluções, os três algoritmos AILS melhoraram os valores médios das soluções do algoritmo PSA para todas as instâncias. Cada algoritmo AILS atingiu $DP = 0$ para vinte e três instâncias nesse conjunto.

Tabela 25 – Resultados encontrados pelos algoritmos AILS0, AILS1 e AILS2 para instâncias com $30 \leq n \leq 70$ facilidades e $t = n/5$

Instância	PSA	AILS0		AILS1		AILS2	
	$v^{*(a)}(D_{média})$	$D_{melhor}(D_{média})$	*DP*	$D_{melhor}(D_{média})$	*DP*	$D_{melhor}(D_{média})$	*DP*
N30_01	**7289**(8,80)	**0**(0)	0	**0**(0)	0	**0**(0)	0
N30_02	**19785,5**(31,48)	**0**(0)	0	**0**(0)	0	**0**(0)	0
N30_03	**39524**(55,25)	**0**(0)	0	**0**(0)	0	**0**(0)	0
N30_04	**59587,5**(85,15)	**0**(0)	0	**0**(0)	0	**0**(0)	0
N30_05	**104449**(202,84)	**0**(0)	0	**0**(0)	0	**0**(0)	0
N40_01	**108168,5**(46,30)	**0**(0)	0	**0**(0)	0	**0**(0)	0
N40_02	**78263**(29,58)	**0**(0)	0	**0**(0)	0	**0**(0)	0
N40_03	**71428,5**(37,79)	**0**(0)	0	**0**(0)	0	**0**(0)	0
N40_04	**69254**(34,00)	**0**(0)	0	**0**(0)	0	**0**(0)	0
N40_05	81644(29,88)	**-32,0**(-32,00)	0	**-32,0**(-32,00)	0	**-32,0**(-32,00)	0
sko56_01	55248(63,59)	**-68,0**(-68,00)	0	**-68,0**(-56,15)	28,8	**-68,0**(-68,00)	0
sko56_02	398605(497,86)	**-507,0**(-507,00)	0	**-507,0**(-507,00)	0	**-507,0**(-506,60)	1,8
sko56_03	141393(151,01)	**-2,0**(-2,00)	0	**-2,0**(-2,00)	0	**-2,0**(-2,00)	0
sko56_04	264078(347,26)	**-41,0**(-41,00)	0	**-41,0**(-41,00)	0	**-41,0**(-41,00)	0
sko56_05	539340,5(775,03)	**-166,0**(-166,00)	0	**-166,0**(-166,00)	0	**-166,0**(-166,00)	0
AKV_60_01	**1302507**(170,63)	**0**(0,65)	2,9	**0**(0)	0	**0**(0)	0
AKV_60_02	**679772**(131,20)	**0**(0)	0	**0**(29,85)	133,5	**0**(0)	0
AKV_60_03	**582558**,5(117,09)	**0**(0)	0	**0**(0)	0	**0**(0)	0
AKV_60_04	342637(48,65)	**-16,0**(-16,00)	0	**-16,0**(-16,00)	0	**-16,0**(-16,00)	0
AKV_60_05	273604(46,24)	**-5,0**(-5,00)	0	**-5,0**(-5,00)	0	**-5,0**(-5,00)	0
AKV_70_01	**1272547**(332,13)	**0**(0)	0	**0**(0)	0	**0**(0)	0
AKV_70_02	1230359(248,53)	**-129,0**(-129,00)	0	**-129,0**(-129,00)	0	**-129,0**(-129,00)	0
AKV_70_03	1355131,5(258,83)	**-18,0**(92,85)	91,9	**-18,0**(-18,00)	0	**-18,0**(139,60)	64,6
AKV_70_04	827460(245,76)	**-11,0**(-11,00)	0	**-11,0**(-11,00)	0	**-11,0**(-11,00)	0
AKV_70_05	3696447,5(935,20)	**-12,0**(-12,00)	0	**-12,0**(-12,00)	0	**-12,0**(-12,00)	0
Média	544043,24(392,12)	-40,3(-35,82)	3,8	-40,3(-38,61)	6,5	-40,3(-33,96)	2,7

[a] O melhor valor conhecido obtido por Maadi et al. (2017).
Os melhores valores são destacados em negrito.

Os resultados apresentados (Tabelas 22, 23, 24 e 25) mostraram que os algoritmos propostos funcionaram muito bem nas 100 instâncias testadas, melhorando as melhores soluções conhecidas para 49 instâncias e atingindo os melhores valores conhecidos na literatura para as 51 instâncias restantes. O AILS0 apresentou um desempenho melhor do que os demais algoritmos em termos dos melhores valores e dos valores médios, como pode ser observado pelos valores de D_{melhor} ($D_{média}$). Os desvios-padrão observados (*DP*) são pequenos para os três algoritmos AILS; e $DP = 0$ para a maioria das instâncias.

6.6.4 Avaliação das três variantes do AILS para instâncias com tamanhos $250 \leq n \leq 300$

Nesta subseção, os algoritmos propostos são avaliados utilizando seis instâncias com $250 \leq n \leq 300$ dadas em Palubeckis (2015) para o SRFLP e introduzidas nesse trabalho para o PROP. Para construir instâncias PROP, foram adicionados o valor de t a cada instância SRFLP. O valor de t adicionado (mostrado na Tabela 26) produz uma instância PROP com um comprimento de

corredor menor, que é uma abordagem realista para instâncias de tamanho grande ($250 \leq n \leq 300$) e, particularmente, em um contexto de espaço limitado.

Tabela 26 – Valores de t adicionados para formar uma instância do PROP

Instância	t
p250	130
p260	131
p270	130
p280	141
P290	148
p300	158

A Tabela 27 apresenta uma comparação das soluções obtidas por AILS0, AILS1 e AILS2, dado um limite de tempo de execução (T_{Lim}) de 3600 segundos. A tabela possui a seguinte estrutura: a primeira coluna mostra os nomes das instâncias; a segunda coluna mostra os valores de t usados nos testes; a terceira coluna apresenta v^*, o melhor valor da solução encontrada pelas versões do algoritmo AILS para uma determinada instância; e, para cada algoritmo, a tabela fornece D_{melhor} ($D_{média}$) seguido pelo valor do desvio-padrão DP. A última linha da tabela mostra as médias de todas as instâncias da tabela.

Tabela 27 – Comparação dos resultados encontrados pelas versões do AILS para instâncias com $250 \leq n \leq 300$ com tempo limite de 3600s

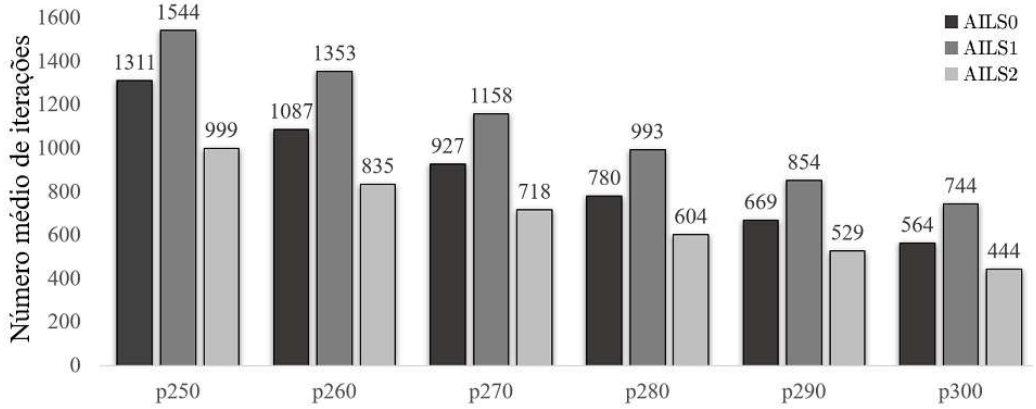
Instância	t	v^*[(a)]	AILS0		AILS1		AILS2	
			D_{melhor} ($D_{média}$)	DP	D_{melhor} ($D_{média}$)	DP	D_{melhor} ($D_{média}$)	DP
p250	130	**27289929,5**	4872,0 (22095,8)	9265,3	**0 (7419,8)**	5807,9	14939,0 (36916,1)	11268,4
p260	131	**31683937,5**	14872,0 (30007,4)	9806,8	**0 (7444)**	8490,3	12403,0 (49712,3)	18246,3
p270	130	**34552132,5**	10562,0 (37719,1)	14023,3	**0 (7708,2)**	8679,6	44560,0 (68351,6)	13778,5
p280	141	**36945712,0**	6925,0 (46427,5)	22787,8	**0 (21922,6)**	14973,0	42942,0 (81209,6)	22750,0
p290	148	**43152315,5**	21294,0 (51523,4)	22169,4	**0 (17576,7)**	19203,0	35482,0 (80522,6)	23107,5
p300	158	**48012146,0**	24797,0 (62271,9)	18964,0	**0 (31490,1)**	14325,8	38207,0 (76655,3)	27419,2
Média		36939362,2	13887,0 (41674,2)	16169,4	**0 (15593,6)**	11913,3	31422,2 (65561,2)	19428,3

[(a)] O melhor valor conhecido obtido pelas versões do algoritmo AILS.
Os melhores resultados então destacados em negrito.

Para todas as instâncias com $250 \leq n \leq 300$, o AILS1 tem um desempenho melhor em termos das melhores soluções e soluções médias. Observando os desvios-padrão, pode-se dizer que os três algoritmos são robustos, mas o AILS1 é ainda mais robusto, apresentando os menores valores dos desvios-padrão.

A Figura 20 mostra o número médio de iterações realizadas pelos algoritmos AILS para cada instância desse conjunto. Pode-se observar que AILS2 apresentou o menor número médio de iterações, o que poderia indicar que a perturbação em AILS2 requer um esforço computacional maior do que em AILS1 ou AILS0. Por outro lado, a perturbação usada no AILS1 requer um esforço menor, permitindo ao AILS1 executar um número maior de iterações dentro do mesmo limite de tempo, o que pode ter permitido atingir um desempenho superior conforme apresentado na Tabela 27. AILS0 apresentou um desempenho intermediário em relação às outras duas variantes. Parece que AILS0 e AILS2 precisariam de mais tempo para alcançar as soluções de qualidade superior obtidas por AILS1 para as instâncias com $250 \leq n \leq 300$.

Figura 20 – Comparação do número de iterações do AILS0, AILS1 e AILS2



Fonte: próprio autor.

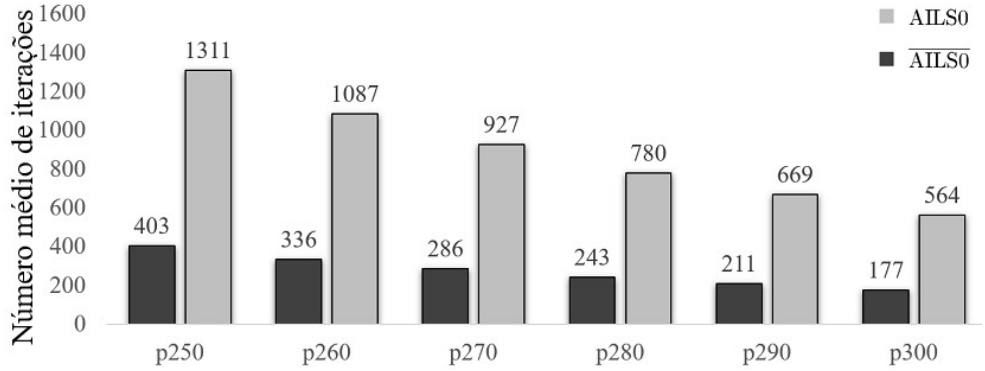
Para avaliar o desempenho de cada algoritmo nos limites de tempo de execução de 600, 1800 e 3600 segundos, nas seis instâncias com $250 \leq n \leq 300$, a Tabela 28 informa, conforme o tempo de execução aumenta, a qualidade da solução para AILS0, AILS1 e AILS2. Aqui, o valor da melhor solução conhecida (v*) para uma determinada instância é o melhor valor encontrado para essa instância pelas três variantes de AILS. Pode-se notar que, para cada limite de tempo considerado, o valor da melhor solução e da média das soluções produzidas pelo AILS1 têm qualidade superior aos outros dois algoritmos. Todos os algoritmos convergem com o aumento do tempo de execução, mas AILS2 parece ter uma convergência mais lenta.

Ainda, para avaliar o desempenho da técnica apresentada na Seção 6.2 para acelerar a busca local, estudou-se o comportamento do AILS0 e do $\overline{\text{AILS0}}$, uma versão do AILS0 sem o uso da técnica para o cálculo do ganho na função objetivo. A Figura 21 mostra uma comparação do número médio de iterações realizadas pelo AILS0 e $\overline{\text{AILS0}}$ ao longo de 20 execuções (onde cada execução tem T_{Lim} = 3600s).

Tabela 28 – Desvios dos valores da melhor solução e da média das soluções, D_{melhor} ($D_{média}$), obtido pelas versões do AILS para instâncias com $250 \leq n \leq 300$.

Instância	t	v^*	Algoritmo	Limite de tempo (s)		
				600	1800	3600
p250	130	27289929,5	AILS0	12410,0(37197,5)	11302,0(28114,0)	4872,0(22095,8)
			AILS1	3080,0(22255,2)	2108,0(12535,2)	**0**(7419,8)
			AILS2	20465,0(47711,5)	20465,0(43484,1)	14939,0(36916,1)
p260	131	31683937,5	AILS0	31869,0(50991,8)	17368,0(34807,3)	14872,0(30007,4)
			AILS1	777,0(20129,8)	20,0(9847,1)	**0**(7444,0)
			AILS2	17488,0(57937,4)	17488,0(50635,9)	12403,0(49712,3)
p270	130	34552132,5	AILS0	10562,0(53304,8)	10562,0(46398,0)	10562,0(37719,1)
			AILS1	1806,0(24836,3)	1739,0(11555,5)	**0**(7708,2)
			AILS2	44560,0(78158,9)	44560,0(68924,1)	44560,0(68351,6)
p280	141	36945712,0	AILS0	46676,0(75576,0)	6925,0(58962,4)	6925,0(46427,5)
			AILS1	2797,0(39826,4)	1279,0(29578,3)	**0**(21922,6)
			AILS2	49817,0(94708,9)	42942,0(83094,8)	42942,0(81209,6)
p290	148	43152315,5	AILS0	25972,0(74149,9)	25972,0(64565,5)	21294,0(51523,4)
			AILS1	9056,0(34421,4)	1358,0(25418,5)	**0**(17576,7)
			AILS2	50629,0(97320,5)	35482,0(85579,2)	35482,0(80522,6)
p300	158	48012146,0	AILS0	42244,0(87456,0)	42244,0(72896,3)	24797,0(62271,9)
			AILS1	10276,0(45964,6)	1103,0(37342,9)	**0**(31490,1)
			AILS2	47385,0(99646,1)	38207,0(80606,8)	38207,0(76655,3)

Figura 21 – Comparação do número de iterações do AILS0 e $\overline{\text{AILS0}}$



Fonte: próprio autor.

Como esperado, o ganho obtido pela aplicação da técnica para acelerar a avaliação das soluções vizinhas geradas pelos movimentos de troca 2-opt permitiu um aumento considerável no número de iterações realizadas pelo AILS0 em relação ao $\overline{\text{AILS0}}$. Isso significa que o AILS0 pode realizar uma investigação mais profunda da vizinhança do que o $\overline{\text{AILS0}}$, no mesmo limite de tempo.

Essa comparação mostra que o uso da técnica proposta para acelerar a busca local permite um aumento do número de iterações realizadas dentro de um determinado limite de tempo de

execução, levando a uma maior exploração do espaço de solução, o que é importante para resolver grandes problemas.

6.7 CONSIDERAÇÕES FINAIS

O PROP é NP-difícil e tem muitas aplicações práticas, particularmente no projeto de sistemas de manufatura. Neste capítulo, um algoritmo denominado *Adaptive iterated local search* (AILS) foi proposto e avaliado para o PROP.

O algoritmo AILS é adaptativo, já que alterna períodos de intensificação e diversificação e decide sobre a intensidade de uma perturbação. Além disso, o algoritmo AILS utiliza uma técnica para acelerar o cálculo da função objetivo do problema.

A perturbação foi alcançada usando movimentos de inserção (Rotina 1) ou usando movimentos de troca (Rotina 2). Três variantes de AILS foram consideradas, denominadas AILS0, AILS1 e AILS2. No AILS0, ambas as Rotinas, 1 e 2, são chamadas com probabilidade de 50% para cada uma. O procedimento de perturbação no AILS1 chama apenas a Rotina 1 e no AILS2, apenas a Rotina 2.

Experimentos com 100 instâncias de tamanho $30 \leq n \leq 70$ mostraram que, dada a mesma quantidade de tempo usada pelo algoritmo PSA em Maadi, Javidnia e Jamshidi (2017), os algoritmos AILS melhoraram os valores conhecidos de 49 instâncias e para as 51 instâncias restantes eles alcançaram os mesmos valores conhecidos. O AILS0 apresentou o melhor desempenho nestes testes, inclusive com valores médios das soluções melhores do que o PSA para todas as instâncias.

Até o momento, nenhum método na literatura foi capaz de abordar instâncias com mais de 70 facilidades. Em contraste, os algoritmos AILS aqui propostos podem lidar com instâncias PROP com tamanhos muito maiores: $250 \leq n \leq 300$. Testes com as maiores instâncias mostraram que AILS1 apresentou o melhor desempenho. O procedimento de perturbação usado no AILS1 parece exigir um esforço menor, portanto, o AILS1 pode executar um número maior de iterações no mesmo período de tempo do que o AILS0 ou o AILS2. Nota-se que o impacto do custo computacional do procedimento de perturbação pareceu ser mais significativo para as maiores instâncias.

7 ALGORITMO ILS PROPOSTO PARA O bCAP

O algoritmo proposto, denominado *Adaptive Iterated Local Search for* bCAP (AILS-bCAP), baseia-se na meta-heurística ILS, mas difere do que normalmente tem-se na literatura. O AILS-bCAP realiza iterações alternando entre duas fases de acordo com critérios de aceitação e intensidade de perturbação, distintos em cada uma das fases. A Fase 1 promove um aspecto de intensificação das soluções no processo de busca do ILS ao considerar o critério de aceitação que escolhe sempre a melhor solução encontrada nas iterações anteriores do ILS e uma intensidade de perturbação variável. Enquanto que, na Fase 2, um aspecto de diversificação é privilegiado ao promover a busca baseando-se na última solução encontrada e uma intensidade da perturbação constante. Além disso, técnicas de aceleração do cálculo do ganho na função objetivo f_1 (Equação 2.5), tanto no movimento de vizinhança por inserção, inicialmente usadas em Guan e Lin (2016) e Cravo e Amaral (2019) para o SRFLP, quanto no movimento de troca 2-opt, propostas em Kothari e Ghosh (2013b) também para o SRFLP, são adaptadas para contexto do bCAP e usadas nas rotinas do procedimento de busca local proposto para o AILS-bCAP.

7.1 OTIMIZAÇÃO COMBINATÓRIA MULTIOBJETIVO

Um problema de otimização multiobjetivo (POM) apresenta $r \geq 2$ objetivos e que na maioria das vezes são conflitantes, como é o caso da formulação do bCAP. O POM pode ser definido da seguinte forma: dado um vetor de variáveis de decisão com dimensão n, $x = \{x_1, ..., x_n\}$ no espaço de busca X, queremos encontrar um vetor $x^* \in X$ que minimize simultaneamente os r objetivos. Assim, o modelo geral de um POM, no formato de minimização, é escrito como (PÉREZ, 2012):

$$\text{(POM)}\ min\, z = f(x) = \{f_1(x),\ ...,\ f_r(x)\} \tag{7.1}$$

Onde $f_i(x): R^n \longrightarrow R$ e $i = 1, ..., r$. Em geral, X é definido por uma série de restrições e limites de especificação para as variáveis de decisão.

Otimizar um POM consiste em encontrar valores aceitáveis para os r objetivos para uma tomada de decisão. Como os objetivos são conflitantes entre si, não existirá uma solução única para o

problema e sim um conjunto de soluções de compromisso (*trade-off*). Assim, o conceito Pareto-otimalidade é utilizado para a caracterização e obtenção das soluções (FERREIRA, 2020).

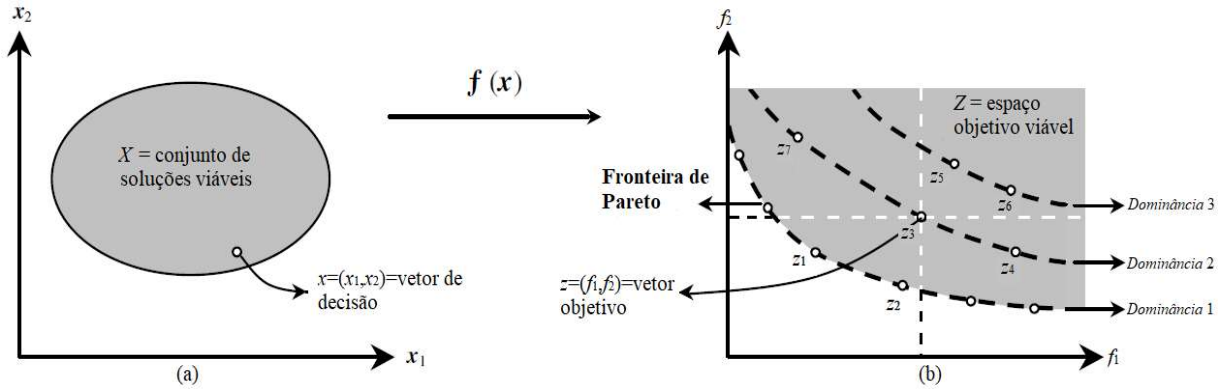
Se todos os objetivos são de minimização, pode-se descrever o conceito de Pareto-otimalidade formalmente, pelas seguintes definições: dadas duas soluções quaisquer, x e y, a solução x domina a solução y ($x \preccurlyeq y$) se as seguintes condições forem verdadeiras (DEB, 2011):

- a solução viável x não é pior que y, ou seja, $f_i(x) \leq f_i(y)$ para $i = 1, ..., r$; e
- a solução viável x é estritamente melhor que y em pelo menos um objetivo, isto é, $f_i(x) < f_i(y)$ em pelo menos uma função objetivo i.

Uma solução é dita Pareto-ótima, não-dominada ou eficiente se ela não é dominada por nenhuma outra solução viável no espaço de busca X. Assim, a solução x não pode ser melhorada com relação a qualquer objetivo sem que ocorra uma piora em pelo menos um objetivo.

O conjunto das soluções não-dominadas em X é denominado conjunto Pareto-ótimo e a imagem de um determinado conjunto Pareto-ótimo é chamada de fronteira de Pareto (KONAK; COIT; SMITH, 2006). A Figura 22 ilustra o espaço de soluções viáveis e o espaço objetivo viável com a relação de dominância entre as sete soluções $\{z_1, z_2, z_3, z_4, z_5, z_6, z_7\}$ de um problema de minimização com dois objetivos.

Figura 22 – (a) conjunto de soluções viáveis e (b) espaço objetivo viável e grau de dominância em um problema de minimização com dois objetivos



Fonte: adaptado Hashimoto (2004)

Na Figura 22, a solução z_1 oferece um valor menor para f_1, mas um valor maior para f_2, se comparado com a solução z_2, ilustrando que a diminuição de uma função objetivo implica no aumento da outra. Desse modo, a avaliação das soluções viáveis deve levar em conta o conceito

de dominância de Pareto, definido anteriormente. No exemplo de dois objetivos da Figura 22, tomando como referência o vetor objetivo z_3, os vetores objetivos nos pontos z_1 e z_2 dominam z_3, os pontos z_5 e z_6 são dominados por z_3 e os pontos z_4 e z_7 são indiferentes em relação a z_3.

Não é possível determinar uma única solução dentre as soluções Pareto-ótimas sem informações adicionais para a tomada de decisão, ou seja, na ausência de outras informações, todas as soluções do Pareto são igualmente importantes e, em uma situação típica do mundo real, o processo de tomada de decisão pode levar em conta muitos fatores. Assim, as heurísticas para otimização multiobjetivo precisam encontrar um conjunto de soluções o mais próximo possível da fronteira de Pareto e encontrar um conjunto diversificado de soluções viáveis (ARAUJO; POLDI; SMITH, 2014).

É difícil de estender técnicas convencionais de otimização, como o método simplex, para os problemas multiobjetivos, pois não foram projetadas para considerarem múltiplas soluções. Entretanto, há tempos os algoritmos populacionais são comumente propostos para a otimização multiobjetivo (FONSECA; FLEMING, 1995). A grande maioria das metaheurísticas utilizadas é baseada em Algoritmos Genéticos. A preferência está fundamentada no fato dos Algoritmos Genéticos utilizarem um conjunto de soluções que podem conter informações sobre várias regiões do espaço de busca, oferecendo, portanto, maiores possibilidades de encontrar o conjunto Pareto-ótimo ou uma aproximação (ARROYO, 2002; DEB, 2011). Mais detalhes sobre otimização multiobjetivo podem ser encontrados em Fonseca e Fleming (1995), Arroyo (2002), Deb (2011) e Zhou et al. (2011).

7.2 REPRESENTAÇÃO E GERAÇÃO DA SOLUÇÃO INICIAL

Uma solução é representada no AILS-bCAP como dois arranjos π_1 e π_2 contendo t e $(n - t)$ facilidades, respectivamente. O número inteiro t define o ponto de quebra do arranjo de modo que seja possível a representação das duas linhas do bCAP.

A geração de solução inicial utilizada é um procedimento simples que retorna uma solução com a disposição de n facilidades. A disposição das facilidades é gerada de maneira aleatória. Assim, dado o ponto de quebra t do arranjo e uma lista L com as n facilidades, o procedimento escolhe aleatoriamente t facilidades, uma por uma, e adiciona à direita da sequência no arranjo da linha 1 (π_1). O mesmo procedimento é executado para selecionar as $(n - t)$ facilidades restantes da

lista L para construir a sequência na linha 2 (π_2). O Algoritmo 15 exemplifica a construção de uma solução inicial $S = (\pi_1,\pi_2)$. O algoritmo recebe como parâmetros o número n de facilidades e o ponto de quebra t.

Algoritmo 15 – Gerar Solução Inicial CAP – AILS-bCAP

```
entrada: n, t
saída: S
 1: L ← {1, ..., n}
 2: π1 ← ∅; π2 ← ∅
 3: Enquanto |π1| < t faça
 4:     i ← ObterFacilidadeAleatoriamente(L)
 5:     π1 ← π1+{i}  /*Adiciona i à direita das facilidades em π1*/
 6: Fim
 7: Enquanto |π2| < n - t faça
 8:     i ← ObterFacilidadeAleatoriamente(L)
 9:     π2 ← π2+{i} /*Adiciona i à direita das facilidades em π2*/
10: Fim
11: S ← (π1, π2)
12: retorne S
```

Na Linha 1 do Algoritmo 15 é inicializada a lista L com as facilidades de 1, ..., n; na Linha 2, os arranjos π_1 e π_2 são inicializados vazios; no laço das Linhas 3-6, o arranjo da linha 1 do layout é criado. A cada iteração, uma facilidade é selecionada aleatoriamente e removida da lista L (Linha 4) e, então, adicionado no arranjo π_1. O laço termina quando as t facilidades forem adicionadas ao arranjo π_1. De modo semelhante, no laço das Linhas 7-10, as (n - t) facilidades restantes da lista L são escolhidas, aleatoriamente, e adicionadas ao arranjo π_2. Na Linha 11, a solução completa S é criada, e ao final, o algoritmo retorna à solução S (Linha 12).

7.3 PROCEDIMENTO PARA COMPARAÇÃO DE SOLUÇÕES

As soluções normalmente propostas na literatura para otimização multiobjetivo utilizam algoritmos populacionais e sendo a saída do modelo bCAP um conjunto de soluções em uma fronteira de Pareto, consistindo assim, de um conjunto de soluções não-dominadas, naturalmente Kalita e Datta (2014) e Kalita, Datta e Palubeckis (2019) propuseram implementações do algoritmo pGA para resolver o bCAP. Entretanto, como sugerido por Kalita, Data e Palubeckis (2019), no bCAP a otimização das soluções nos dois pontos extremos em uma fronteira de Pareto faz mais sentido na visão de um tomador de decisão, pois o decisor pode não estar interessado em soluções intermediárias nesse problema específico, mas em minimizar o custo diário do fluxo de materiais ou o custo único da implantação do corredor,

tendo em vista que a redefinição do layout não é realizada com alta frequência. Assim, a implementação do AILS-bCAP, considerou a otimização dos dois pontos nos extremos de uma fronteira de Pareto, sendo um extremo a solução em relação ao melhor custo total de manuseio de materiais, f_1^{best} (o melhor valor de f_1 dado pela Equação (2.5)), e o outro extremo a solução em termos do melhor desvio percentual do comprimento do corredor, f_2^{best} (o melhor valor de f_2 dado pela Equação (2.6)).

O pseudocódigo do Algoritmo 16 exemplifica o procedimento para avaliar duas soluções, dados os vetores objetivos associados. O algoritmo recebe como parâmetros dois vetores com os custos das soluções a serem avaliadas. A ordem das componentes do vetor define a prioridade das funções objetivos do bCAP a serem consideradas. O algoritmo devolve qual das soluções é a melhor, dada uma priorização definida.

Algoritmo 16 – *Avaliar_soluções*, Avaliação das Soluções no AILS-bCAP

```
entrada: x, y
saída: m
 1: Se x0 < y0 então
 2:     m ← 0
 3: Senão Se x0 > y0 então
 4:     m ← 1
 5: Senão Se x1 < y1 então
 6:     m ← 0
 7: Senão
 8:     m ← 1
 9: Fim
10: retorne m
```

O Algoritmo 16 prioriza a solução que tem o menor custo associado à primeira posição dos vetores objetivos *x* e *y* (Linha 1 e Linha 3), e em caso de igualdade, na Linha 5, utiliza-se o valor do custo associado à segunda posição dos vetores. Ao final, o algoritmo retorna o valor *m = 0*, se o vetor objetivo *x* for melhor, ou *m* = 1, se o vetor objetivo *y* for melhor, dada a priorização definida.

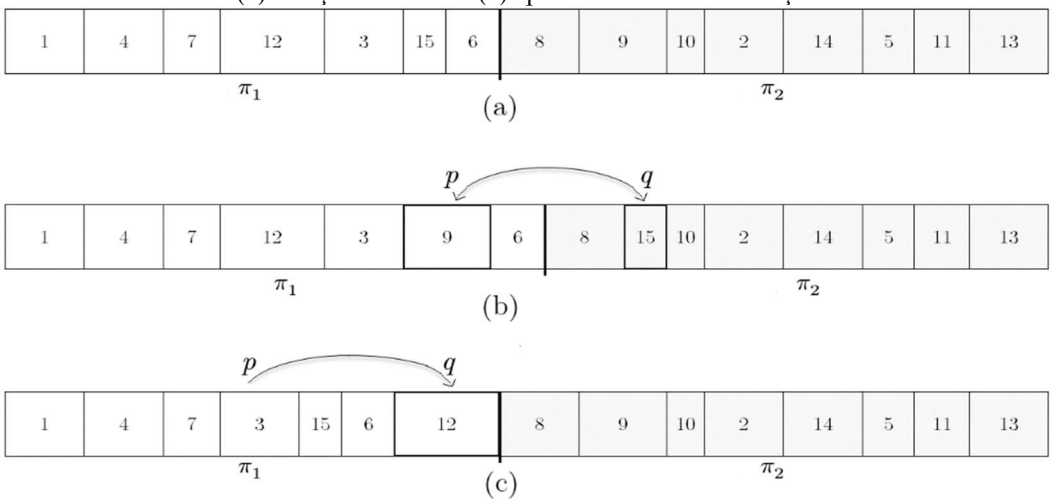
Ao comparar duas soluções A e B no extremo (f_1^{best}, f_2), os vetores *x* e *y* no Algoritmo 16 terão as componentes definidas como $x = (f_1(A), f_2(A))$ e $y = (f_1(B), f_2(B))$. Assim, a solução com o menor custo de manuseio de materiais é escolhida na Linha 2 ou Linha 4. Caso os custos sejam iguais (Linha 5), o melhor desvio percentual do comprimento do corredor é considerado, ou seja, a solução com o menor valor da função objetivo f_2 é escolhida na Linha 7 ou Linha 9. Do mesmo modo, ao avaliar duas soluções A e B no extremo do melhor desvio percentual do

comprimento do corredor, os vetores x e y terão as componentes como $x = (f_2(A), f_1(A))$ e $y = (f_2(B), f_1(B))$.

7.4 PROCEDIMENTO DE BUSCA LOCAL

O procedimento de busca local é composto por duas rotinas auxiliares que aplicam movimentos de troca 2-opt ou de inserção. Esses movimentos são amplamente utilizados em procedimentos heurísticos, independente do problema a ser resolvido, na tentativa de intensificar a busca no espaço de soluções, e também usados no contexto dos problemas de layout em linha. Além disso, uma terceira rotina é utilizada na busca local para determinar o melhor ponto de quebra em uma solução corrente. As rotinas são aplicadas de forma escalonada e o processo é repetido enquanto houver melhora na solução. A Figura 23 exemplifica os movimentos 2-opt e de inserção.

Figura 23 – (a) Exemplo de solução bCAP com $t = 7$, (b) solução vizinha de (a) após movimento de troca 2-opt e (c) solução vizinha de (a) após movimento de inserção



Fonte: próprio autor.

A rotina da busca local, denominada *Busca_entre_linhas* utiliza movimentos de troca 2-opt com trocas de facilidades ocorrendo apenas entre linhas opostas. Deste modo, são aplicados movimentos de trocas considerando os pares (p, q) com $p = 1, ..., |\pi_1|$; $q=|\pi_1|+1, ..., n$. A Figura 23 (a) apresenta uma solução para o bCAP e na Figura 23 (b) as facilidades das posições $p = 6$ e $q = 9$ foram trocadas. Note que, nesse caso, as duas funções objetivos, f_1 e f_2, deverão

ser reavaliadas, pois além da alteração da ordem das facilidades nos arranjos, também ocorre alteração no comprimento do corredor. O pseudocódigo do Algoritmo 17 exemplifica a rotina *Busca_entre_linhas* que realiza a busca baseada nos movimentos 2-opt. O algoritmo recebe a solução corrente S e a função biobjetivo z, que retorna o vetor objetivo dada uma solução do bCAP, com a priorização das funções de custos definida na ordem desejada.

Algoritmo 17 – Rotina *Busca_entre_linhas*, movimento 2-opt

```
entrada: S, z
saída: S, melhorou
 1: melhorou ← 0
 2: S* ← S
 3: Para p ← 1 até |S*_π1| faça
 4:    Para q ← |S*_π1| + 1 até |S*_π1|+|S*_π2| faça
 5:       S' ← troca_2-opt (S*, p, q);
 6:       r ← Avaliar_soluções (z (S'), z (S*))
 7:       Se r = 0 então
 8:          S* ← S'
 9:          melhorou ← 1;
10:       Fim
11:    Fim
12: Fim
13: retorne (S*, melhorou)
```

Na Linha 1, do Algoritmo 17, a *flag melhorou*, que sinaliza se a solução corrente foi melhorada durante o processo de busca, é inicializada com zero. Na Linha 2, a melhor solução local S^* é inicializada. Os laços das Linhas 3-12 e 4-11 perfazem os pares (p, q) a serem utilizados na Linha 5, para gerar uma solução vizinha S', com a troca das facilidades das posições p e q. Na Linha 6, as soluções S' e S^* são avaliadas de acordo com o critério adotado (Algoritmo 16, Seção 7.3). Se a solução vizinha S' for melhor (Linha 7), a melhor solução corrente S^* (Linha 8) e a *flag melhorou* (Linha 9) são atualizadas. Ao final, Linha 13, o algoritmo retorna a melhor solução encontrada S^* e a *flag melhorou*.

A segunda rotina utilizada na busca local, denominada *Busca_na_linha*, é baseada no movimento de inserção em que os vizinhos são gerados ao inserir uma facilidade da posição p em uma nova posição q. A rotina percorre os pares (p, q) para todas as posições $p = 1, ..., |\pi_1|$ na tentativa de inserir a facilidade da posição p nas posições $q = 1, ..., |\pi_1|$ com $p \neq q$ no arranjo π_1 (linha 1 do layout). Caso uma inserção gere um vizinho melhor, o movimento é aceito, a solução corrente é atualizada e a busca segue para a próximo par (p, q) da sequência. De modo semelhante, são gerados os pares (p, q) para as posições $p = |\pi_1|+1, ..., n$, na tentativa de inserir a facilidade da posição p nas posições $q = |\pi_1|+1, ..., n$ com $p \neq q$ relativos ao arranjo π_2 (linha 2 do layout).

A Figura 23 (c) exemplifica um movimento de inserção realizado na Figura 23 (a). A facilidade da posição $p = 4$ foi inserida na posição $q = 7$, sendo as facilidades intermediárias deslocadas para que a facilidade seja inserida na nova posição. Aqui, apenas a função objetivo f_1 precisará ser reavaliada, pois os movimentos são restritos a cada linha do layout, não sendo inseridas as facilidades em linhas opostas. O pseudocódigo do Algoritmo 18 exemplifica a rotina *Busca_na_linha* implementada com o movimento de inserção. O algoritmo recebe a solução corrente S e a função biobjetivo z que retorna o vetor objetivo dada uma solução do bCAP, com a priorização das funções de custos definida na ordem que se deseja avaliar.

Algoritmo 18 – Rotina *Busca_na_linha*, exemplo com movimento de inserção

```
entrada: S, z
saída: S*, melhorou
 1: melhorou ← 0
 2: S* ← S
 3: Para p ←1 até |S*_π1| faça
 4:    Para q ← 1 até |S*_π1| faça
 5:       Se p ≠ q então
 6:          S' ← inserção (S*, p, q)
 7:          r ← Avaliar_soluções (z(S'), z(S*))
 8:          Se (r = 0) então
 9:             S* ← S'
10:             melhorou ← 1
11:          Fim
12:       Fim
13:    Fim
14: Fim
15: Para p ← |S*_π1|+1 até |S*_π1|+|S*_π2| faça
16:    Para q ← |S*_π1|+1 até |S*_π1|+|S*_π2| faça
17:       Se p ≠ q então
18:          S' ← inserção (S*, p, q)
19:          r ← Avaliar_soluções (z(S'), z(S*))
20:          Se r = 0 então
21:             S* ← S'
22:             melhorou ← 1
23:          Fim
24:       Fim
25:    Fim
26: Fim
27: retorne (S*, melhorou)
```

Nas Linhas 1 e 2, são inicializadas a *flag melhorou*, que indica se uma solução melhor foi encontrada, e a melhor solução corrente S^*, respectivamente. Nos laços das Linhas 3-14 e 4-13 são gerados os pares (p, q), pertencentes ao arranjo da linha 1 do layout, utilizados para a geração de uma solução vizinha S' com a inserção da facilidade da posição p na nova posição q, na Linha 6 (caso $p \neq q$, Linha 5). Na Linha 7, as soluções S' e S^* são avaliadas de acordo com o critério adotado (Algoritmo 16, Seção 7.3). Se a solução vizinha S' for melhor, então a melhor solução corrente S^* (Linha 9) e a *flag melhorou* (Linha 10) são atualizadas.

De modo semelhante, os pares (p, q) pertencentes ao arranjo da linha 2 do layout são gerados nos laços das Linhas 15-26 e 16-25 sendo utilizados para a geração de uma solução vizinha S' na Linha 18 (caso $p \neq q$, Linha 17). Na Linha 19, as soluções S' e S^* são avaliadas (Algoritmo 16, Seção 7.3). Se a solução vizinha S' for melhor, a melhor solução corrente S^* (Linha 21) e a *flag melhorou* (Linha 22) são atualizadas. Ao final, Linha 27, o algoritmo retorna a melhor solução encontrada S^* e a *flag melhorou*.

Considerando cada linha individualmente do bCAP, temos um subproblema SRFLP. Como pode ser observado na Equação (2.5), os dois primeiros somatórios expressam a função objetivo do SRFLP aplicado a cada linha do layout (KALITA; DATTA; PALUBECKIS, 2019); e o terceiro somatório acrescenta o custo dos pares (i, j) de facilidades com i pertencente à linha 1 e j pertencente à linha 2. Assim, a rotina *Busca_na_linha* realiza os movimentos de inserção considerando as seguintes possibilidades:

- Para $(p, q) \in \pi_1$, não há alteração na ordem e nem nas posições das facilidades pertencentes a π_2. Então, o segundo somatório não precisará ser avaliado e o ganho na função objetivo será calculado pelo ganho do subproblema SRFLP, representado pelo arranjo da linha 1, mais o custo definido pelo terceiro somatório na Equação (2.5); e
- Para $(p, q) \in \pi_2$, a alteração não ocorrerá na ordem e nem nas posições das facilidades pertencentes a π_1. Então, não será avaliado o primeiro somatório e o ganho na função objetivo será calculado pelo ganho do subproblema SRFLP, representado pelo arranjo da linha 2, acrescido do custo definido pelo terceiro somatório na Equação (2.5).

Para calcular o ganho no valor da função objetivo f_1 dos arranjos π_1 e π_2 (1º e 2º somatórios da Equação (2.5)) adaptou-se a técnica descrita para o SRFLP em Guan e Lin (2016) e Cravo e Amaral (2019) (APÊNCIDE A). A técnica aproveita o cálculo realizado para determinar o ganho na função objetivo para o par (p, q) ao realizar o cálculo do par (p, q+1), caso o movimento com o (p, q) não apresente melhora na função objetivo, melhorando, assim, o desempenho da rotina *Busca_na_linha* em termos de esforço computacional. Entretanto, o 3º somatório do Equação (2.5) é recalculado a cada movimento.

Uma alternativa a ser considerada para o Algoritmo 18 é a substituição do movimento de inserção das Linhas 6 e 18 pelo movimento de troca 2-opt. Dessa forma, o Algoritmo 18 explorará a vizinhança da solução com trocas 2-opt das facilidades das posições p e q de cada linha do layout. De forma análoga às considerações apresentadas anteriormente para o movimento de inserção, ao calcular o ganho no objetivo f_1 dos arranjos π_1 e π_2 (1º e 2º

somatórios da Equação (2.5)) para a troca 2-opt, a técnica descrita em Kothari e Ghosh (2013b) foi utilizada. A técnica consiste em aproveitar o cálculo realizado para determinar o ganho na função objetivo para o par (p, q) no cálculo do par (p, q+1), reduzindo o esforço computacional da busca. Contudo, o 3º somatório da Equação (2.5) é recalculado a cada movimento.

As rotinas de busca apresentadas não exploram soluções vizinhas com a alteração do ponto de quebra t do layout. Assim, uma rotina denominada *Busca_t* é definida para realizar a busca pelo melhor ponto de quebra em uma solução corrente S. Nesse caso, são geradas soluções vizinhas com os pontos de quebra $t = t_{min}, ..., t_{max}$ e a melhor solução é selecionada. O Algoritmo 19 mostra o pseudocódigo da rotina *Busca_t*. O algoritmo recebe a solução corrente S; os limites para os pontos de quebra, t_{min} e t_{max}; e a função biobjetivo z que retorna o vetor objetivo dada uma solução do bCAP, com a priorização das funções dos custos na ordem de avaliação desejada.

Algoritmo 19 – Rotina *Busca_t*

entrada: S, t_{min}, t_{max}, z
saída: S^*, *melhorou*
1: $\pi \leftarrow S_{\pi_1} \cup S_{\pi_2}$
2: *melhorou* $\leftarrow 0$, $n \leftarrow |\pi|$
3: $S^* \leftarrow S$
4: **Para** $t \leftarrow t_{min}$ **até** t_{max} **faça**
5: $S' \leftarrow (\pi\ (1,\ ...,\ t),\ \pi\ (t+1,\ ...,\ n))$
6: $r \leftarrow$ *Avaliar_soluções* $(z\ (S'), z\ (S^*))$
7: **Se** $r = 0$ **então**
8: $S^* \leftarrow S'$
9: *melhorou* $\leftarrow 1$
10: **Fim**
11: **Fim**
12: **retorne** (S^*, *melhorou*)

Na Linha 1 do Algoritmo 19 é criado um arranjo π com as facilidades organizadas de acordo com os arranjos π_1 e π_2 da solução inicial S. Na Linha 2, a *flag melhorou* e a variável n são inicializadas. Na Linha 3, a melhor solução corrente S^* é inicializada com a solução S. No laço das Linhas 4-11 os valores de t são gerados e, para cada t, uma solução vizinha S' é gerada considerando o novo ponto de quebra no arranjo π, Linha 5. As soluções S' e S^* são avaliadas na Linha 6 de acordo com o critério adotado (Seção 7.3). Se a solução S' for melhor, a melhor solução corrente S^* é atualizada (Linha 8) e, na Linha 9, a *flag melhorou* é atualizada. Ao final, na Linha 12, o algoritmo retorna a melhor solução encontrada S^* e a *flag melhorou*.

Dadas as rotinas descritas anteriormente, a busca local proposta para o bCAP é definida como mostrada no Algoritmo 20. O algoritmo recebe como parâmetros a solução corrente S, os limites

t_{min} e t_{max} para a rotina *Busca_t* e a função biobjetivo z que retorna o vetor objetivo dada uma solução do bCAP, com a priorização das funções de custos na ordem de avaliação desejada.

Algoritmo 20 – Procedimento de Busca Local

entrada: S, t_{min}, t_{max}, z
saída: S^*
1: *melhorou_ entre_linhas* ← 0; *melhorou_ na_linha* ← 0; *melhorou_t* ← 0
2: S^* ← S
3: **Faça**
4: **Faça**
5: (S^*, *melhorou_entre_linhas*) ← *Busca_entre_linhas*(S^*, z)
6: (S^*, *melhorou_na_linha*) ← *Busca_na_linha* (S^*, z)
7: **Enquanto** (*melhorou_entre_linhas* + *melhorou_na_linha* ≠ 0)
8: (S^*, *melhorou_t*) ← *Busca_t* (S^*, $t_{\min}$, $t_{\max}$, z)
9: **Enquanto** (*melhorou_t* ≠ 0)
10: **retorne** S^*

No Algoritmo 20, as *flags melhorou_entre_linhas*, *melhorou_na_linha* e *melhorou_t* indicam se as rotinas auxiliares melhoraram a solução corrente, sendo inicializadas na Linha 1. Na Linha 2, a melhor solução corrente S^* é inicializada. O laço externo, definido nas Linhas 3-9, itera enquanto a solução for melhorada com a troca do ponto de quebra pela rotina *Busca_t* (Linha 8). No laço interno das Linhas 4-7, a solução corrente S^* é submetida às rotinas *Busca_entre_linhas* e *Busca_na_linha*, e termina quando não houver melhora. Ao final, na Linha 10, o algoritmo devolve a melhor solução encontrada S^*.

Kalita, Datta e Palubeckis (2019) propõem uma hibridização do algoritmo pGA com uma busca local proposta por Palubeckis (2015) para o SRFLP, que também utiliza técnicas para acelerar o cálculo do ganho na função objetivo no movimento de inserção. Entretanto, a busca local é aplicada aos indivíduos da população do pGA apenas para cada linha do layout bCAP, como um subproblema SRFLP. Em seguida, os autores realizam uma recombinação das linhas, para cada indivíduo, considerando os arranjos antes e após a aplicação da busca local, de tal modo que são gerados três possíveis novos vizinhos a serem inseridos na população do pGA em caso de melhora da função objetivo f_1. A abordagem utilizada no pGA limita o espaço de soluções (vizinhança) aos possíveis movimentos realizados apenas em cada linha, não havendo a troca ou inserção de facilidades entre as linhas.

A busca local proposta diferencia-se da abordagem utilizada no pGA, ao explorar, utilizando a rotina *Busca_entre_linhas*, o espaço de soluções com movimentos de trocas 2-opt em linhas opostas do layout, além de utilizar a busca por um novo ponto de quebra t com a rotina *Busca_t*.

7.5 PROCEDIMENTO DE PERTURBAÇÃO

O procedimento de perturbação no AILS-bCAP gera uma nova solução S', aplicando aleatoriamente perturbações no arranjo da solução corrente S. Na perturbação são utilizadas duas sub-rotinas denominadas *inverter* e *inserir*, sendo aplicadas dada uma probabilidade P.

No procedimento *inverter*, dado um arranjo e o par (i, j), as facilidades das posições $[i, ..., j]$ do arranjo têm suas posições invertidas. Para exemplificar, supondo o arranjo π = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] e (i, j) = (4, 7), o procedimento *inverter* perturba o arranjo π tornando π = [1, 2, 3, **7**, **6**, **5**, **4**, 8, 9, 10]. Enquanto que, no procedimento *inserir*, dado um arranjo e o par (i, j), a facilidade da posição i é inserida na posição j, consequentemente, as facilidades das posições intermediárias são deslocadas. Por exemplo, supondo o arranjo π = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] e (i, j) = (4, 7), o procedimento *inserir* perturba o arranjo π tornando π = [1, 2, 3, **5**, **6**, **7**, **4**, 8, 9, 10].

O procedimento de perturbação do AILS-bCAP mantém uma lista com as posições das facilidades para serem realocadas. Assim, o valor da intensidade da perturbação I_r, em cada Fase $r = \{1, 2\}$, é utilizado para determinar a quantidade de posições a serem selecionadas da lista de posições disponíveis. O valor de I_r recebido pelo procedimento de perturbação é calculado durante o processo iterativo do AILS-bCAP, conforme Equação 7.1:

$$I_r = \lfloor \alpha_r \times \ n \rfloor \tag{7.1}$$

Onde α_r define o percentual do total de facilidades n que serão consideradas na perturbação em cada Fase $r = \{1, 2\}$ do AILS-bCAP.

O valor de I_1, utilizado na Fase 1 do AILS-bCAP, varia durante o processo de busca pela variação do valor de α_1. Inicialmente $\alpha_1 = \alpha_{min}$ e durante a Fase 1 do AILS-bCAP é ajustado por um fator, calculado pela razão entre custo da função objetivo f_1, a solução ótima local corrente S^k e o custo da função objetivo da melhor solução S^*, conforme Equação (7.2):

$$\alpha_1 = \alpha_1 \times \frac{f_1(S^k)}{f_1(S^*)} \tag{7.2}$$

Onde f_1 é a função do custo total de manuseio de materiais entre as facilidades do bCAP definida pela Equação (2.5). Após a atualização do valor de α_1, é verificado se o mesmo extrapolou o intervalo $[\alpha_{min}, \alpha_{max}]$. Caso afirmativo, o valor de α_1 é reiniciado, ou seja, $\alpha_1 = \alpha_{min}$.

Na Fase 2 do AILS-bCAP, para o cálculo da intensificação I_2, optou-se por uma estratégia diferente da Fase 1, sendo utilizado um valor fixo, $\alpha_2 = \alpha$, onde α é obtido em uma etapa de calibração dos parâmetros do algoritmo. Com isso, tem-se um comportamento diferente para o mesmo procedimento de perturbação dado pela diferenciação do cálculo da intensidade.

O pseudocódigo do Algoritmo 21 exemplifica o procedimento de perturbação proposto. O procedimento recebe como parâmetros a melhor solução corrente como *S*, o número total de facilidades, *n*, a intensidade da perturbação, *I* e a probabilidade para a escolha da sub-rotina de perturbação, *P*.

Algoritmo 21 – Procedimento de Perturbação

```
Dado: S, n, I, P
Resultado: S'
 1: S' ← S
 2: L ← {1, ..., n}
 3: Enquanto I ≥ 2 faça
 4:    (i, j) ← SelecionarAleatoriamentePar(L)
 5:    Se rand() < P então
 6:       S' ← inserir (S', i, j)
 7:    Senão
 8:       S' ← inverter (S', i, j)
 9:    Fim
10:    I ← I - 2 /* duas posições foram removidas a lista L */
11: Fim
12: retorne S'
```

Na Linha 1, a solução *S'* é inicializada com a solução *S*. Na Linha 2, uma lista *L* com as posições disponíveis é definida. No laço das Linhas 3-11 são aplicadas as perturbações na solução corrente *S'*. Na Linha 4, é selecionado aleatoriamente o par de posições (*i*, *j*) da lista *L* de posições disponíveis. Na Linha 5, *rand*() gera aleatoriamente um número real no intervalo $[0, 1)$ e, se o valor for menor que a probabilidade *P*, a sub-rotina *inserir* é utilizada na Linha 6. Caso contrário, a sub-rotina *inserir* é aplicada na Linha 8. Como, a cada iteração, duas facilidades são removidas da lista de posições disponíveis, o valor de *I* é decrementado de duas unidades (Linha 10). Ao final, Linha 12, a solução *S'* é retornada.

7.6 CRITÉRIO DE ACEITAÇÃO DO AILS-bCAP

O critério de aceitação é escolha da solução que irá ser submetida na próxima iteração e influencia o nível de intensificação/diversificação na pesquisa (LOURENÇO; MARTIN; STÜTZLE, 2003). As duas fases do AILS-bCAP usam diferentes critérios de aceitação. Na Fase 1, para intensificar a busca, adotou-se o critério de que a melhor solução encontrada até então será submetida à próxima iteração. Assim, define-se o procedimento de aceitação pela Equação (7.3):

$$S = \begin{cases} S^k & \text{se } \textit{Avaliar\_soluções } (z(S^k), z(S^*)) = 0 \\ S^* & \text{c. c.} \end{cases} \tag{7.3}$$

Onde S^* é a melhor solução atual, S^k é a melhor solução encontrada na iteração corrente k e *Avaliar_soluções* é a função que avalia as soluções S^k e S^*, dependendo do extremo da fronteira considerado, de acordo com o critério adotado, conforme descrito na Seção 7.3.

Na Fase 2, para diversificar a busca do AILS-bCAP, o critério de aceitação escolhe para a próxima iteração do algoritmo a última solução ótima local encontrada, que não necessariamente é a melhor solução. Assim, em uma dada iteração k da Fase 2 no AILS-bCAP, o critério de aceitação é dado pela Equação (7.3):

$$S = S^k \tag{7.3}$$

Onde S^k é a última solução encontrada mais recentemente.

7.7 O ALGORITMO AILS-bCAP PROPOSTO

Com a construção da solução inicial, da busca local, da perturbação e os critérios de aceitação definidos para as duas fases, o algoritmo proposto para o bCAP é definido como mostrado no Algoritmo 22. O AILS-bCAP recebe como entrada os seguintes parâmetros: n, tamanho do problema; os limites t_{min} e t_{max} utilizados no procedimento *Busca_t*; *nIterWI*, número máximo de iterações sem melhora na solução na Fase 1; *nIter*, número máximo de iterações da Fase 2; T_{Lim}: o tempo limite de execução do algoritmo; α_{min} e α_{max}, intervalo de α_1 para o cálculo da intensidade da perturbação da Fase 1; α, fator para o cálculo da intensidade da perturbação da

Fase 2; e *P*, probabilidade para escolha das rotinas na perturbação. No pseudocódigo, as partes destacadas em cinza representam a execução em paralelo no AILS-bCAP.

Algoritmo 22 – Algoritmo AILS-bCAP proposto

entrada: n, t_{min}, t_{max}, $nIterWI$, $nIter$, T_{Lim}, α_{min}, α_{max}, α, P
saída: A^*, B^*

1: $t \leftarrow \lfloor \frac{n}{2} \rfloor$, $k \leftarrow 1$, $\alpha_1 \leftarrow \alpha_{min}$, $\alpha_2 \leftarrow \alpha$, $z_1 \leftarrow z(S):(f_1(S), f_2(S))$, $z_2 \leftarrow z(S):(f_2(S), f_1(S))$,
2: $A^0 \leftarrow GerarSoluçãoInicialCAP\ (n, t)$

/* thread 1 – extremo f_1^{best} */	/* thread 2 – extremo f_2^{best} */
3: $A^k \leftarrow BuscaLocal\ (A^0,\ t_{min}, t_{max}, z_1)$	$B^k \leftarrow BuscaLocal\ (A^0,\ t_{min}, t_{max}, z_2)$
4: $A^* \leftarrow A^k$ /* Inicializa a melhor no extremo (f_1^{best}, f_2) */	$B^* \leftarrow B^k$ /* Inicializa a melhor no extremo (f_1 , f_2^{best}) */

5: **Enquanto** $tempo\_decorrido \leq T_{Lim}$ **faça**

/* Fase 1 */

6: $iter\_melhora \leftarrow k$
7: **Enquanto** k - $iter\_melhora < nIterWI$ **E** $tempo\_decorrido \leq T_{Lim}$ **faça**
8: $k \leftarrow k + 1$
9: $I_1 \leftarrow \lfloor \alpha_1 \times n \rfloor$
10: $A \leftarrow A^*$ /* Critério de aceitação para intensificação */
11: $\hat{S} \leftarrow Perturbação\ (A, n, I_1, P)$

/* thread 1 – extremo f_1^{best} */	/* thread 2 – extremo f_2^{best} */
12: $A^k \leftarrow BuscaLocal\ (\hat{S}\ , t_{min}, t_{max}, z_1)$	$B^k \leftarrow BuscaLocal\ (\hat{S}\ , t_{min}, t_{max}, z_2)$
13: $\alpha_1 \leftarrow \alpha_1 \times \frac{f_1(A^k)}{f_1(A^*)}$	
14: **Se** $\alpha_1 < \alpha_{min}$ **OU** $\alpha_1 < \alpha_{max}$ **então**	
15: $\alpha_1 \leftarrow \alpha_{min}$	
16: **Fim**	
17: **Se** $Avaliar\_soluções(z_1(A^k), z_1(A^*)) = 0$ **então**	**Se** $Avaliar\_soluções(z_2(B^k), z_2(B^*)) = 0$ **então**
18: $A^* \leftarrow A^k$ /* Atualiza melhor no extremo (f_1^{best}, f_2) */	$B^* \leftarrow B^k$ /* Atualiza a melhor no extremo (f_1 , f_2^{best}) */
19: $iter\_melhora \leftarrow k$	
20: **Fim**	**Fim**
21: **Fim**	

/* Fase 2 */

22: $iter\_diver \leftarrow 0$
23: **Enquanto** $iter\_diver < nIter$ **E** $tempo\_decorrido \leq T_{Lim}$ **faça**
24: $A \leftarrow A^k$ /* Critério de aceitação para diversificação */
25: $iter\_diver \leftarrow iter\_diver + 1$
26: $k \leftarrow k + 1$
27: $I_2 \leftarrow \lfloor \alpha_2 \times n \rfloor$
28: $\hat{S} \leftarrow Perturbação\ (A, n, I_2, P)$

/* thread 1 – extremo f_1^{best} */	/* thread 2 – extremo f_2^{best} */
29: $A^k \leftarrow BuscaLocal\ (\hat{S}, t_{min}, t_{max}, z_1)$	$B^k \leftarrow BuscaLocal\ (\hat{S}, t_{min}, t_{max}, z_2)$
30: **Se** $Avaliar\_soluções(z_1(A^k), z_1(A^*)) = 0$ **então**	**Se** $Avaliar\_soluções(z_2(B^k), z_2(B^*)) = 0$ **então**
31: $A^* \leftarrow A^k$ /* Atualiza melhor no extremo (f_1^{best}, f_2) */	$B^* \leftarrow B^k$ /* Atualiza a melhor no extremo (f_1 , f_2^{best}) */
32: **Fim**	**Fim**
33: **Fim**	
34: **Fim**	
35: $C \leftarrow A^*$, $D \leftarrow B^*$	

/* thread 1 – extremo f_1^{best} */	/* thread 2 – extremo f_2^{best} */
36: **Se** $Avaliar\_soluções(z_1(D), z_1(A^*)) = 0$ **então**	**Se** $Avaliar\_soluções(z_2(C), z_2(B^*)) = 0$ **então**
37: $A^*, \leftarrow D$ /* Atualiza a melhor no extremo (f_1^{best}, f_2) */	$B^* \leftarrow C$ /* Atualiza a melhor no extremo (f_1 , f_2^{best}) */
38: **Fim**	**Fim**

39: **retorne** A^*, e B^*

Na Linha 1 do Algoritmo 22, o valor inicial do número (*t*) de facilidades na primeira linha é definido, o contador *k* e os fatores para determinar a intensidade da perturbação α_1 e α_2 são

inicializados, bem como as funções z_1, para avaliação das soluções no extremo relativo ao melhor custo total de manuseio de materiais entre as facilidades, f_1^{best} dado pela f_1 (Equação (2.5)) e z_2, para a avaliação das soluções no extremo relativo ao melhor desvio percentual do comprimento do corredor, f_2^{best} dado pela f_2 (Equação (2.6)). Em seguida, na Linha 2, a solução inicial A^0 é obtida pelo procedimento *GerarSoluçãoInicialCAP*.

Na Linha 3, uma solução ótima local A^k é obtida com a aplicação da busca local considerando a prioridade definida em z_1. Na Linha 4, a melhor solução A^* é inicializada com a solução A^k. Do mesmo modo, e, paralelamente, as soluções B^k e B^* são encontradas considerando a função z_2.

O processo iterativo do AILS-bCAP ocorre no laço das Linhas 5-34. O AILS-bCAP termina após ter executado um limite de tempo T_{Lim}. Dentro do laço principal ocorrem as Fases 1 e 2 intercaladas. A Fase 1 começa inicializando a variável *iter_melhora* (Linha 6), que indica a iteração em que ocorreu a última melhora na solução A^* e itera no laço das Linhas 7-21 enquanto o número de iterações sem melhora for menor que *nIterWI* e o tempo decorrido não ultrapasse o limite T_{Lim}. Na Linha 8, a variável k é incrementada e na Linha 9 a intensidade da perturbação I_1 é calculada. Na Linha 10, a solução corrente A é atualizada com a melhor solução A^*, que é critério de aceitação considerando o aspecto de intensificação no AILS-bCAP, como definido na Seção 7.6. Na Linha 11, a solução $\hat{S}$ recebe a solução retornada pela perturbação da solução A. A partir da Linha 12, o algoritmo aplica paralelamente o procedimento de busca local à solução $\hat{S}$. Seguindo o *thread* 1 de execução, o algoritmo considera a prioridade definida pela função z_1. Na Linha 13, α_1 é atualizado conforme Equação (7.2) e nas Linhas 14 a 16 é verificado se α_1 está no limite [α_{min}, α_{max}]. Caso esteja fora do intervalo, o valor de α_1 é atualizado para α_{min}. Na Linha 17, a nova solução ótima local A^k é comparada com a melhor solução A^* utilizando o procedimento para avaliar as soluções (Seção 7.3), considerando o critério de priorização definido em z_1. Caso a comparação retorne 0, a melhor solução atual A^* é atualizada na Linha 18 e a variável *iter_melhora* é atualizada na Linha 19. Paralelamente, no *thread* 2, o algoritmo aplica a busca local considerando a priorização definida na função z_2 (Linha 12). Em seguida a solução B^k retornada pela busca local é comparada à melhor solução B^* (Linha 17). Caso seja melhor, a solução B^* é atualizada na Linha 18. Ao terminar a execução das buscas locais, o algoritmo retorna a um único *thread* de execução (Linha 21). O número de iterações da Fase 1 é adaptativo, sendo redefinido a cada iteração (Linha 13).

A Fase 2 do algoritmo inicia na Linha 22, com a variável *iter_diver* sendo inicializada com zero. O laço das Linhas 23-33 executa a Fase 2, que termina após *nIter* iterações ou pelo tempo limite T_{Lim}. Na Linha 24, a solução corrente A é atualizada com a última solução ótima local A^k, sendo o critério de aceitação o que busca uma maior diversificação no AILS-bCAP, como definido na Seção 7.6. Nas Linhas 25 e 26, as variáveis *iter_diver* e k são incrementadas, respectivamente. Na Linha 27, a intensidade da perturbação I_2 é calculada. Na Linha 28, uma nova solução $\hat{S}$ é obtida com a perturbação da solução A. Novamente, de forma paralela, o algoritmo aplica as buscas locais à solução $\hat{S}$, que leva aos ótimos locais, A^k e B^k, de acordo com o critério considerado em cada *thread* de execução (Linha 29). Na Linha 30, a solução ótima local A^k é comparada à melhor solução A^* (*thread* 1) e a solução B^k é comparada com a melhor solução B^* (*thread* 2). Caso as soluções melhores sejam encontradas, A^* e B^* são atualizadas em suas respectivas *threads* de execuções.

Como o AILS-bCAP encontra duas soluções em um processo estocástico e pela implementação proposta não há troca de informações entre as melhores soluções, A^* e B^*, durante o processo de busca, ao final pode ocorrer da solução A^* ser melhor que a solução B^* ao se considerar a função z_2, ou a solução B^* ser melhor que a A^* ao se considerar a função z_1. Assim, na Linha 35, as soluções C e D são inicializadas e, na Linha 36, a solução D é comparada com a solução A^*. Se D for melhor, A^* é atualizada (Linha 37). O mesmo é realizado em relação às soluções C e B^*. Ao final, Linha 39, o AILS-bCAP retorna A^*, a melhor solução encontrada no extremo (f_1^{best}, f_2), e B^*, a melhor solução no extremo (f_1, f_2^{best}).

O paralelismo na execução das *threads* 1 e 2 do AILS-bCAP, exemplificado em destaque no pseudocódigo, é realizado utilizando a interface de programação de aplicações OpenMP (do inglês, *Open Multi-Processing*), disponível em *https://www.openmp.org*, que oferece suporte à programação paralela em memória compartilhada em várias plataformas, nas linguagens de programação C/C++ e Fortran.

7.8 EXPERIMENTOS COMPUTACIONAIS

O AILS-bCAP proposto para resolver o bCAP foi implementado em linguagem C usando o compilador GCC 5.1.0 x64 com as flags de otimização -O3 e -march em um computador com processador Intel i7-7700k 4,2 GHz com 16GB de memória e sistema operacional MS Windows

10. Os experimentos computacionais foram realizados com 76 instâncias utilizadas na literatura, sendo:

- 20 instâncias, inicialmente utilizadas para o SRFLP por Anjos, Kennings e Vannelli (2005) agrupadas em quatro grupos contendo cinco instâncias de tamanhos n = {60, 70, 75, 80} facilidades e usadas por Kalita e Datta (2014) para o bCAP;
- 36 instâncias, inicialmente propostas por Ahonen, De Alvarenga e Amaral (2014) para o CAP, com tamanhos n = 60 facilidades e usada por Kalita e Datta (2014) para o bCAP; e
- 20 instâncias, introduzidas por Palubeckis (2015) para o SRFLP com tamanhos $110 \leq n \leq 300$ facilidades e usadas por Kalita, Datta e Palubeckis (2019) para o bCAP.

7.8.1 Calibração dos parâmetros

No AILS-bCAP, o parâmetro n é dado pelo tamanho da instância e T_{Lim} é informado de acordo com o tempo de execução desejado. Além disso, os valores de $t_{\min}$ e t_{max} foram fixados em $t_{min} = \lfloor \frac{n}{2} \rfloor$ e $t_{max} = \lfloor \frac{n}{2} \rfloor + 4$ (AMARAL, 2020; CRAVO; BISSOLI; AMARAL, 2021). Para determinar os outros parâmetros, utilizou-se o PSO de calibração (APÊNDICE B) com a avaliação das soluções geradas, considerando o melhor custo total de manuseio de materiais.

Cada partícula no PSO contém um vetor x_i onde cada componente do vetor é um parâmetro do AILS-bCAP, sendo limitados no PSO como $1 \leq x_{i1} \leq N$ (*nIterWI*), $1 \leq x_{i2} \leq N$ (*nIter*), $0{,}01 \leq x_{i3} \leq 0{,}9$ (α_{min}), $\alpha_{min} \leq x_{i4} \leq 0{,}9$ (α_{max}) e $0 \leq x_{i5} \leq 0{,}9$ (α).

Para avaliar o efeito do procedimento de perturbação, foram consideradas variantes do AILS-bCAP com valores do parâmetro P = 0,5 (AILS-bCAP0), P = 1 (AILS-bCAP1); e P = 0 (AILS-bCAP2).

O PSO foi executado com os limites N = {100, 150, 200, 250} e com m = 10 partículas, *IterMax* = 36 e o tempo limite de 10 segundos (T_{Lim}) para avaliar cada partícula com o AILS-bCAP. As instâncias utilizadas na calibração foram n60p30s30-1, n60p60s30-1, n60p60s50-1 e n60p90s30-1 (AHONEN; DE ALVARENGA; AMARAL, 2014). Assim, para cada par (N, *instância*), o PSO encontrou um conjunto de parâmetros, totalizando 16 conjuntos de parâmetros para cada versão do AILS-bCAP.

Por fim, para definir um único conjunto de parâmetros de cada versão do AILS-bCAP, os algoritmos foram executados 10 vezes para cada uma das instâncias da calibração, com um limite de tempo T_{Lim} = 100 segundos. O conjunto de parâmetros com a melhor média geral da função objetivo f_1 foi selecionado para cada versão do AILS-bCAP e é mostrado na Tabela 29.

Tabela 29 – Parâmetro encontrados para as variantes do AILS-bCAP

Versão do AILS-bCAP	*nIterWI*	*nIter*	α_{min}	α_{max},	α
AILS-bCAP0	175	147	0,0189	0,2047	0,6460
AILS-bCAP1	126	233	0,1224	0,1242	0,3542
AILS-bCAP2	76	73	0,1059	0,1377	0,8798

7.8.2 Análise das variantes do AILS-bCAP

Nesta subseção são avaliadas as principais componentes do AILS-bCAP utilizando quatro instâncias de tamanhos n = {150, 200, 250, 300} (PALUBECKIS, 2015). Nos experimentos, cada algoritmo foi executado 30 vezes com um tempo limite de 3600 segundos para a instância com $n = 300$ e, para as outras instâncias, o tempo foi proporcional ao tamanho n, assim, $T_{Lim} = n \times \frac{3600}{300}$.

Inicialmente foram avaliadas as versões AILS-bCAP0, AILS-bCAP1 e AILS-bCAP2, as quais diferem entre si pelo uso das rotinas do procedimento de perturbação (Seção 7.5). A Tabela 30 apresenta os resultados encontrados e está estruturada da seguinte forma: a primeira coluna apresenta os nomes das instâncias; a segunda coluna, os tamanhos das instâncias; a coluna f_1^* apresenta os melhores valores conhecidos para f_1 obtido pelo pGA de Kalita, Datta e Palubeckis (2019); e para cada versão do AILS-bCAP são apresentados o valor $D_{f_1^{best}}$, diferença entre o melhor valor de f_1 e o melhor valor conhecido f_1^*; o valor $D_{f_1^{avg}}$, diferença entre as médias dos valores f_1 e f_1^*; e a média do número de iterações. Os valores destacados em negrito mostram as melhores diferenças encontradas. As médias dos valores da função objetivo f_2 não são apresentadas, pois os algoritmos obtiveram os mesmos valores para todas as instâncias. A última linha da tabela apresenta a média sobre todas as colunas.

Tabela 30 – Comparação dos resultados das variantes do AILS-bCAP

Instância	n	pGA	AILS-bCAP0			AILS-bCAP1			AILS-bCAP2		
		f_1^{*} [a]	$D_{f_1^{best}}$	$D_{f_1^{avg}}$	$Iter_{avg}$	$D_{f_1^{best}}$	$D_{f_1^{avg}}$	$Iter_{avg}$	$D_{f_1^{best}}$	$D_{f_1^{avg}}$	$Iter_{avg}$
p150	150	5317335,5	-4743,0	-3711,3	760	**-4866,0**	-4538,8	1246	-4535,0	-3203,5	841
p200	200	13778294,5	-36235,0	-31371,3	272	**-36868,0**	-32047,8	506	-35615,0	-30120,7	303
p250	250	27357813,5	-89390,0	-79588,1	125	**-93982,0**	-76903,0	256	-86068,0	-77070,7	140
p300	300	48107893,0	-133007,0	-111131,5	59	**-138243,0**	-106081,4	137	-131961,0	-106558,3	65
Média		23640334,1	-65843,7	-56450,5	304	-68489,7	-54892,7	536	-64544,7	-54238,3	337

[a] O melhor valor conhecido foi obtido por (KALITA; DATTA; PALUBECKIS, 2019).
Os melhores valores são destacados em negrito.

Como pode ser visto na Tabela 30, todas as versões obtiveram valores médios melhores que os melhores valores conhecidos. Entretanto, as melhores soluções foram obtidas pela versão do AILS-bCAP1 para todas as instâncias utilizadas. Avaliando o número médio de iterações, nota-se que o AILS-bCAP1 consegue, no mesmo limite de tempo, executar uma quantidade de iterações maior que as outras versões, o que sugere um esforço computacional menor da perturbação utilizada no AILS-bCAP1, já que as outras componentes do algoritmo permaneceram iguais em todas as versões, o que proporcionou uma exploração melhor do espaço de soluções, possibilitando ao AILS-bCAP1 encontrar as melhores soluções para esse conjunto de instâncias.

Considerando a versão AILS-bCAP1, com $P = 1$, serão analisadas as versões alternativas do procedimento de busca local, a fim de determinar qual apresenta o melhor desempenho, tanto em relação a soluções encontradas quanto ao número de iterações realizadas. Assim, o Quadro 2 apresentada as versões alternativas propostas para a avaliação do AILS-bCAP1, com os nomes das versões na primeira coluna e na segunda coluna as descrições das alterações sugeridas.

Quadro 2 – Versões do AILS-bCAP com buscas locais alternativas

Variantes	Característica do procedimento de Busca Local
AILS-bCAP1-Alt1	Rotina *Busca_na_linha* utilizando movimento 2-opt
AILS-bCAP1-Alt2	Rotina *Busca_na_linha* utilizando o movimento de inserção sem a técnica para acelerar o cálculo do ganho na função objetivo
AILS-bCAP1-Alt3	Busca local sem a rotina *Busca_entre_linhas*

Fonte: próprio autor.

Os experimentos realizados com as três variantes do AILS-bCAP1, considerando a função objetivo f_1, melhor custo do manuseio de materiais, são mostrados na Tabela 31. A primeira coluna da tabela mostra os nomes das instâncias; a segunda coluna, os nomes dos algoritmos

avaliados; a terceira coluna apresenta as médias dos valores da função objetivo f_1 das soluções encontradas; na quarta coluna têm-se as médias dos tempos efetivos de execução; e, por fim, as médias dos números de iterações. As melhores médias são destacadas em negrito.

As médias dos valores da função objetivo f_2 novamente não foram analisadas, pois os valores obtidos pelas variantes do AILS-bCAP1 foram iguais para todas as instâncias.

Tabela 31 – Comparação dos resultados do AILS-bCAP1 com as buscas locais alternativas

Instância	Algoritmo	f_1^{avg}	T_{avg}	$Iter_{avg}$
p150	AILS-bCAP1	**5312796,7**	1802	1246
	AILS-bCAP1-Alt1	5313167,6	1801	1154
	AILS-bCAP1-Alt2	5313336,1	1803	626
	AILS-bCAP1-Alt3	5322682,0	1802	1183
p200	AILS-bCAP1	**13746246,7**	2406	506
	AILS-bCAP1-Alt1	13747050,6	2404	475
	AILS-bCAP1-Alt2	13752863,4	2414	251
	AILS-bCAP1-Alt3	13765118,3	2407	442
p250	AILS-bCAP1	**27280910,5**	3014	256
	AILS-bCAP1-Alt1	27282482,8	3019	246
	AILS-bCAP1-Alt2	27283864,1	3042	124
	AILS-bCAP1-Alt3	27307658,6	3026	216
p300	AILS-bCAP1	**48001811,6**	3643	137
	AILS-bCAP1-Alt1	48003361,3	3645	133
	AILS-bCAP1-Alt2	48016387,7	3691	66
	AILS-bCAP1-Alt3	48053644,1	3661	107

Os resultados encontrados mostram a versão do AILS-bCAP1 padrão com um desempenho melhor para esse conjunto de instâncias em comparação com variantes AILS-bCAP1-Alt1, AILS-bCAP1-Alt2 e AILS-bCAP1-Alt3, tendo as melhores médias e o maior número médio de iterações nas 30 execuções. Como pode ser visto, o uso da rotina *Busca_na_linha*, sem a técnica para acelerar o cálculo do ganho na função objetivo, reduziu o número de iterações da versão AILS-bCAP1-Alt2.

Pelos resultados expostos, o AILS-bCAP1 será o método escolhido para os experimentos computacionais subsequentes, de modo que o desempenho do mesmo será avaliado em comparação com pGA de Kalita, Datta e Palubeckis (2019).

7.8.3 Critério de avaliação do desempenho

Para a avaliação do desempenho do AILS-bCAP, as soluções obtidas serão comparadas com as melhores soluções da literatura. O AILS-bCAP1 foi executado 30 vezes para cada instância com um tempo limite de execução T_{Lim} que depende do tamanho da instância. As soluções

obtidas pelo AILS-bCAP no extremo (f_1^{best},f_2), são dadas pelo vetor z_1=(f_1^{best},f_2), e, no extremo (f_1,f_2^{best}), pelo vetor z_2=(f_1,f_2^{best}). Assim, dado o vetor objetivo v_1=(f_1^*, f_2) com a melhor solução conhecida no extremo (f_1^{best},f_2), são calculadas as diferenças ($D_{f_1^{best}}$,D_{f_2})=z_1-v_1, e, para o extremo (f_1,f_2^{best}), dada a melhor solução da literatura como sendo o vetor objetivo v_2=(f_1,f_2^*), são calculadas as diferenças (D_{f_1},$D_{f_2^{best}}$)=z_2-v_2. Os melhores valores conhecidos na literatura são do algoritmo pGA (KALITA; DATTA; PALUBECKIS, 2019). Os autores apresentam os tempos computacionais apenas para as instâncias com tamanhos $110 \leq n \leq 300$, entretanto, pela ausência de informações da especificação do computador utilizado, não foi possível realizar os ajustes dos tempos de execução para o processador utilizado neste trabalho. Desse modo, foi definido um tempo limite de execução de 3600 segundos para a instância de tamanho n = 300, bem inferior aos tempos do pGA, e para as outras instâncias, os tempos limites foram definidos conforme expressão $T_{Lim} = n \times \frac{3600}{300}$.

Para observar a consistência na convergência e a robustez dos algoritmos nos diversos testes realizados, são calculadas duas medidas de desempenho baseadas nas médias dos valores de f_1 (f_1^{avg}), e nos desvios-padrão de f_1 (DP) das soluções encontradas, como segue:

- *Desv*, desvio percentual da média f_1^{avg} em relação ao valor da melhor solução conhecida f_1^*, dado por $Desv = 100 \times \frac{f_1^{avg} - f_1^*}{f_1^*}$; e
- *DPr*, desvio-padrão relativo, calculado como $DPr = 100 \times \frac{DP}{f_1^{avg}}$.

7.8.4 Avaliação dos resultados para o bCAP para as instâncias Anjos, Kennings e Vannelli (2005)

Os resultados dos experimentos com o AILS-bCAP1 para as instâncias com tamanhos n = {60, 70, 75, 80} (ANJOS; KENNINGS; VANNELLI, 2005) são apresentados na Tabela 32 e comparados com os melhores valores conhecidos na literatura. A Tabela 32 está estruturada seguinte forma: a primeira coluna mostra os nomes das instâncias; a segunda coluna, os tamanhos da instâncias; para as soluções no extremo (f_1^{best},f_2), tem-se a solução de referência da literatura (f_1^*, f_2) seguida das diferenças ($D_{f_1^{best}}$,D_{f_2}) do AILS-bCAP1; e, para o extremo (f_1,f_2^{best}),

são mostrados os melhores valores da literatura (f_1, f_2^*) seguidos das diferenças (D_{f_1}, $D_{f_2^{best}}$) do AILS-bCAP1; e, na última coluna, T_{avg}, são mostradas as médias dos tempos de execução, em segundos, para cada instância. A última linha da tabela apresenta a média sobre todas as colunas da tabela. Os valores destacados em negrito representam os melhores valores obtidos em comparação aos melhores valores conhecidos para a função objetivo f_1.

Tabela 32 – Comparação dos resultados do bCAP para instâncias com tamanhos $60 \leq n \leq 80$ facilidades

Instância	n	Melhor custo do manuseio de materiais				Melhor comprimento do corredor				T_{avg}
		pGA[a]		AILS-bCAP1		pGA[a]		AILS-bCAP1		
		f_1^*	f_2	$D_{f_1^{best}}$	D_{f_2}	f_1	f_2^*	D_{f_1}	$D_{f_2^{best}}$	
AKV60-1	60	739516,0	**0,10**	**-635,0**	0,42	739582,0	**0**	**-682,0**	**0**	720
AKV60-2	60	421074,0	1,87	**-166,0**	**-0,21**	421419,0	**0,05**	**-226,0**	**0**	720
AKV60-3	60	324931,5	1,88	**-732,0**	**-0,22**	325222,5	**0,05**	**-859,0**	**0**	720
AKV60-4	60	199434,0	**0,09**	**-304,0**	1,83	199441,0	**0**	**-260,0**	**0**	720
AKV60-5	60	159652,0	**1,39**	**-10,0**	0,93	159870,0	**0,07**	**-46,0**	**0**	720
AKV70-1	70	765123,0	1,17	**-676,0**	**-0,07**	765711,0	**0,04**	**-1039,0**	**0**	840
AKV70-2	70	721156,0	0,97	**-436,0**	**-0,62**	721251,0	**0**	**-488,0**	**0**	840
AKV70-3	70	760263,5	0,60	**-837,0**	**-0,60**	760469,5	**0**	**-1043,0**	**0**	840
AKV70-4	70	484746,0	**0,26**	**-412,0**	0,21	484848,0	**0,04**	**-493,0**	**0**	840
AKV70-5	70	2109741,5	0,78	**-17,0**	**-0,07**	2109845,5	**0**	**-18,0**	**0**	840
AKV75-1	75	1197910,5	**0,13**	**-881,0**	0,87	1198083,5	**0**	**-893,0**	**0**	900
AKV75-2	75	2162182,0	0,53	**-1394,0**	**-0,12**	2162581,0	**0,03**	**-1645,0**	**0**	900
AKV75-3	75	625118,0	0,99	**-709,0**	**-0,35**	625317,0	**0**	**-768,0**	**0**	900
AKV75-4	75	1972587,5	**0,41**	**-1248,0**	0,19	1972897,5	**0,03**	**-1187,0**	**0**	900
AKV75-5	75	896016,0	**0,45**	**-177,0**	0,26	896127,0	**0**	**-136,0**	**0**	900
AKV80-1	80	1035134,5	**0,12**	**-585,0**	0,19	1035150,5	**0**	**-552,0**	**0**	960
AKV80-2	80	961338,0	**0,09**	**-633,0**	0,57	961420,0	**0,03**	**-633,0**	**0**	960
AKV80-3	80	1626934,0	1,23	**-875,0**	**-0,43**	1627508,0	**0**	**-1148,0**	**0**	960
AKV80-4	80	1874656,0	**0,57**	**-685,0**	0,72	1875113,0	**0,03**	**-785,0**	**0**	960
AKV80-5	80	795089,0	**0,03**	**-792,0**	0,87	795089,0	**0,03**	**-684,0**	**0**	960
Média		991630,2	0,68	-610,2	0,22	991847,3	0,02	-679,3	0	855

[a] Os melhores valores conhecidos obtidos por (KALITA; DATTA; PALUBECKIS, 2019).
Os melhores valores são destacados em negrito.

Os resultados apresentados na Tabela 32 mostram que o algoritmo proposto obteve excelentes resultados para esse grupo de instâncias, melhorando os melhores valores conhecidos para a função objetivo f_1 para as 20 instâncias.

Considerados a relação de dominância entre soluções de problemas multiobjetivos (Seção 7.1), os resultados em que as soluções encontradas dominam as melhores soluções da literatura, o algoritmo AILS-bCAP1 encontrou soluções que dominam em 9 instâncias (AKV60-2, AKV60-3, AKV70-1, AKV70-2, AKV70-3, AKV70-5, AKV75-2, AKV75-3 e AKV80-3), como mostrado na Tabela 32 com os valores das diferenças ($D_{f_1^{best}}, D_{f_2}$) negativos ou zero e nas 11 instâncias restantes, nenhuma solução do AILS-bCAP1 foi dominada pela solução da literatura.

Em relação às soluções do AILS-bCAP1, considerando o melhor desvio percentual do comprimento do corredor, as melhores soluções foram obtidas para todas as 20 instâncias. Também pode-se destacar que o algoritmo AILS-bCAP1 encontrou soluções que dominam as melhores soluções da literatura para todas as instâncias, como mostrado na Tabela 32 com os valores das diferenças (D_{f_1}, $D_{f_2^{best}}$) negativos ou zero.

A Tabela 33 apresenta as medidas de desempenho dos algoritmos para esse conjunto de testes. Os valores são calculados a partir das soluções médias e do desvio-padrão para cada instância no extremo (f_1^{best},f_2). Para esse conjunto de teste, as medidas de desempenho do pGA (KALITA; DATTA; PALUBECKIS, 2019) não estão disponíveis. A primeira coluna da tabela apresenta os nomes das instâncias; a segunda coluna, os tamanhos das instâncias; a terceira coluna, os valores dos desvios percentuais da média *Desv* e, por fim, os valores do desvios-padrão relativos (*DPr*). A última linha mostra as médias sobre todas as colunas da tabela.

Tabela 33 – Desempenho do AILS-bCAP1 no extremo (f_1^{best}, f_2) para instâncias com tamanhos $60 \leq n \leq 80$ facilidades

Instância	*n*	AILS-bCAP1	
		Desv	*DPr*
AKV60-1	60	-0,068	0,009
AKV60-2	60	-0,004	0,030
AKV60-3	60	-0,173	0,020
AKV60-4	60	-0,125	0,014
AKV60-5	60	0,029	0,021
AKV70-1	70	-0,063	0,008
AKV70-2	70	-0,051	0,010
AKV70-3	70	-0,106	0,003
AKV70-4	70	-0,076	0,004
AKV70-5	70	0,006	0,004
AKV75-1	75	-0,056	0,008
AKV75-2	75	-0,049	0,013
AKV75-3	75	-0,087	0,010
AKV75-4	75	-0,056	0,005
AKV75-5	75	-0,011	0,005
AKV80-1	80	-0,050	0,004
AKV80-2	80	-0,057	0,005
AKV80-3	80	-0,045	0,006
AKV80-4	80	-0,029	0,006
AKV80-5	80	-0,081	0,010
Média		-0,048	0,008

Como mostrado na Tabela 33, o algoritmo proposto apresenta um desempenho robusto, mantendo os valores baixos dos desvios percentuais das médias mesmo com o aumento do tamanho das instâncias, sendo os únicos valores não negativos os obtidos para as instâncias AKV60-5 e AKV70-5. Ainda, pelos baixos valores dos desvios-padrão relativos, os algoritmos apresentam convergências consistentes para esse conjunto de instâncias. Para 18 das 20

instâncias, o AILS-bCAP1 obteve os valores médios das soluções melhores que as soluções de referência.

7.8.5 Avaliação dos resultados para o bCAP para as instâncias Ahonen, De Alvarenga e Amaral (2014)

Os resultados dos experimentos com o AILS-bCAP1 para as instâncias com tamanhos n = 60 facilidades (AHONEN; DE ALVARENGA; AMARAL, 2014) são apresentados e comparados na Tabela 34 com os melhores valores conhecidos encontrados pelo algoritmo pGA (KALITA; DATTA; PALUBECKIS, 2019). A tabela está estruturada como a Tabela 32. Os valores destacados em negrito são os melhoramentos obtidos em relação aos valores da literatura para o valor de f_1.

Novamente, pelos resultados apresentados na Tabela 34, o AILS-bCAP1 é capaz de encontrar resultados competitivos, melhorando os melhores valores conhecidos para a função objetivo f_1, de 17 das 36 instâncias. Os mesmos valores da literatura foram obtidos para 13 instâncias e em apenas 6 das 36 instâncias (n60p30s60-3, n60p60s30-1, n60p60s50-1, n60p60s50-2, n60p90s40-1 e n60p90s50-3) os valores encontrados são ligeiramente piores que os da literatura.

Considerando os resultados em que as soluções encontradas dominam as melhores soluções da literatura, o AILS-bCAP1 apresentou soluções melhores para 12 instâncias, como mostrado na Tabela 34 pelos valores negativos ou zero dos pares ($D_{f_1^{best}}, D_{f_2}$); os mesmos valores da literatura foram obtidos para 13 instâncias; soluções indiferentes em relação às soluções da literatura para 6 instâncias, ou seja, soluções onde um objetivo é melhor, mas o outro é pior; e, nas 5 instâncias restantes (n60p30s60-3, n60p60s30-1, n60p60s50-2, n60p90s40-1 e n60p90s50-3), as soluções do AILS-bCAP1 são dominadas pela solução de referência.

Ao considerar o comprimento do corredor, o AILS-bCAP1 encontrou soluções com os mesmos valores para a função objetivo f_2 para todas as instâncias. Considerando a relação de dominância das soluções, foram obtidas soluções dominantes para 26 das 36 instâncias, como mostram os valores negativos ou zero dos pares (D_{f_1}, $D_{f_2^{best}}$), e os mesmos valores de referência foram encontrados para 4 instâncias (n60p30s40-1, n60p60s60-1, n60p90s40-3 e n60p90s60-3). O algoritmo proposto não foi bem sucedido nas instâncias n60p30s30-1, n60p30s30-2,

n60p30s50-1, n60p60s40-1, n60p90s30-3 e n60p90s40-2, tendo obtido soluções que são dominadas pelas soluções da literatura.

Tabela 34 - Comparação dos resultados do bCAP para instâncias de Ahonen, De Alvarenga e Amaral (2014)

Instância	n	Melhor custo do manuseio de materiais				Melhor comprimento do corredor				T_{avg}
		pGA[(a)]		AILS-bCAP1		pGA[(a)]		AILS-bCAP1		
		f_1^*	f_2	$D_{f_1^{best}}$	D_{f_2}	f_1	f_2^*	D_{f_1}	$D_{f_2^{best}}$	
n60p30s30-1	60	204099,0	**0,78**	**-7,0**	0,62	**204237,0**	**0,16**	4,0	**0**	720
n60p30s30-2	60	193212,5	**1,26**	**-35,0**	**0**	**193322,5**	**0**	22,0	**0**	720
n60p30s30-3	60	161475,5	**0,50**	**-1,0**	**0**	161496,5	**0,17**	**-18,0**	**0**	720
n60p30s40-1	60	**135133,5**	**0,20**	**0**	**0**	**135133,5**	**0,20**	**0**	**0**	720
n60p30s40-2	60	**159114,0**	**0,39**	**0**	**0**	159131,0	**0**	**-6,0**	**0**	720
n60p30s40-3	60	158973,5	**0,00**	**-34,0**	0,38	158973,5	**0**	**-15,0**	**0**	720
n60p30s50-1	60	110493,5	**2,11**	**-6,0**	**0**	**110657,5**	**0,23**	105,0	**0**	720
n60p30s50-2	60	115319,0	**2,51**	**-26,0**	**0**	115440,0	**0,23**	**-42,0**	**0**	720
n60p30s50-3	60	114157,0	**0,66**	**-22,0**	0,44	114199,0	**0,22**	**-46,0**	**0**	720
n60p30s60-1	60	108123,0	**0,31**	**-10,0**	**0**	108123,0	**0,31**	**-10,0**	**0**	720
n60p30s60-2	60	109937,5	**0,55**	**-25,0**	0,55	109939,5	**0**	**-10,0**	**0**	720
n60p30s60-3	60	**91619,0**	**0,66**	5,0	**0**	91779,0	**0**	**-76,0**	**0**	720
n60p60s30-1	60	**445377,5**	**1,68**	1,0	**0**	445543,5	**0,15**	**-48,0**	**0**	720
n60p60s30-2	60	407891,5	**0,98**	**-5,0**	**0**	407964,5	**0**	**-31,0**	**0**	720
n60p60s30-3	60	416930,5	**0,33**	**-5,0**	0,33	417058,5	**0**	**-71,0**	**0**	720
n60p60s40-1	60	313320,0	**0,81**	**-14,0**	**0**	**313466,0**	**0**	14,0	**0**	720
n60p60s40-2	60	**320758,5**	**0,79**	**0**	**0**	320832,5	**0**	**-15,0**	**0**	720
n60p60s40-3	60	**363020,5**	**0,53**	**0**	**0**	363097,5	**0,18**	**-10,0**	**0**	720
n60p60s50-1	60	**273420,0**	0,71	6,0	**-0,47**	273470,0	**0,24**	**-44,0**	**0**	720
n60p60s50-2	60	**269680,5**	**0,97**	2,0	**0**	269747,5	**0**	**-6,0**	**0**	720
n60p60s50-3	60	295423,0	**0,72**	**-9,0**	**0**	295482,0	**0,24**	**-46,0**	**0**	720
n60p60s60-1	60	**227896,0**	**0,00**	**0**	**0**	**227896,0**	**0**	**0**	**0**	720
n60p60s60-2	60	**246523,0**	**0,56**	**0**	**0**	246548,0	**0**	**-20,0**	**0**	720
n60p60s60-3	60	206493,5	**0,66**	**-11,0**	**0**	206519,5	**0**	**-11,0**	**0**	720
n60p90s30-1	60	628859,0	**0,49**	**-9,0**	**0**	628930,0	**0,16**	**-51,0**	**0**	720
n60p90s30-2	60	**561170,5**	**0,36**	**0**	**0**	561214,5	**0**	**-13,0**	**0**	720
n60p90s30-3	60	587821,5	**0,34**	**-5,0**	**0**	**587842,5**	**0**	2,0	**0**	720
n60p90s40-1	60	**474046,0**	**0,41**	7,0	**0**	474065,0	**0**	**-3,0**	**0**	720
n60p90s40-2	60	**479963,0**	**0,81**	**0**	**0**	**479999,0**	**0**	1,0	**0**	720
n60p90s40-3	60	**512452,0**	**0,19**	**0**	**0**	**512452,0**	**0,19**	**0**	**0**	720
n60p90s50-1	60	479692,0	**0,44**	**-9,0**	**0**	479710,0	**0**	**-9,0**	**0**	720
n60p90s50-2	60	**445030,0**	**0,71**	**0**	**0**	445074,0	**0,24**	**-33,0**	**0**	720
n60p90s50-3	60	**495053,5**	**1,96**	1,0	**0**	495519,5	**0,22**	**-23,0**	**0**	720
n60p90s60-1	60	**385430,5**	**0,58**	**0**	**0**	385454,5	**0**	**-3,0**	**0**	720
n60p90s60-2	60	**344775,0**	**0,62**	**0**	**0**	344790,0	**0**	**-3,0**	**0**	720
n60p90s60-3	60	**411205,0**	**0,00**	**0**	**0**	**411205,0**	**0**	**0**	**0**	720
Média		312608,0	0,70	-5,9	0,05	312675,4	0,09	-14,3	0	720

[(a)] Os melhores valores conhecidos obtidos por (KALITA; DATTA; PALUBECKIS, 2019).
Os melhores valores são destacados em negrito.

A Tabela 35 apresenta as medidas de desempenho dos algoritmos para esse conjunto de instâncias, calculadas com os valores das soluções médias e dos desvios-padrão de f_1 no extremo (f_1^{best}, f_2). A primeira coluna da tabela apresenta os nomes das instâncias; a segunda coluna, os tamanhos das instâncias; na terceira coluna são mostradas as médias das soluções, seguida dos desvios percentuais das médias (*Desv*) e dos desvios-padrão relativos (*DPr*) das soluções de referência (KALITA; DATTA; PALUBECKIS, 2019); e, para o AILS-bCAP1,

têm-se os valores médios das soluções, os desvios percentuais das médias (*Desv*) e os valores do desvios-padrão relativos (*DPr*). A última linha mostra as médias sobre todas as colunas da tabela. As melhores médias dos valores de f_1 estão destacadas em negrito.

Pelo exposto na Tabela 35, o algoritmo proposto apresenta um desempenho robusto mantendo valores baixos dos desvios percentuais das médias e as melhores médias das soluções para 23 das 36 instâncias. Para as 8 instâncias (n60p30s40-2, n60p30s50-1, n60p30s60-3, n60p60s30-1, n60p60s50-1, n60p60s50-2, n60p90s40-1 e n60p90s50-3), as quais o AILS-bCAP1 não melhorou as melhores soluções conhecidas, os valores dos desvios percentuais também permaneceram baixos. Ainda, pelos baixos valores dos desvios-padrão relativos, o algoritmo apresenta convergência consistente para esse conjunto de instâncias.

Tabela 35 – Desempenho das versões do AILS-bCAP1 no extremo (f_1^{best}, f_2) para instâncias Ahonen, De Alvarenga e Amaral (2014)

Instância	*n*	pGA[(a)]			AILS-bCAP1		
		f_1^{avg}	*Desv*	*DPr*	f_1^{avg}	*Desv*	*DPr*
n60p30s30-1	60	204173,5	0,037	0,024	**204143,6**	0,022	0,012
n60p30s30-2	60	193310,9	0,051	0,054	**193229,0**	0,009	0,013
n60p30s30-3	60	161533,3	0,036	0,022	**161503,2**	0,017	0,015
n60p30s40-1	60	**135171,3**	0,028	0,021	135180,9	0,035	0,021
n60p30s40-2	60	159230,0	0,073	0,058	**159191,2**	0,049	0,054
n60p30s40-3	60	**158973,5**	0,000	0,000	158992,4	0,012	0,016
n60p30s50-1	60	**110514,1**	0,019	0,033	110542,0	0,044	0,044
n60p30s50-2	60	**115320,3**	0,001	0,001	115365,8	0,041	0,034
n60p30s50-3	60	114287,4	0,114	0,093	**114181,4**	0,021	0,046
n60p30s60-1	60	108139,4	0,015	0,018	**108134,1**	0,010	0,015
n60p30s60-2	60	110040,4	0,094	0,085	**109965,9**	0,026	0,022
n60p30s60-3	60	**91652,2**	0,036	0,054	91660,0	0,045	0,027
n60p60s30-1	60	**445380,4**	0,001	0,000	445409,0	0,007	0,008
n60p60s30-2	60	407962,9	0,018	0,012	**407922,4**	0,008	0,005
n60p60s30-3	60	**416933,8**	0,001	0,000	416962,3	0,008	0,006
n60p60s40-1	60	313383,0	0,020	0,015	**313344,2**	0,008	0,011
n60p60s40-2	60	320802,7	0,014	0,008	**320776,5**	0,006	0,005
n60p60s40-3	60	363185,3	0,045	0,022	**363093,7**	0,020	0,016
n60p60s50-1	60	**273422,8**	0,001	0,000	273468,8	0,018	0,013
n60p60s50-2	60	**269682,8**	0,001	0,000	269740,8	0,022	0,010
n60p60s50-3	60	**295423,9**	0,000	0,000	295460,1	0,013	0,013
n60p60s60-1	60	**227897,8**	0,001	0,000	227907,7	0,005	0,005
n60p60s60-2	60	246570,6	0,019	0,013	**246542,9**	0,008	0,007
n60p60s60-3	60	**206496,2**	0,001	0,003	206519,9	0,013	0,009
n60p90s30-1	60	629012,4	0,024	0,014	**628889,8**	0,005	0,006
n60p90s30-2	60	561663,8	0,088	0,065	**561185,3**	0,003	0,002
n60p90s30-3	60	**587827,6**	0,001	0,002	587828,8	0,001	0,001
n60p90s40-1	60	474100,4	0,011	0,008	**474067,9**	0,005	0,002
n60p90s40-2	60	480241,7	0,058	0,056	**479967,5**	0,001	0,002
n60p90s40-3	60	512495,9	0,009	0,005	**512460,4**	0,002	0,002
n60p90s50-1	60	479771,2	0,017	0,020	**479697,0**	0,001	0,002
n60p90s50-2	60	445212,5	0,041	0,030	**445070,0**	0,009	0,008
n60p90s50-3	60	495130,7	0,016	0,027	**495062,3**	0,002	0,001
n60p90s60-1	60	385458,8	0,007	0,006	**385436,8**	0,002	0,002
n60p90s60-2	60	344811,4	0,011	0,007	**344777,3**	0,001	0,001
n60p90s60-3	60	411269,3	0,016	0,014	**411212,5**	0,002	0,002
Média		312680,1	0,026	0,022	312635,9	0,014	0,013

As melhores médias estão destacadas em negrito.

7.8.6 Avaliação dos resultados para o bCAP para as instâncias Palubeckis (2015)

Os resultados dos testes com as maiores instâncias utilizadas para o bCAP na literatura, com tamanhos $110 \leq n \leq 300$ (PALUBECKIS, 2015) são mostrados na Tabela 36. Os melhores valores conhecidos na literatura também foram obtidos pelo pGA (KALITA; DATTA; PALUBECKIS, 2019). A tabela está estruturada conforme a Tabela 32, sendo adicionada a penúltima coluna, T_{avg}(pGA), com as médias dos tempos de execução, em segundos, do pGA. Os valores destacados em negrito representam os melhores valores obtidos em comparação aos melhores valores conhecidos para a função objetivo f_1.

Tabela 36 – Comparação dos resultados do bCAP para instâncias Palubeckis (2015)

Instância	n	Melhor custo do manuseio de materiais				Melhor comprimento do corredor				T_{avg} (pGA)	T_{avg}
		pGA[(a)]		AILS-bCAP1		pGA[(a)]		AILS-bCAP1			
		f_1^*	f_2	$D_{f_1^{best}}$	D_{f_2}	f_1	f_2^*	D_{f_1}	$D_{f_2^{best}}$		
p110	110	2219187,0	**0,00**	**-1352,0**	**0**	2219187,0	**0,00**	**-1352,0**	**0**	17341	1320
p120	120	3143674,0	**0,00**	**-2427,0**	**0**	3143674,0	**0,00**	**-2427,0**	**0**	24521	1440
p130	130	3942320,5	**0,13**	**-2067,0**	**0**	3942320,5	**0,13**	**-2067,0**	**0**	30906	1561
p140	140	4634890,0	**0,13**	**-6361,0**	**0**	4634890,0	**0,13**	**-6361,0**	**0**	36409	1681
p150	150	5317335,5	0,39	**-4866,0**	**-0,26**	5317468,5	**0,13**	**-4999,0**	**0**	43844	1802
p160	160	7446625,0	0,22	**-9958,0**	**-0,22**	7446719,0	**0,00**	**-10052,0**	**0**	37434	1922
p170	170	8324458,0	**0,11**	**-9179,0**	**0**	8324458,0	**0,11**	**-9179,0**	**0**	46635	2042
p180	180	9383039,5	0,21	**-9790,0**	**-0,21**	9383213,5	**0,00**	**-9964,0**	**0**	57002	2164
p190	190	12249508,0	**0,09**	**-22800,0**	**0**	12249508,0	**0,09**	**-22800,0**	**0**	62754	2284
p200	200	13778294,5	**0,09**	**-36868,0**	**0**	13778294,5	**0,09**	**-36868,0**	**0**	89963	2406
p210	210	14787232,5	**0,09**	**-29582,0**	**0**	14787232,5	**0,09**	**-29582,0**	**0**	106335	2524
p220	220	18711170,5	**0,00**	**-34329,0**	**0**	18711170,5	**0,00**	**-34329,0**	**0**	123913	2648
p230	230	23423917,0	0,15	**-51517,0**	**-0,15**	23425867,0	**0,00**	**-53467,0**	**0**	136198	2770
p240	240	23426253,0	**0,00**	**-66965,0**	**0**	23426253,0	**0,00**	**-66965,0**	**0**	146489	2893
p250	250	27357813,5	0,22	**-93982,0**	**-0,15**	27358146,5	**0,07**	**-94315,0**	**0**	55578	3014
p260	260	31739516,5	0,14	**-88692,0**	**-0,14**	31740716,5	**0,00**	**-89892,0**	**0**	176807	3146
p270	270	34558517,5	0,13	**-77671,0**	**-0,13**	34559577,5	**0,00**	**-78731,0**	**0**	194937	3272
p280	280	37051720,0	**0,00**	**-128335,0**	**0**	37051720,0	**0,00**	**-128335,0**	**0**	208487	3394
p290	290	43270045,5	0,25	**-141428,0**	**-0,25**	43270405,5	**0,00**	**-141788,0**	**0**	174877	3527
p300	300	48107893,0	0,18	**-138243,0**	**-0,12**	48108164,0	**0,06**	**-138514,0**	**0**	197561	3643
Média		18643670,6	0,13	-47820,6	-0,08	18643949,3	0,05	-48099,4	0	98400	2473

[(a)] Os melhores valores conhecidos obtidos por (KALITA; DATTA; PALUBECKIS, 2019).
Os melhores valores são destacados em negrito.

Para os experimentos computacionais com as maiores instâncias, Tabela 36, o AILS0bCAP1 melhorou os melhores valores conhecidos para a função objetivo f_1 de todas as 20 instâncias no extremo (f_1^{best}, f_2). Ainda, ao considerar a relação de dominância, o AIL-bCAP1 encontrou soluções que dominam todas as melhores soluções da literatura, dados os valores negativos ou zero dos pares ($D_{f_1^{best}}$,D_{f_2}).

Considerando os resultados alcançados no extremo (f_1, f_2^{best}) os valores para a função objetivo f_2 foram iguais aos melhores valores conhecidos para todas as instâncias. Contudo, os melhores custos de manuseio de materiais foram melhorados, ou seja, as soluções encontradas nesse extremo dominam todas as melhores soluções conhecidas, como mostram os valores negativos ou zero dados pares (D_{f_1}, $D_{f_2^{best}}$).

Mesmo não sendo possível uma relação direta entre os tempos de execução dos algoritmos, é possível notar que os tempos limites utilizados pelo AILS-bCAP1 para esse conjunto de instâncias são significativamente menores que os tempos do algoritmo pGA. A média geral do tempo de execução foi de 2473s do AILS-bCAP1 contra 98400s do pGA.

A Tabela 37 apresenta as medidas de desempenho para as instâncias desse conjunto de testes, calculados a partir das soluções médias e dos desvios-padrão da função objetivo f_1 no extremo (f_1^{best}, f_2). A tabela está estruturada conforme Tabela 35.

Tabela 37 – Desempenho do AILS-bCAP1 no extremo (f_1^{best}, f_2) para instâncias Palubeckis (2015)

Instância	n	pGA			AILS-bCAP1		
		f_1^{avg}	*Desv*	*DPr*	f_1^{avg}	*Desv*	*DPr*
p110	110	2220799,40	0,073	0,032	**2217888,4**	-0,059	0,002
p120	120	3145535,40	0,059	0,031	**3141309,5**	-0,075	0,001
p130	130	3945289,60	0,075	0,035	**3940416,8**	-0,048	0,003
p140	140	4636382,20	0,032	0,019	**4628844,4**	-0,130	0,008
p150	150	5318504,30	0,022	0,021	**5312796,7**	-0,085	0,008
p160	160	7448021,30	0,019	0,017	**7437630,3**	-0,121	0,028
p170	170	8325377,30	0,011	0,010	**8316810,6**	-0,092	0,023
p180	180	9384793,30	0,019	0,010	**9373802,6**	-0,098	0,007
p190	190	12250721,00	0,010	0,011	**12228305,2**	-0,173	0,009
p200	200	13783185,40	0,035	0,010	**13746246,7**	-0,233	0,022
p210	210	14787394,40	0,001	0,008	**14761157,8**	-0,176	0,024
p220	220	18717421,80	0,033	0,008	**18688587,6**	-0,121	0,054
p230	230	23425623,30	0,007	0,003	**23400769,4**	-0,099	0,102
p240	240	23427773,10	0,006	0,005	**23374472,0**	-0,221	0,046
p250	250	27360061,10	0,008	0,004	**27280910,5**	-0,281	0,043
p260	260	31752321,80	0,040	0,005	**31672376,3**	-0,212	0,059
p270	270	34558838,80	0,001	0,008	**34520040,6**	-0,111	0,099
p280	280	37061642,00	0,027	0,008	**36952742,0**	-0,267	0,051
p290	290	43270825,30	0,002	0,006	**43159683,5**	-0,255	0,037
p300	300	48132771,60	0,052	0,006	**48001811,6**	-0,221	0,039
Média		18647664,12	0,027	0,013	18607830,1	-0,192	0,047

As melhores médias estão destacadas em negrito.

Para esse conjunto de testes, os valores baixos dos desvios percentuais das médias mostram que os algoritmos propostos também mantêm um desempenho robusto, mesmo com o aumento considerável dos tamanhos das instâncias. O algoritmo obteve valores médios das soluções menores que as melhores soluções da literatura para todas as instâncias. A convergência

consistente do algoritmo para esse conjunto de instâncias pode ser observada pelos baixos valores dos desvios-padrão.

7.9 CONSIDERAÇÕES FINAIS

O algoritmo proposto para o bCAP executa duas fases com aspectos de intensificação e diversificação distintos, buscando as melhores soluções nos extremos da fronteira de Pareto. Os experimentos computacionais foram realizados com várias instâncias do bCAP consideradas de grande porte, com tamanhos de $30 \leq n \leq 300$ facilidades. Os testes mostraram que o algoritmo é robusto, apresentando excelentes resultados, mantendo uma convergência consistente e um bom desempenho, mesmo com o aumento nos tamanhos das instâncias.

Uma análise das principais componentes do AILS-bCAP foi realizada com o objetivo de identificar o comportamento do algoritmo com rotinas de perturbação distintas, sendo identificada a versão AILS-bCAP1 como sendo a versão com desempenho superior às outras variantes. Além disso, a comparação das versões do AILS-bCAP1 com buscas locais alternativas, permitiu constatar que a busca local com movimentos de inserção com o uso de técnicas para o cálculo do ganho na função objetivo nos movimentos de vizinhanças tem um desempenho melhor dentre as alternativas propostas.

De modo geral, o AILS-bCAP1 obteve resultados e desempenho competitivos para as instâncias testadas. Sendo assim, o AILS-bCAP1 pode ser considerado uma boa ferramenta para se obter soluções para bCAP de alta qualidade e em tempos computacionais razoáveis, ao considerar os dois extremos da fronteira de Pareto.

8 CONCLUSÕES E RECOMENDAÇÕES FUTURAS

Como proposto inicialmente, este estudo apresenta soluções para o problema de layout em linha única (SRFLP) e três problemas de layouts em linha dupla conhecidos com DRLP, PROP e bCAP, que têm grande importância na indústria, principalmente em sistemas flexíveis de manufatura. Os objetivos específicos foram atingidos com as propostas de algoritmos inteligentes baseados em meta-heurísticas que alcançaram bons resultados para os problemas.

Para o SRFLP, foi desenvolvido um GRASP, denominado GRASP-F, com uma etapa adicional de diversificação baseada em uma matriz de frequência e com técnicas para acelerar o cálculo do ganho na função objetivo nos movimentos de vizinhanças da busca local. Para o DRLP, caracterizado pela dificuldade de resolução por ser tratar de um problema com variáveis inteiras e contínuas, a proposta apresentada foi baseada no PSO, denominada PSO-DRLP, com a definição de duas fases e uso de busca local para melhorar a convergência do algoritmo, não sendo utilizados modelos de programação linear para a determinação das posições absolutas das facilidades nos arranjos das linhas. Para o PROP, foi proposto um algoritmo adaptativo em duas fases, que alterna entre fases com características de intensificação e diversificação distintos, além de fazer uso de técnicas que reduzem o esforço computacional para a avaliação no ganho da função objetivo nos movimentos de troca 2-opt, utilizado para explorar as soluções vizinhas na busca local. O algoritmo proposto para o bCAP, denominado AILS-bCAP, também alterna entre duas fases com aspectos de intensificação e diversificação distintos e faz uso, na busca local, de técnicas que reduzem o esforço computacional para a avaliação no ganho da função objetivo nos movimentos de troca 2-opt e de inserção.

Uma melhoria foi realizada na técnica do cálculo do ganho na função objetivo do movimento de vizinhança por inserção proposta por Guan e Lin (2016), sendo utilizada nas rotinas da busca local dos algoritmos propostos.

Os algoritmos apresentaram resultados competitivos para a base de testes com instâncias consideradas de grande porte na literatura. Com o GRASP-F soluções para instâncias com até 1000 facilidades foram obtidas, tornando este trabalho o segundo na literatura conhecida a considerar problemas dessa magnitude. Foram obtidas melhoras em 23 das 93 instâncias utilizadas. Para as demais, soluções bem próximas aos melhores resultados conhecidos foram encontradas.

O PSO para o DRLP encontrou soluções melhores para 6 instâncias e alcançou os mesmos valores da literatura para 29 instâncias. Para as 16 restantes, o PSO-DRLP não obteve sucesso, mas, mesmo assim, os valores encontrados estão bem próximos dos melhores valores conhecidos.

A proposta de solução para o PROP melhorou 49 de 100 instâncias com resultados prévios conhecidos e para 51 instâncias restantes, os melhores resultados conhecidos foram alcançados, inclusive melhorando as médias dos valores das soluções para todas as instâncias. Além desses testes, foram realizados experimentos com três variantes do AILS, utilizando 6 instâncias com tamanhos n = {250, 260, 270, 280, 290, 300}, inéditas no contexto do PROP.

O bCAP, quando considerados o extremo (f_1^{best}, f_2), encontrou soluções que dominam as soluções da literatura em 41 instâncias. Em apenas 5 instâncias as soluções foram dominadas e as 30 restantes são iguais ou indiferentes. Considerando a melhora a função objetivo f_1, nesse extremo, foram obtidas soluções melhores para 71 das 76 instâncias. No extremo (f_1, f_2^{best}), 66 soluções dominam as soluções da literatura e apenas 6 soluções foram dominadas, sendo as 4 restantes iguais a literatura. Considerando a melhora a função objetivo f_1, nesse extremo, foram obtidas soluções melhores para 70 das 76 instâncias.

As propostas de soluções apresentadas podem ser consideradas excelentes alternativas para o tratamento de problemas de layouts em linhas única e dupla, sendo possível a obtenção de resultados muito bons para problemas de grande porte em tempos computacionais baixos.

Como trabalhos futuros, destacam-se os seguintes tópicos a serem abordados:

- A necessidade de um estudo ainda mais aprofundado sobre técnicas que possam melhorar o desempenho na avaliação dos ganhos na função objetivo, dos movimentos de vizinhança para o PSO-DRLP, habilitando-o a resolver instâncias ainda maiores;
- Um estudo em relação à etapa de perturbação do AILS e AILS-bCAP, pois a solução proposta é simples e mostrou-se efetiva, entretanto, fazendo o uso de um histórico de buscas ou de uma técnica mais refinada na atualização do valor da intensidade P, pode-se levar a soluções ainda melhores;
- Adição à meta-heurística AILS-bCAP de um conjunto Pareto-ótimos de soluções, pois a implementação proposta neste trabalho considera apenas as soluções nos extremos da fronteira de Pareto; e

- Por fim, o estudo de técnicas que possam agilizar o cálculo do ganho na função objetivo, nos movimentos de vizinhança, que considerem pares de facilidades em linhas opostas nos problemas em duas linhas.

REFERÊNCIAS

ABRAHAM, A.; GUO, H.; LIU, H. Swarm Intelligence: Foundations, Perspectives and Applications. In: NEDJAH, N.; MOURELLE, L. M. (Ed.) **Swarm Intelligent Systems. Studies in Computational Intelligence**, Springer, Berlin, Heidelberg, v. 26, p. 3-25, 2006.

AHONEN, H.; DE ALVARENGA, A. G.; AMARAL, A. R. S. Simulated annealing and tabu search approaches for the Corridor Allocation Problem, **European Journal of Operational Research**, v. 232, n. 1, p. 221-233, 2014.

AIEX, R. M.; BINATO, S.; RESENDE, M. G. C. Parallel GRASP with path-relinking for job shop scheduling. **Parallel Computing,** v. 29, n. 4, p. 393-430, 2003.

ALFEREZ, W. E. B. **Estudo de variantes da Particle Swarm Optimization aplicadas ao planejamento da expansão de sistemas de transmissão**. 2013, 83 f. Dissertação (Mestrado em Engenharia Elétrica) – Universidade Estadual de Campinas, Campinas, 2013.

AMARAL, A. R. S. On the exact solution of a facility layout problem. **European Journal of Operational Research**, v. 173, n. 2, p. 508-518, 2006.

AMARAL, A. R. S. An exact approach to the one-dimensional facility layout problem. **Operations Research**, v. 56, n. 4, p. 1026-1033., 2008a.

AMARAL, A. R. S. Enhanced local search applied to the single row facility layout problem. In: XL SBPO - SIMPÓSIO BRASILEIRO DE PESQUISA OPERACIONAL, 2008, João Pessoa, **Anais eletrônico...** João Pessoa, 2008b, p. 1638-1647. Disponível em: <http://www.din.uem.br/sbpo/sbpo2008/pdf/arq0026.pdf>. Acesso em: 04 jan. 2019.

AMARAL, A. R. S. A new lower bound for the single row facility layout problem. **Discrete Applied Mathematics**, v. 157, n. 1, p. 183-190, 2009.

AMARAL, A. R. S. The corridor allocation problem, **Computers & Operations Research**, v. 39, n. 12, p. 3325-3330, 2012.

AMARAL, A. R. S. A parallel ordering problem in facilities layout, **Computers & Operations Research**, v. 40, n. 12, p. 2930-2939, 2013a.

AMARAL, A. R. S. Optimal solutions for the double row layout problem. **Optimization Letters**, v. 7, n. 2, p. 407-413. 2013b.

AMARAL, A. R. S. A mixed-integer programming formulation for the double row layout of machines in manufacturing systems, **International Journal of Production Research**, v. 57, n. 1, p. 34-47, 2018.

AMARAL, A. R. S. A heuristic approach for the double row layout problem. **Ann Oper Res**, 2020. Disponível em: <https://doi.org/10.1007/s10479-020-03617-5>. Acesso em: 05 maio 2020.

AMARAL, A. R. S. A mixed-integer programming formulation of the double row layout problem based on a linear extension of a partial order. **Optimization Letters**, v. 15, p. 1407-1423, 2021.

AMARAL, A. R. S.; LETCHFORD A. N. A polyhedral approach to the single row facility layout problem. **Mathematical Programming**, v. 141, n. 1-2, p. 453-477, 2013.

ANDRADE, R. C.; FERREIRA, M. D. S. Problema de ordenação de departamentos em linhas simples. In: XLIX SBPO-SIMPÓSIO BRASILEIRO DE PESQUISA OPERACIONAL, 2017,

Blumenau. **Anais eletrônico...** Blumenau, 2017, p. 2874-2879. Disponível em: <http://www.sbpo2017.iltc.br/pdf/168752.pdf>. Acesso em: 26 jun. 2019.

ANJOS, M. F.; FISCHER A.; HUNGERLÄNDER P. **Solution approaches for equidistant double and multi-row facility layout problems**, HEC Montréal, Polytechnique Montréal, McGill University, Université du Quèbec à Montréal, 2015. Disponível em: <https://www.gerad.ca/en/papers/G-2015-06>. Acesso em: 28 maio 2019.

ANJOS, M. F.; HUNGERLÄNDER, P. **A semidefinite optimization approach to space-free multi-row facility layout**. HEC Montréal, Polytechnique Montréal, McGill University, Université du Quèbec à Montréal, 2012. Disponível em: <https://www.gerad.ca/en/papers/G-2012-03>. Acesso em: 28 maio 2019.

ANJOS, M. F.; KENNINGS, A.; VANNELLI, A. A semidefinite Optimization approach for the single-row layout problem with unequal dimensions. **Discrete Optimization**, v. 2, n. 2, p. 113-122, 2005.

ANJOS, M. F.; VANNELLI, A. Computing globally optimal solutions for single-row layout problems using semidefinite programming and cutting planes. **INFORMS Journal on Computing**, v. 20, n. 4, p. 611-617, 2008.

ANJOS, M. F.; VIEIRA, M. V. Mathematical Optimization approaches for facility layout problems: The state-of-the-art and future research directions. **European Journal of Operational Research**, v. 261, n. 1, p. 1-16, 2017.

ANJOS, M. F.; YEN, G. Provably near-optimal solutions for very large single-row facility layout problems. **Optimization Methods and Software**, v. 24, n. 4, p. 805-817, 2009.

ARAUJO, S. A. D.; POLDI, K. C.; SMITH, J. A Genetic Algorithm for the one-dimensional cutting stock problem with setups. **Pesquisa Operacional**, Rio de Janeiro, v. 34, n. 2, p. 165-187, 2014.

ARROYO, J. E. C. **Heurísticas e meta-heurísticas para otimização combinatória multiobjetivo**. 2002, 255 f. Tese (Doutorado em Engenharia Elétrica e de Computação) – Unicamp, Campinas, 2002.

BAZARAA, M. S.; JARVIS, J. J.; SHERALI, H. D. **Linear Programming and Network Flows**. 4. ed. New Jersey, John Wiley & Sons, 2009

BERGH F, V.; ENGELBRECHT, A. P. A study of particle swarm optimization particle trajectories. **Information Sciences** v. 176, n. 8, p. 937-971, 2006.

BISSOLI, D. C. **Meta-heurísticas para resolução de alguns problemas de planejamento e controle da produção**. 2018, 142 f. Tese (Doutorado em Ciência da computação) – Universidade Federal do Espirito Santo, Vitória, 2018.

BONYADI, M. R.; MICHALEWICZ, Z. Particle Swarm Optimization for Single Objective Continuous Space Problems: A Review, **Evolutionary Computation**, v. 25, n. 1, p. 1-54, 2017.

BRUNESE, P. A.; TANCHOCO, J. M. A. On Implied Within-building Constraints for Machine Layout. **International Journal of Production Research**, v. 51, n. 6, p. 1937-1952, 2013.

CELLIN, B. D. P. **Métodos para Resolução eficiente de Problemas de Layout**. 2017, 72 f. Dissertação (Mestrado em informática) – Universidade Federal do Espirito Santo, Vitória, 2017.

CHAE, J.; REGAN, A. C. A mixed integer programming model for a double row layout problem. **Computers & Industrial Engineering**, v. 140, p. 106-244, 2020.

CHIARANDINI, M.; STÜTZLE, T. An application of iterated local search to the graph coloring problem, In: COMPUTATIONAL SYMPOSIUM ON GRAPH COLORING AND ITS GENERALIZATIONS, 2002, Ithaca. **Anais eletrônico...** New York, A.M.D.S. Johnson, 2002, p. 112-125. Disponível em: <http://www.cs.colostate.edu/~howe/cs640/papers/stutzle.pdf>. Acesso em: 15 maio 2019.

CHUNG, J.; TANCHOCO, J. M. A. The double row layout problem, **International Journal of Production Research**, v. 48, n. 3, p. 709-727, 2010.

CPUBENCHMARK. **PassMark Software: CPU Benchmarks**, c2020. Página inicial. Disponível em: https://www.cpubenchmark.net. Acesso em: 26 de jun. de 2020.

CRAVO, G. L.; AMARAL, A. R. S. Um algoritmo GRASP aplicado ao problema de layout de facilidades em fila única. In: XLVII SBPO - SIMPÓSIO BRASILEIRO DE PESQUISA OPERACIONAL, 2015, Porto de Galinhas. **Anais eletrônico...** Porto de Galinhas, 2015. Disponível em: <http://www.din.uem.br/sbpo/sbpo2015/pdf/142380.pdf>. Acesso em: 16 mar. 2017.

CRAVO, G. L.; AMARAL, A. R. S. Um algoritmo GRASP com fase de diversificação para problema de layout de facilidades em fila única. In: XLIX SIMPÓSIO BRASILEIRO DE PESQUISA OPERACIONAL, 2017, Blumenau. **Anais eletrônico...** Blumenau, 2017. Disponível em: <http://www.sbpo2017.iltc.br/pdf/169032.pdf>. Acesso em: 16 jun. 2018.

CRAVO, G. L.; AMARAL, A. R. S. A GRASP algorithm for solving large-scale single row facility layout problems, **Computers & Operations Research**, v. 106, p. 49-61, 2019.

CRAVO, G. L.; BISSOLI D.C., D.; AMARAL S., A. R. Otimização por Enxame de Partículas híbrido de duas fases Aplicado o Problema de Layout em Linha Dupla: Two-phase hybrid Particle Swarm Optimization Applied to the Double Row Layout Problem. **Inteligencia Artificial**, v. 24, n. 67, p. 51–70, 2021.

CRAVO, G. L.; RIBEIRO, G. M.; LORENA, L. A. N. A greedy randomized adaptive search procedure for the point-feature cartographic label placement. **Computers & Geosciences**, v. 34, n. 4, p. 373-386, 2008.

CURY, A. **Organização & métodos - uma visão holística**, 8. ed., São Paulo: Editora Atlas, 2006.

DAHLBECK, M., FISCHER, A., FISCHER, F. Decorous combinatorial lower bounds for row layout problems. **European Journal of Operational Research**, v. 286, n. 3, p. 929-944, 2020.

DATTA, D.; AMARAL, A. R. S.; FIGUEIRA, J. R. Single row facility layout problem using a permutation-based genetic algorithm. **European Journal of Operational Research**, v. 213, n. 2, p. 388-394, 2011.

DE ALVARENGA, A. G.; NEGREIROS-GOMES, F. J.; MESTRIA, M. Metaheuristic methods for a class of the facility layout problem. **Journal of Intelligent Manufacturing**, v. 11, n. 4, p. 421-430, 2000.

DEB, K. Multi-objective Optimisation Using Evolutionary Algorithms: An Introduction. **Multi-objective Evol. Optim. Prod. Des. Manuf.** [S.l.]: Springer London, 2011, p. 3–34.

DIAS, M. A. P. **Administração de materiais: uma abordagem logística**. 4. ed. São Paulo: Editora Atlas, 1993

DICKEY, J.; HOPKINS, J. Campus building arrangement using topaz. **Transportation Research**, v. 6, n. 1, p. 59-68, 1972.

DRIRA, A.; PIERREVAL, H.; HAJRI-GABOUJ, S. Facility layout problems: A survey. **Annual Reviews in Control**, v. 31, n. 2, p. 255-267, 2007.

EBERHART, R. C.; SHI, Y. Comparing inertia weights and constriction factors in particle swarm optimization, In: 2000 CONGRESS ON EVOLUTIONARY COMPUTATION. CEC00 (CAT. NO.00TH8512), 2000, La Jolla, **Anais eletrônico...** La Jolla, 2000, p. 84-88. Disponível em: <https://doi.org/10.1109/CEC.2000.870279>. Acesso em: 27 maio. 2019.

EBERHART, R. C.; SHI, Y. Particle swarm optimization: developments, applications and resources, In: 2001 CONGRESS ON EVOLUTIONARY COMPUTATION (IEEE CAT. NO.01TH8546), 2001, Seoul. **Anais eletrônico...** Seoul, 2001, p. 81-86. Disponível em: <https://doi.org/10.1109/CEC.2001.934374>. Acesso em: 27 maio 2019.

ELBES, M. et al. A survey on particle swarm optimization with emphasis on engineering and network applications. **Evolutionary Intelligence**, v. 12, n. 2, p. 113-129, 2019.

ELSHAFEI, A. N. Hospital layout as a quadratic assignment problem. **Operational Research Quarterly**, v. 28, n. 1, p. 167-179, 1977.

FARIA, H. et al. Power transmission network design by greedy randomized adaptive path relinking. **IEEE Transactions on Power Systems**, v. 20, n. 1, p. 43-49, 2005.

FEO, T. A.; RESENDE, M. G. C. Greedy randomized adaptive search procedures. **Journal of Global Optimization**, v. 6, n. 2, p. 109-133, 1995.

FERREIRA, J. H. I. **Um modelo de otimização multi-objetivo aplicado no dimensionamento da capacidade instalada de uma Pequena Central Hidrelétrica**. 2020. 214 f. Tese (Doutorado em Engenharia Elétrica) - Universidade de Federal de Uberlândia, Uberlândia, 2020.

FESTA, P.; RESENDE, M. G. C. An annotated bibliography of GRASP - Part I: Algorithms. **International Transactions in Operational Research**, Blackwell Publishing Ltd, v. 16, n. 1, p. 1-24, 2009a.

FESTA, P.; RESENDE, M. G. C. An annotated bibliography of GRASP - Part II: Applications. **International Transactions in Operational Research**, Blackwell Publishing Ltd, v. 16, n. 2, p. 131-172, 2009b.

FISCHER, A.; FISCHER, F.; HUNGERLÄNDER, P. New exact approaches to row layout problems. **Math. Program. Comput.**, v. 11, n. 4, p. 703-754, 2019.

FONSECA, C. M.; FLEMING, P. J. An overview of evolutionary algorithms in multiobjective optimization. **Evolutionary Computation**, v. 3, n. 1, p. 1-16, 1995.

GEN, M.; IDA, K.; CHENG, C. Multirow machine layout problem in fuzzy environment using genetic algorithms. **Computers & Industrial Engineering**, v. 29, n 1-4, p. 519-523, 1995.

GHOSH, D.; KOTHARI, R. **Population Heuristics for the Corridor Allocation Problem.** Research and Publication Department, Indian Institute of Management Ahmedabad, 2012. Disponível em: <https://web.iima.ac.in/assets/snippets/workingpaperpdf/ 8306072762012-09-02.pdf>. Acesso em: 01 jul. 2019.

GLOVER, F.; LAGUNA, M. **Tabu Search**, Norwell, MA, USA: Editora Kluwer Academic Publishers, 1997.

GUAN, J.; LIN, G. Hybridizing variable neighborhood search with ant colony optimization for solving the single row facility layout problem. **European Journal of Operational Research**, v. 248, n. 3, p. 899-909, 2016.

GUAN, J. et al. A decomposition-based algorithm for the double row layout problem. **Applied Mathematical Modelling**, v. 77, p. 963-979, 2020.

HASHIMOTO, K. **Técnicas de otimização combinatória multiobjetivo aplicadas na estimação do desempenho elétrico de redes de distribuição**. 2004, 114 f. Tese (Doutorado em Sistemas de Potência) - Escola Politécnica, Universidade de São Paulo, São Paulo, 2004.

HASSAN, M. M. D. Machine layout problem in modern manufacturing facilities. **International Journal of Production Research**, v. 32, n. 11, p. 2559-2584, 1994.

HELSGAUN, K. (2000). An Effective Implementation of the Lin-Kernighan Traveling Salesman Heuristic. **European Journal of Operational Research**, v. 126, n. 1, p. 106-130.

HERAGU, S. S. **Facilities design**, 6. ed. Boston: Editora PWS Pub. Co, 1997.

HERAGU, S. S.; ALFA, A. S. Experimental analysis of simulated annealing based algorithms for the layout problem. **European Journal of Operational Research**, v. 57, n. 2, p. 190-202, 1992.

HERAGU, S. S.; KUSIAK, A. Machine layout problem in flexible manufacturing systems. **Operations Research**, v. 36, n. 2, p. 258-268, 1988.

HERAGU, S. S.; KUSIAK, A. Efficient models for the facility layout problem. **European Journal of Operational Research**, v. 53, n. 1, p. 1-13, 1991.

HOLLAND, J. H. **Adaptation in Natural and Artificial Systems**, Michigan, University of Michigan Press: Ann Arbor, 1975.

HOSSEINI-NASAB, H. et al. Classification of facility layout problems: a review study. **The International Journal of Advanced Manufacturing Technology**, v. 94, n. 1-4, p. 957-977, 2018.

HUNGERLÄNDER, P.; ANJOS, M. F. A semidefinite optimization-based approach for global optimization of multi-row facility layout. **European Journal of Operational Research**, v. 245, n. 1, p. 46 - 61, 2015.

HUNGERLÄNDER, P.; RENDL, F. A computational study and survey of methods for the single-row facility layout problem. **Computational Optimization and Applications**, v. 55, n. 1, p. 1-20, 2012.

HUNGERLÄNDER, P.; RENDL, F. Semidefinite relaxations of ordering problems. **Mathematical Programming**, v. 140, n. 1, p. 77-97, 2013.

KALITA Z., DATTA D., Solving the bi-objective corridor allocation problem using a permutation-based genetic algorithm, **Computers & Operations Research**, v. 52, p. 123-134, 2014.

KALITA Z., DATTA D., PALUBECKIS G., Bi-objective corridor allocation problem using a permutation-based genetic algorithm hybridized with a local search technique, **Soft Computing**, v. 23, n. 3, p. 961-986, 2019.

KALITA, Z.; DATTA, D. Corridor Allocation as a Constrained Optimization Problem Using a Permutation-Based Multi-objective Genetic Algorithm. In: BENNIS, F.; BHATTACHARJYA, R. (Ed.) **Nature-Inspired Methods for Metaheuristics Optimization. Modeling and Optimization in Science and Technologies**, Springer, Cham, 2020, v. 16. p. 335-358.

KENNEDY, J.; EBERHART, R. Particle swarm optimization, In: ICNN'95 - International Conference on Neural Networks, 1995, Perth. **Anais eletrônico...** Perth, Australia, v. 4, p. 1942-

1948, 1995. Disponível em: <https://doi.org/10.1109/ICNN.1995.488968>. Acesso em: 28 maio 2019.

KOCHHAR, J. S. MULTI-HOPE: A tool for multiple floor layout problems. **International Journal of Production Research**, v. 36, n. 12, p. 3421-3435, 1998.

KONAK, A.; COIT, D.; SMITH, A. Multi-objective optimization using genetic algorithms: a tutorial. **Reliability Engineering & System Safety**, v. 91, p. 992-1007, 2006.

KOTHARI, R.; GHOSH, D. The single row facility layout problem: state of the art. **OPSEARCH,** v. 49, n. 4, p. 442-462, 2012.

KOTHARI, R.; GHOSH, D. An efficient genetic algorithm for single row facility layout. **Optimization Letters**, v. 8, n. 2, p. 679–690, 2013a.

KOTHARI, R.; GHOSH, D. Tabu search for the single row facility layout problem using exhaustive 2-opt and insertion neighborhoods. **European Journal of Operational Research**, v. 224, n. 1, p. 93-100, 2013b.

KOTHARI, R.; GHOSH, D. A scatter search algorithm for the single row facility layout problem. **Journal of Heuristics**, v. 20, n. 2, p. 125-142, 2014.

KOTHARI, V. et al. A Survey on Particle Swarm Optimization in Feature Selection. In: KRISHNA, P.V.; BABU, M. R.; ARIWA, E. (Ed.) **Global Trends in Information Systems and Software Applications. ObCom 2011. Communications in Computer and Information Science**, Berlin, Heidelberg, Springer, 2012, v. 270, p. 192-201.

KUMAR, R. M. S. et al. Scatter search algorithm for single row layout problem in FMS. **Advances in Production Engineering & Management**, v. 3, n. 4, p. 193-204, 2008.

LOURENÇO, H. R.; MARTIN, O. C.; STÜTZLE, T. Iterated local search. In: GLOVER, F.; KOCHENBERGER, G.A. (Ed.) **Handbook of Metaheuristics. International Series in Operations Research & Management Science**, Boston, Springer, 2003, v. 57, p. 320-353.

LOURENÇO, H. R.; MARTIN, O. C.; STÜTZLE, T. Iterated Local Search: Framework and Applications. In: GENDREAU, M.; POTVIN, J. Y. (Ed.) **Handbook of Metaheuristics. International Series in Operations Research & Management Science**, Boston, Springer, v. 146, p. 363-397.

LOVE, R. F.; WONG, J. Y. On solving a one-dimensional space allocation problem with integer programming. **INFOR: Information Systems and Operational Research**, v. 14, n. 2, p. 139-143, 1976.

MAADI, M.; JAVIDNIA, M.; JAMSHIDI, R. Two Strategies Based on Meta-Heuristic Algorithms for Parallel Row Ordering Problem (PROP). **Iranian Journal of Management Studies**, v. 10, n. 2, p. 467-498, 2017.

MARTINS, P. G.; LAUGENI, F. P. **Administração da Produção**, São Paulo, Saraiva, 1998.

MIYAKE, D. I. **Arranjo Físico de Sistemas de Produção**. Escola Politécnica da USP Departamento de Engenharia de Produção, 2005. Disponível em: < https://docplayer.com.br/5369135-Arranjo-fisico-de-sistemas-de-producao-escola-politecnica-da-usp-departamento-de-engenharia-de-producao-prof-dr-dario-ikuo-miyake-2005.html>. Acesso em: 30 jun. 2021.

MOHAMMADI, M.; FORGHANI, K. Designing cellular manufacturing systems considering S-shaped layout. **Computers & Industrial Engineering**, v. 98, p. 221-236, 2016.

MURRAY, C. C.; SMITH, A. E.; ZHANG, Z. An efficient local search heuristic for the double row layout problem with asymmetric material flow. **International Journal of Production Research**, v. 51, n. 20, p. 6129-6139, 2013.

NEARCHOU, A. C. Meta-heuristics from nature for the loop layout design problem. **International Journal of Production Economics**, v. 101, n. 2, p. 312-328, 2006.

NETO, A. A. F. **Meta-Heurísticas de Otimização Tradicionais e Híbridas Utilizadas para Construção de Comitês de Classificação**. 2016, 194 f. Tese (Doutorado em Sistemas e Computação) - Universidade Federal do Rio Grande do Norte - UFRN, Natal, 2016.

OZCELIK, F. A hybrid genetic algorithm for the single row layout problem. **International Journal of Production Research**, v. 50, n. 20, p. 5872-5886, 2012.

PALUBECKIS, G. Fast local search for single row facility layout. **European Journal of Operational Research**, v. 246, n. 3, p. 800-814, 2015.

PALUBECKIS, G. Single row facility layout using multi-start simulated annealing. **Computers & Industrial Engineering**, v. 103, p. 1-16, 2017.

PÉREZ, M. A. F. **Um método heurístico para o problema de escalonamento multi-objetivo em vários ambientes de máquina**. 2012, 109 f. Dissertação (Mestrado em Engenharia de Produção) – PUC-Rio, Rio de Janeiro, 2012.

PÉREZ-GOSENDE, P.; MULA, J.; DÍAZ-MADROÑERO, M. Facility layout planning. An extended literature review. **International Journal of Production Research**, v. 59, n. 12, p. 3777-3816, 2021.

PERMANHANE, R. M. **Problemas de layout: aplicações para o CAP e para o DRLP**. 2016, 57 f. Dissertação (Dissertação em Informática) - Universidade Federal do Espírito Santo - UFES, Vitória, 2016.

PICARD, J. C.; QUEYRANNE, M. On the one-dimensional space allocation problem. **Operations Research**, v. 29, n. 2, p. 371-391, 1981.

POLI, R.; KENNEDY, J.; BLACKWELL, T. Particle swarm optimization. **Swarm Intell**, v. 1, n. 1, p. 33-57, 2007.

POLLATSCHEK, M.; GERSHONI, N.; RADDAY, Y. Optimization of the typewriter keyboard by computer simulation. **Angewandte Informatik**, v. 10, p. 438-439, 1976.

PRAIS, M.; RIBEIRO, C. C. Reactive GRASP: An application to a matrix decomposition problem in TDMA traffic assignment. **INFORMS Journal on Computing**, v. 12, p. 164-176, 2000.

QUALIS CAPES. Plataforma Sucupira, c2021. Página inicial. Disponível em: <https://sucupira.capes.gov.br/>. Acessado em: 02 de mar. de 2021.

RESENDE, M. G. C.; RIBEIRO, C. C. GRASP with Path-Relinking: Recent Advances and Applications. In: IBARAKI, T.; NONOBE, K.; YAGIURA, M. (Ed.) **Metaheuristics: Progress as Real Problem Solvers. Operations Research/Computer Science Interfaces Series**, Boston, Springer, 2005, v. 32, p. 29-63.

RESENDE, M. G. C.; RIBEIRO, C. C. Greedy Randomized Adaptive Search Procedures. In: GLOVER, F.; KOCHENBERGER, G. A. (Ed.) **Handbook of Metaheuristics. International Series in Operations Research & Management Science**, Boston, Springer, 2003, v. 57, p. 219-249.

RUBIO-SÁNCHEZ, M. et al. Grasp with path relinking for the single row facility layout problem. **Knowledge-Based Systems**, v. 106, p. 1-13, 2016.

SAMARGHANDI, H.; ESHGHI, K. An efficient tabu algorithm for the single row facility layout problem. **European Journal of Operational Research**, v. 205, n. 1, p. 98-105, 2010.

SAMARGHANDI, H.; TAABAYAN, P.; JAHANTIGH, F. F. A particle swarm optimization for the single row facility layout problem. **Computers & Industrial Engineering**, v. 58, n. 4, p. 529-534, 2010.

SECCHIN, L. D.; AMARAL, A. R. S. An improved mixed-integer programming model for the double row layout of facilities. **Optim Lett (2018)**, v. 13, n. 1, p. 193-199, 2018.

SENGUPTA, S.; BASAK, S.; PETERS II, R. A. Particle Swarm Optimization: A Survey of Historical and Recent Developments with Hybridization Perspectives. **Machine Learning & Knowledge Extraction**, v. 1, n. 1, p. 157-191, 2019.

SHI, Y.; EBERHART, R. C. Parameter selection in particle swarm optimization, In: 7TH ANNUAL CONFERENCE ON EVOLUTIONARY PROGRAMMING, 1998, New York. **Anais eletrônico...** New York, 1998, p. 591-600. Disponível em: <http://dl.acm.org/citation.cfm?id=647902.738978>. Acesso em: 22 maio 2019

SIMMONS, D. M. One-dimensional space allocation: An ordering algorithm. **Operations Research**, v. 17, n. 5, p. 812-826, 1969.

SOLIMANPUR, M.; VRAT, B.; SHANKAR, R. An Ant Algorithm for the Single Row Layout Problem in Flexible Manufacturing Systems. **Computers and Operations Research**, v. 32, n. 3, p. 583-598, 2005.

STEINBERG, L. The backboard wiring problem: A placement algorithm. **SIAM Review**, v. 3, n. 1, p. 37-50, 1961.

STORN, R.; PRICE, K. Differential evolution—A simple and efficient heuristic for global optimization over continuous spaces. **J. Glob. Optim**. 1997, v. 11, n. 4, p. 341-359, 1997.

STÜTZLE, T. **Applying iterated local search to the permutation flow shop problem**, 1998. Disponível em: <http://iridia.ulb.ac.be/~stuetzle/publications/AIDA-98-04.pdf>. Acessado em: 20 jan. 2019.

STÜTZLE, T.; RUIZ, R. **Iterated Local Search: A Concise Review**, 2018. Disponível em: <http://iridia.ulb.ac.be/IridiaTrSeries/rev/IridiaTr2018-004r001.pdf>. Acessado em: 25 jul. 2019.

SUGANTHAN, P. N. Particle swarm optimiser with neighborhood operator, In: IEEE CONGRESS ON EVOLUTIONARY COMPUTATION, 1999, Piscataway. **Anais eletrônico...** Piscataway, IEEE Press, 1999, p. 1958-1962. Disponível em: <https://doi.org/10.1109/CEC.1999.785514>. Acesso em: 16 jun. 2019.

TOMPKINS, J. A. et al. **Facility Planning**. 2. ed. New York, John Wiley & Sons Inc., 1996.

TUBAILEH, A.; SIAM, J. Single and multi-row layout design for flexible manufacturing systems. **International Journal of Computer Integrated Manufacturing**, v. 30, n. 12, p. 1316-1330, 2017.

WANG, S. et al. Solving dynamic double row layout problem via combining simulated annealing and mathematical programing. **Appl Soft Comput**, v. 37, p. 303-310, 2015.

YANG, X. et al. An improved model for the parallel row ordering problem. **Journal of the Operational Research Society**, v. 71, n. 3, p. 475-490, 2019.

YOSHIDA, H. et al. A particle swarm optimization for reactive power and voltage control in electric power systems considering voltage security assessment, In: IEEE SMC'99 CONFERENCE PROCEEDINGS (IEEE INTERNATIONAL CONFERENCE ON SYSTEMS, MAN, AND CYBERNETICS), 1999, Tokyo. **Anais eletrônico...** Tokyo, 1999, v. 6, p. 497-502. Disponível em: <https://doi.org/10.1109/ICSMC.1999.816602>. Acesso em: 15 maio 2019.

ZHANG, Z.; MURRAY, C. A Corrected Formulation for the Double Row Layout Problem. **International Journal of Production Research**, v. 50, n. 15, p. 4220-4223, 2012.

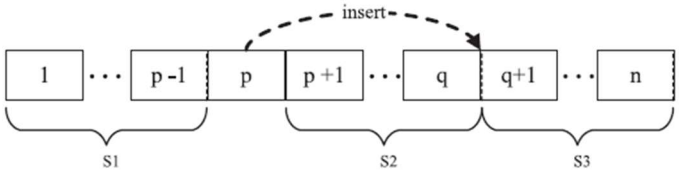
ZHOU, A. et al. Multiobjective evolutionary algorithms: A survey of the state of the art. **Swarm and Evolutionary Computation**, v. 1, n. 1, p. 32-49, 2011.

ZUO X., MURRAY C. C., SMITH A. E. Solving an extended double row layout problem using multiobjetive tabu serach and linear programming. **IEEE Trans Autom Sci Eng**, v. 11, n.4, p. 1122-1132, 2014.

APÊNDICE A – MELHORIA NO CÁLCULO DO GANHO DA INSERÇÃO

Guan e Lin (2016) apresentam um método para o cálculo do ganho na função objetivo do movimento de inserção da facilidade da posição p na posição q, quando $p<q$, com um esforço computacional de $O(n^2)$ mostrado na Equação (A.1). Para a definição da Equação (A.1), os autores apresentam uma partição da solução em conjuntos, conforme Figura 24.

Figura 24 – Conjuntos de partições no movimento de inserção, quando $p < q$



Fonte: Guan e Lin (2016)

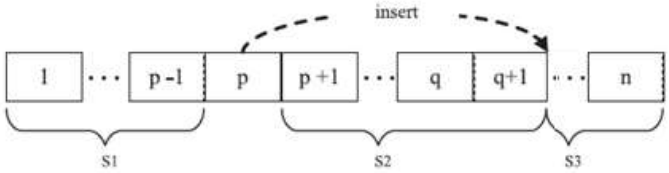
$$\Delta_{pq(p<q)} = \sum_{i \in S1} c_{ip} D_2 - \sum_{i \in S1} \sum_{j \in S2} c_{ij} l_p + \sum_{i \in S2} c_{pi}(D_2 + l_p - 2d_{pi}) - \sum_{i \in S3} c_{pi} D_2 + \sum_{i \in S2} \sum_{j \in S3} c_{ij} l_p \quad \text{(A.1)}$$

Na Equação (A.1), $S1$ contém as facilidades das posições $i < p$; $S2$ contém as facilidades das posições $p < i \leq q$; $S3$ é o conjunto das facilidades das posições $i > q$; $D_2 = \sum_{i \in S2} l_i$; l_p é o comprimento da facilidade da posição p; e d_{pi} é a distância entre as facilidades das posições p e i.

Assim, se $\Delta_{pq_{(p<q)}} < 0$, a inserção da facilidade da posição p na posição q melhora o valor da função objetivo. Caso contrário, não havendo uma melhora na função objetivo, pode-se, então, avaliar o próximo par $(p, q+1)$.

Para calcular o valor de $\Delta_{pq+1(p<q)}$ (Figura 24), as alterações serão nos conjuntos $S2$ e $S3$, pois $S2$ passará a conter $q + 1$ e $S3$ perderá o elemento $q + 1$ que está exemplificada na Figura 25. Assim, ao reescrever a Equação A.1 para calcular o ganho de $\Delta_{pq+1(p<q)}$, onde $S2' = S2 \cup \{q + 1\}$, $S3' = S3 \setminus \{q + 1\}$ e $D'_2 = D_2 + l_{q+1}$, tem-se a Equação (A.2).

Figura 25 – Conjuntos de partições no mov. de inserção (p, q+1), quando $p < q$



Fonte: próprio autor.

$$\Delta_{pq+1_{(p<q)}} = \sum_{i \in S1} c_{ip} D'_2 - \sum_{i \in S1} \sum_{j \in S2'} c_{ij} l_p + \sum_{i \in S2'} c_{pi}(D'_2 + l_p - 2d_{pi}) - \sum_{i \in S3'} c_{pi} D'_2 + \sum_{i \in S2'} \sum_{j \in S3'} c_{ij} l_p \quad \text{(A.2)}$$

Reescrevendo a Equação (A.2) com os conjuntos $S2'$, $S3'$ e D'_2 em função de $S2$, $S3$ e D_2, respectivamente, chega-se na Equação (4.4) apresentada na Seção (4.2). Sendo o desenvolvimento descrito a seguir:

Seja $S2' = S2 \cup \{q+1\}$, $S3' = S3 \setminus \{q+1\}$ e $D'_2 = D_2 + l_{q+1}$, segue:

$$\Delta_{pq+1_{(p<q)}} = \overbrace{\sum_{i \in S1} c_{ip} D'_2}^{\text{A}} - \overbrace{\sum_{i \in S1} \sum_{j \in S2'} c_{ij} l_p}^{\text{B}} + \overbrace{\sum_{i \in S2'} c_{pi}(D'_2 + l_p - 2d_{pi})}^{\text{C}} - \overbrace{\sum_{i \in S3'} c_{pi} D'_2}^{\text{D}} + \overbrace{\sum_{i \in S2'} \sum_{j \in S3'} c_{ij} l_p}^{\text{E}}$$

Calculando o termo A:

$$\text{A} = \sum_{i \in S1} c_{ip} D'_2$$

$$\text{A} = \sum_{i \in S1} c_{ip}(D_2 + l_{q+1})$$

$$\text{A} = \sum_{i \in S1} c_{ip} D_2 + \sum_{i \in S1} c_{ip} l_{q+1}$$

Calculando o termo B:

$$\text{B} = \sum_{i \in S1} \sum_{j \in S2'} c_{ij} l_p$$

$$\text{B} = \sum_{i \in S1} \left(\left(\sum_{j \in S2} c_{ij} l_p \right) + c_{iq+1} l_p \right)$$

$$B=\sum_{i\in S1}\sum_{j\in S2} c_{ij}l_p+\sum_{i\in S1} c_{iq+1}l_p$$

Calculando o termo C:

$$C=\sum_{i\in S2'} c_{pi}(D'_2+l_p-2d_{pi})$$

$$C=\left(\sum_{i\in S2} c_{pi}(D_2+l_{q+1}+l_p-2d_{pi})\right)+\left(c_{pq+1}(D_2+l_{q+1}+l_p-2d_{pq+1})\right)$$

$$C=\sum_{i\in S2} c_{pi}(D_2+l_p-2d_{pi})+\sum_{i\in S2} c_{pi}l_{q+1}+\left(c_{pq+1}(D_2+l_{q+1}+l_p-2d_{pq+1})\right)$$

Calculando o termo D:

$$D=\sum_{i\in S3'} c_{pi}D'_2$$

$$D=\left(\sum_{i\in S3} c_{pi}D'_2\right)-c_{pq+1q}D'_2$$

$$D=\left(\sum_{i\in S3} c_{pi}(D_2+l_{q+1})\right)-c_{pq+1}(D_2+l_{q+1})$$

$$D=\sum_{i\in S3} c_{pi}D_2+\sum_{i\in S3} c_{pi}l_{q+1}-c_{pq+1}(D_2+l_{q+1})$$

Calculando o termo E:

$$E=\sum_{i\in S2'}\sum_{j\in S3'} c_{ij}l_p$$

$$E=\left(\sum_{i\in S2}\sum_{j\in S3'} c_{ij}l_p\right)+\sum_{j\in S3'} c_{q+1j}l_p$$

$$E=\sum_{i\in S2}\left(\left(\sum_{j\in S3} c_{ij}l_p\right)-c_{iq+1}l_p\right)+\left(\left(\sum_{j\in S3} c_{q+1j}l_p\right)-0c_{q+1q+1}l_p\right)$$

$$E=\sum_{i\in S2}\sum_{j\in S3} c_{ij}l_p-\sum_{i\in S2} c_{iq+1}l_p+\sum_{j\in S3} c_{q+1j}l_p$$

Substituindo os termos:

$$\Delta_{pq+1_{(p<q)}} = \overbrace{\left(\sum_{i\in S1} c_{ip}D_2 + \sum_{i\in S1} c_{ip}l_{q+1}\right)}^{A} - \overbrace{\left(\sum_{i\in S1}\sum_{j\in S2} c_{ij}l_p + \sum_{i\in S1} c_{iq+1}l_p\right)}^{B} +$$

$$\overbrace{\left(\sum_{i\in S2} c_{pi}(D_2 + l_p - 2d_{pi}) + \sum_{i\in S2} c_{pi}l_{q+1} + c_{pq+1}(D_2 + l_{q+1} + l_p - 2d_{pq+1})\right)}^{C} -$$

$$\overbrace{\left(\sum_{i\in S3} c_{pi}D_2 + \sum_{i\in S3} c_{pi}l_{q+1} - c_{pq+1}(D_2 + l_{q+1})\right)}^{D} +$$

$$\overbrace{\left(\sum_{i\in S2}\sum_{j\in S3} c_{ij}l_p - \sum_{i\in S2} c_{iq+1}l_p + \sum_{j\in S3} c_{q+1j}l_p\right)}^{E}$$

Rearranjando as parcelas, ajustando e agrupando os somatórios, tem-se:

$$\Delta_{pq+1_{(p<q)}} = \left[\sum_{i\in S1} c_{ip}D_2 - \sum_{i\in S1}\sum_{j\in S2} c_{ij}l_p - \sum_{i\in S3} c_{pi}D_2 + \right.$$

$$\left. \sum_{i\in S2} c_{pi}(D_2 + l_p - 2d_{pi}) + \sum_{i\in S2}\sum_{j\in S3} c_{ij}l_p \right] +$$

$$\left[\sum_{i\in S1} (c_{ip}l_{q+1} - c_{iq+1}l_p) + \sum_{i\in S2} (c_{pi}l_{q+1} - c_{iq+1}l_p) + \right.$$

$$\left. \sum_{i\in S3} (c_{q+1i}l_p - c_{pi}l_{q+1}) + c_{pq+1}(2(D_2 + l_{q+1}) + l_p - 2d_{pq+1}) \right]$$

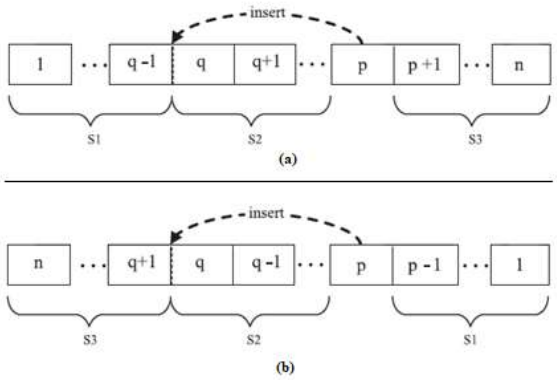
Que é a Equação (4.4) (ver Seção 4.2).

Para o movimento de inserção da facilidade da posição p na posição q quanto $p > q$, pode-se visualizar o movimento de inserção para $p > q$ pela

Figura 26(a). A

Figura 26(b) apresenta o arranjo da Figura 24 invertido.

Figura 26 – Conjuntos de partições no movimento de inserção, quando $p > q$



Fonte: próprio autor

Assim, o cálculo do valor de $\Delta_{pq_{(p>q)}}$ pode ser realizado pela Equação (A.1) considerando a inversão no arranjo; e se $\Delta_{pq_{(p>q)}}<0$, a inserção da facilidade da posição p na posição q melhora o valor da função objetivo. Caso contrário, não haverá uma melhora na função objetivo e pode-se, então, avaliar o próximo par (p, q+1).

Observando a

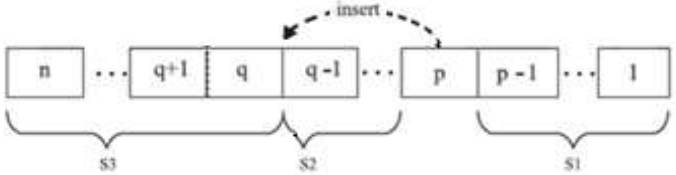
Figura 26 nota-se que a disposição das facilidades no cálculo do valor do ganho no movimento de inserção de p na posição p+1, ou seja, $\Delta_{pq+1_{(p>q)}}$ (

Figura 26a), é igual à do valor da inserção de p na posição q–1, $\Delta_{pq-1_{(p<q)}}$ (

Figura 26b), se o arranjo de facilidades for invertido, o que representa a mesma solução. Assim, pode-se realizar o cálculo de $\Delta_{pq+1_{(p>q)}}$ como o cálculo de $\Delta_{pq-1_{(p<q)}}$. Neste caso, pela a

Figura 26b, as alterações serão nos conjuntos $S2$ e $S3$, pois $S2$ perderá q e $S3$ passa a conter o elemento q, exemplificado na Figura 27. Então, ao reescrever a Equação (A.1) para calcular o ganho de $\Delta_{pq-1_{(p<q)}}$ sendo $S3'=S3 \cup \{q\}$, $S2'=S2 \setminus \{q\}$ e $D'_2=D_2-l_q$, tem-se a Equação (A.3).

Figura 27 – Conjuntos de partições no mov. de inserção (p, q+1), quando $p > q$



Fonte: próprio autor

$$\Delta_{pq\text{-}1_{(p<q)}} = \sum_{i \in S1} c_{ip} D'_2 - \sum_{i \in S1} \sum_{j \in S2'} c_{ij} l_p + \sum_{i \in S2'} c_{pi}(D'_2 + l_p - 2d_{pi}) - \sum_{i \in S3'} c_{pi} D'_2 + \sum_{i \in S2'} \sum_{j \in S3'} c_{ij} l_p \quad \text{(A.3)}$$

Reescrevendo a Equação (A.3) com os conjuntos $S2'$ e $S3'$; e D'_2 em relação à $S2$ e $S3$; e D_2, respectivamente, chega-se à Equação (4.5) mostrada na Seção (4.2). A qual, o desenvolvimento pode ser verificado a seguir:

Seja $S2' = S2 \setminus \{q\}$, $S3' = S3 \cup \{q\}$ e $D'_2 = D_2 - l_q$, segue:

$$\Delta_{pq\text{-}1_{(p<q)}} = \overbrace{\sum_{i \in S1} c_{ip} D'_2}^{A} - \overbrace{\sum_{i \in S1} \sum_{j \in S2'} c_{ij} l_p}^{B} + \overbrace{\sum_{i \in S2'} c_{pi}(D'_2 + l_p - 2d_{pi})}^{C} - \overbrace{\sum_{i \in S3'} c_{pi} D'_2}^{D} + \overbrace{\sum_{i \in S2'} \sum_{j \in S3'} c_{ij} l_p}^{E}$$

Calculando o termo A:

$$A = \sum_{i \in S1} c_{ip} D'_2$$

$$A = \sum_{i \in S1} c_{ip}(D_2 - l_q)$$

$$A = \sum_{i \in S1} c_{ip} D_2 - \sum_{i \in S1} c_{ip} l_q$$

Calculando o termo B:

$$B = \sum_{i \in S1} \sum_{j \in S2'} c_{ij} l_p$$

$$B = \sum_{i \in S1} \left(\left(\sum_{j \in S2} c_{ij} l_p \right) - c_{iq} l_p \right)$$

$$B = \sum_{i \in S1} \sum_{j \in S2} c_{ij} l_p - \sum_{i \in S1} c_{iq} l_p$$

Calculando o termo C:

$$C=\sum_{i\in S2'} c_{pi}(D'_2+l_p-2d_{pi})$$

$$C=\left(\sum_{i\in S2} c_{pi}(D_2-l_q+l_p-2d_{pi})\right)-(c_{pq}(D_2-l_q+l_p-2d_{pq}))$$

$$C=\sum_{i\in S2} c_{pi}(D_2+l_p-2d_{pi})-\sum_{i\in S2} c_{pi}l_q-(c_{pq}(D_2-l_q+l_p-2d_{pq}))$$

Calculando o termo D:

$$D=\sum_{i\in S3'} c_{pi}D'_2$$

$$D=\left(\sum_{i\in S3} c_{pi}D'_2\right)+c_{pq}D'_2$$

$$D=\left(\sum_{i\in S3} c_{pi}(D_2-l_q)\right)+c_{pq}(D_2-l_q)$$

$$D=\sum_{i\in S3} c_{pi}D_2-\sum_{i\in S3} c_{pi}l_q+c_{pq}(D_2-l_q)$$

Calculando o termo E:

$$E=\sum_{i\in S2'}\sum_{j\in S3'} c_{ij}l_p$$

$$E=\left(\sum_{i\in S2}\sum_{j\in S3'} c_{ij}l_p\right)-\sum_{j\in S3'} c_{qj}l_p$$

$$E=\sum_{i\in S2}\left(\left(\sum_{j\in S3} c_{ij}l_p\right)+c_{iq}l_p\right)+\left(\left(\sum_{j\in S3} c_{qj}l_p\right)+0c_{qq}l_p\right)$$

$$E=\sum_{i\in S2}\sum_{j\in S3} c_{ij}l_p+\sum_{i\in S2} c_{iq}l_p+\sum_{j\in S3} c_{qj}l_p$$

Substituindo os termos:

$$\Delta_{pq\text{-}1_{(p<q)}}=\overbrace{\left(\sum_{i\in S1} c_{ip}D_2-\sum_{i\in S1} c_{ip}l_q\right)}^{A}-\overbrace{\left(\sum_{i\in S1}\sum_{j\in S2} c_{ij}l_p-\sum_{i\in S1} c_{iq}l_p\right)}^{B}+$$

$$\overbrace{\left(\sum_{i\in S2} c_{pi}(D_2+l_p-2d_{pi})-\sum_{i\in S2} c_{pi}l_q-(c_{pq}(D_2-l_q+l_p-2d_{pq}))\right)}^{C}-$$

$$\overbrace{\left(\sum_{i\in S3} c_{pi}D_2 - \sum_{i\in S3} c_{pi}l_q + c_{pq}(D_2 - l_q)\right)}^{\text{D}} +$$

$$\overbrace{\left(\sum_{i\in S2}\sum_{j\in S3} c_{ij}l_p + \sum_{i\in S2} c_{iq}l_p + \sum_{j\in S3} c_{qj}l_p\right)}^{\text{E}}$$

Rearranjando as parcelas, ajustando e agrupando os somatórios, tem-se:

$$\Delta_{pq\text{-}1_{(p<q)}} = \left[\sum_{i\in S1} c_{ip}D_2 - \sum_{i\in S1}\sum_{j\in S2} c_{ij}l_p - \sum_{i\in S3} c_{pi}D_2 + \right.$$

$$\left.\sum_{i\in S2} c_{pi}(D_2 + l_p - 2d_{pi}) + \sum_{i\in S2}\sum_{j\in S3} c_{ij}l_p\right] +$$

$$\left[\sum_{i\in S1} (c_{iq}l_p - c_{ip}l_q) + \sum_{i\in S2} (c_{iq}l_p - c_{pi}l_q) + \right.$$

$$\left.\sum_{i\in S3} (c_{qi}l_p - c_{pi}l_q) - c_{pq}(2(D_2 - l_q) + l_p - 2d_{pq+1})\right]$$

Que é a Equação (4.5) (ver Seção 4.2).

APÊNDICE B – CALIBRAÇÃO DOS PARÂMETROS DOS ALGORITMOS

Este apêndice apresenta as implementações da meta-heurística PSO utilizada para a calibração dos algoritmos propostos nesta tese.

Com o objetivo de encontrar os melhores valores para os parâmetros dos algoritmos, utilizou-se uma implementação padrão da meta-heurística PSO. Nesse PSO-calibrador, inicialmente um grupo com m partículas (soluções) são geradas aleatoriamente (soluções) e então novas soluções são encontradas por buscas repetitivas. Como dito anteriormente (Seção 3.2), cada partícula i tem um vetor posição x_i e uma velocidade representada por um vetor v_i em um espaço multidimensional. Todas as partículas lembram de seu melhor vetor posição p_i e conhecem o melhor vetor posição global g. O vetor v_i e o vetor x_i são atualizados de acordo com as Equações (3.3) e (3.2). As partículas são avaliadas por uma função de aptidão (*fitness*), que no caso da calibração, é o algoritmo que está sendo calibrado. A função de aptidão avalia o vetor posição x_i e retorna o valor de aptidão da partícula i. Então, para o PSO são definidos os componentes do vetor posição x_i como sendo os parâmetros dos algoritmos a serem calibrados.

A partícula i armazena um vetor de aptidão h_i, que contém os valores encontrados pelo algoritmo que está sendo calibrado quando executado com os parâmetros x_i. Os componentes do vetor de aptidão h_i são listados a seguir:

- *fo*: valor da função objetivo;
- *IterMAX*: máximo de iterações;
- *TotalTime*: tempo total (em segundos);
- *IterLastImprov*: iteração que ocorreu a última melhora na função objetivo; e
- *TimeLastImprov*: tempo (em segundos) que ocorreu a última melhora na função objetivo.

Para decidir qual é a melhor partícula, foi definido o Algoritmo 23, que compara as aptidões entre duas partículas.

Algoritmo 23 – Função de Comparação de Aptidão – PSO calibração

entrada: $x1$, $h1$, $x2$, $h2$
saída: x, h
1: $aux_1 \leftarrow h1_{IterMAX} + h1_{IterLastImprov} + h1_{TimeLastImprov} + h1_{TotalTime}$
2: $aux_2 \leftarrow h2_{IterMAX} + h2_{IterLastImprov} + h2_{TimeLastImprov} + h2_{TotalTime}$
3: **Se** $h1_{fo} < h2_{fo}$ **então**
4: $x \leftarrow x1$, $h \leftarrow h1$
5: **Senão Se** $h2_{fo} < h1_{fo}$ **então**
6: $x \leftarrow x2$, $h \leftarrow h2$
7: **Senão Se** $aux_1 < aux_2$ **então**
8: $x \leftarrow x1$, $h \leftarrow h1$
9: **Senão**
10: $x \leftarrow x2$, $h \leftarrow h2$
11: **Fim**
12: **retorne** x, h

A função de comparação de aptidão recebe como entrada dois vetores posição e seus vetores aptidão correspondentes. Primeiramente, são comparados os valores da função objetivo encontrados, relacionados aos dois vetores posição, Linha 3 e 5, registrando como x o vetor posição com o melhor valor de função objetivo (problema de minimização) e registrando como h o vetor de aptidão associado ao x. No caso de empate nos valores de *fo*, os valores de aux_1 e aux_2, calculados nas Linhas 1 e 2, são usados como critério secundário para identificar o melhor vetor posição x e seu vetor de aptidão associado h (Linhas 7-11). A função de comparação de aptidão retorna o valor de x e h (Linha 12).

O pseudocódigo da implementação do PSO para calibração é mostrado no Algoritmo 24. O algoritmo recebe como parâmetros: *Algoritmo*, o algoritmo a ser calibrado; *m*, número de partículas; e *IterMax*, número máximo de iterações.

Algoritmo 24 – PSO para Calibração dos Parâmetros

entrada: *Algoritmo*, *m*, *IterMax*
saída: *g*
1: $g \leftarrow (0, 0, 0, 0)$, $h \leftarrow (\infty, \infty, \infty, \infty)$
2: **Para** $i \leftarrow 1$ **até** m **faça**
3: Inicializa as posições x_i e a velocidade v_i da partícula i
4: $h_i \leftarrow$ *Algoritmo* (x_i)
5: $[p_i, \overline{h_i}] \leftarrow [x_i, h_i]$
6: $[g, h] \leftarrow$ *Algoritmo 23* (x_i, h_i, g, h)
7: **Fim**
8: **Para** $t \leftarrow 0$ **até** *IterMax*-1 **faça**
9: $\omega \leftarrow \omega_{max} - (\omega_{max} - \omega_{min}) \times (\frac{t}{IterMax})$
10: **Para** $i \leftarrow 1$ **até** m **faça**
11: $v_i \leftarrow \omega v_i + r_1 c_1 (p_i - x_i) + r_2 c_2 (g - x_i)$
12: $x_i \leftarrow x_i + v_i$
13: $h_i \leftarrow$ *Algoritmo* (x_i)
14: $[p_i, \overline{h_i}] \leftarrow$ *Algoritmo 23* $(x_i, h_i, p_i, \overline{h_i})$
15: $[g, h] \leftarrow$ *Algoritmo 23* $(p_i, \overline{h_i}, g, h)$
16: **Fim**
17: **Fim**
18: **retorne** g

No laço das Linhas 2-7, a posição x_i e a velocidade v_i de cada partícula i (i=1, ..., m) são inicializadas aleatoriamente (Linha 3); a aptidão h_i é calculada (Linha 4); e p_i e o vetor $\overline{h_i}$ são inicializados (Linha 5). No final do laço, g e sua aptidão associada h serão determinados pela chamada da função de comparação de aptidão, Algoritmo 23 (Linha 6). O laço das Linhas 8-17 é executado *IterMax* iterações. Na Linha 9, o valor de ω é atualizado. Nas Linhas 10 a 16, a informação de cada partícula i é atualizada, de acordo com as Equações (3.3) e (3.2): o vetor v_i (Linha 11) e x_i (Linha 12). Implicitamente, na Linha 12, os limites dos novos valores das componentes do vetor x_i são verificados, dependendo da configuração definida para o algoritmo que está sendo calibrado. Caso os limites sejam extrapolados, um novo valor para a componente é gerado aleatoriamente dentro do limite definido. O vetor h_i é determinado na Linha 13. Os vetores p_i e $\overline{h_i}$ (Linha 14) e os vetores g e h (Linha 15) são atualizados pela comparação com a nova posição da partícula i, usando a função de comparação de aptidão. No final, o procedimento retorna a melhor posição encontrada, g (Linha 18).

Para a execução do PSO de calibração foi utilizado o fator ω (fator de inércia), linearmente variável com o tempo (SENGUPTA; BASAK; PETERS II, 2018). O algoritmo então, inicia com ω_{max} e a cada iteração, o valor ω é atualizado até um valor limite ω_{min}. Assim, como definido em Abraham, Guo e Liu (2006), os valores utilizados para os parâmetros do PSO de calibração foram: ω_{max} = 0,9 e ω_{min} = 0,4; para as constantes c_1 e c_2, foram utilizados $c_1 = c_2 =$ 2; e os valores de r_1 e r_2 são escolhidos aleatoriamente.