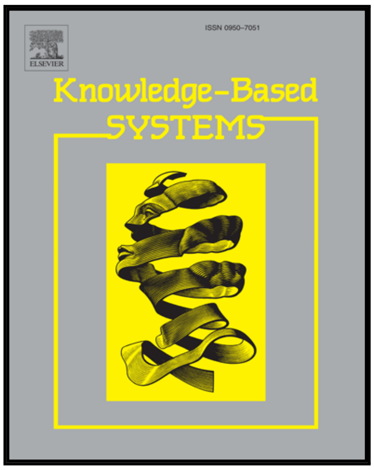
Manuscrito aceito



GRASP com revinculação de caminho para o problema de layout de instalação de linha única

Manuel Rubio-Sá´nchez, Micael Gallego, Francisco Gortazar,
Abraham Duarte

PII: S0950-7051 (16) 30120-4
DOI: 10.1016 / j.knosys.2016.05.030
Referência: KNOSYS 3527

Aparecer em: *Sistemas Baseados em Conhecimento*

Data de recebimento: 9 de outubro de 2015
Data de revisão: 12 de maio de 2016
Data de aceitação: 13 de maio de 2016

Como citar este artigo: Manuel Rubio-Sá´nchez, Micael Gallego, Francisco Gortazar, Abraham Duarte, GRASP com revinculação de caminho para o problema de layout de instalação de linha única, *Sistemas Baseados em Conhecimento (* 2016), doi: 10.1016 / j.knosys.2016.05.030

PEGA COM CAMINHO RELINKING PARA LINHA ÚNICA PROBLEMA DE LAYOUT DE INSTALAÇÃO

MANUEL RUBIO-SÁ ´NCHEZ, MICAEL GALLEGO, FRANCISCO GORTAZAR,
E ABRAHAM DUARTE?

Abstrato. O problema de layout de instalação de linha única (SRFLP) é um *NP*- Problema difícil que consiste em encontrar um arranjo ótimo de um conjunto de instalações retangulares (com altura igual e comprimentos diferentes), colocando-as próximas umas das outras ao longo de uma linha. O SRFLP tem aplicações práticas em contextos como disposição de salas ao longo de corredores, colocação de livros em estantes, distribuição de informações em discos magnéticos, armazenamento de itens em depósitos ou criação de layouts para máquinas em sistemas de manufatura. Este artigo combina a metodologia greedy randomized adaptive search procedure (GRASP) e path relinking (PR) a fim de buscar soluções de alta qualidade para o SRFLP de maneira eficiente. Em particular, apresentamos: (i) vários procedimentos de construção, (ii) uma nova estratégia de busca local rápida e (iii) uma abordagem relacionada à distância de Ulam para construir trajetórias de religação de caminho curto. Também apresentamos um novo conjunto de grandes instâncias desafiadoras, uma vez que conjuntos anteriores não permitem determinar diferenças significativas entre metaheurísticas avançadas. Os experimentos mostram que nosso procedimento supera os métodos de última geração em todos os cenários que consideramos. Em primeiro lugar, o GRASP com PR encontra as soluções mais conhecidas para instâncias anteriores utilizadas na literatura, mas empregando consideravelmente menos tempo de computação do que seus concorrentes. Em segundo lugar, nosso método supera os métodos atuais de última geração em 38 das 40 novas instâncias quando executado pela mesma quantidade de tempo de computação. Finalmente, os testes não paramétricos para detectar diferenças entre relatórios de algoritmos Os experimentos mostram que nosso procedimento supera os métodos de última geração em todos os cenários que consideramos. Em primeiro lugar, o GRASP com PR encontra as soluções mais conhecidas para instâncias anteriores utilizadas na literatura, mas empregando consideravelmente menos tempo de computação do que seus concorrentes. Em segundo lugar, nosso método supera os métodos atuais de última geração em 38 das 40 novas instâncias quando executado pela mesma quantidade de tempo de computação. Finalmente, os testes não paramétricos para detectar diferenças entre relatórios de algoritmos Os experimentos mostram que nosso procedimento supera os métodos de última geração em todos os cenários que consideramos. Em primeiro lugar, o GRASP com PR encontra as soluções mais conhecidas para instâncias anteriores utilizadas na literatura, mas empregando consideravelmente menos tempo de computação do que seus concorrentes. Em segundo lugar, nosso método supera os métodos atuais de última geração em 38 das 40 novas instâncias quando executado pela mesma quantidade de tempo de computação. Finalmente, os testes não paramétricos para detectar diferenças entre relatórios de algoritmos nosso método supera os métodos atuais de última geração em 38 das 40 novas instâncias quando executado pela mesma quantidade de tempo de computação. Finalmente, os testes não paramétricos para detectar diferenças e

1 Introdução

Os problemas de localização de instalações estão relacionados a encontrar localizações ideais de instalações (máquinas-ferramentas, centros de trabalho, células de manufatura, oficinas de máquinas, etc.) em uma determinada área. Sua função objetivo pode refletir vários tipos de custos (por exemplo, transporte, transmissão ou comunicação), ou simplesmente preferências de adjacência entre máquinas. Neste artigo, nos concentramos no problema de layout de instalação de linha única (SRFLP), também conhecido como o problema de alocação de espaço unidimensional (Simmons, 1969). Tem sido aplicado em vários domínios a fim de resolver problemas relacionados com a disposição de quartos ao longo de corredores (por exemplo, hospitais ou edifícios de escritórios), colocação de livros nas prateleiras, alocação de informações em discos magnéticos, armazenamento de itens em armazéns ou criação de layouts para máquinas em sistemas de manufatura (Simmons, 1969; Picard e Queyranne, 1981; Heragu e Kusiak, 1988).

O SRFLP é um *NP*- Problema difícil que consiste em encontrar um arranjo ótimo de um conjunto de instalações retangulares, colocando-as próximas umas das outras ao longo de uma linha.

Palavras-chave e frases. Metaheurísticas, planejamento e projeto de instalações, GRASP, reconexão de caminhos.

? Autor correspondente: Manuel Rubio-S´ ánchez (manuel.rubio@urjc.es).

Comprimentos das instalações:

eu	1	2	3	4	5
eu_{eu}	5	3	6	4	2

Pesos entre instalações:

$c_{eu\,j}$	1	2	3	4	5
1	0	5	2	4	1
2	5	0	3	0	2
3	2	3	0	0	0
4	4	0	0	0	5
5	1	2	0	5	0

Figura 1. Instância de tamanho $n = 5$ da SRFLP, definida por meio de uma lista eu de comprimentos de instalação, e uma matriz de peso quadrada simétrica c.

Em particular, o objetivo é obter uma ordenação das instalações que minimize a soma ponderada das distâncias entre os centros de todos os pares de instalações. Formalmente, o SRFLP é definido da seguinte forma: $F = \{1, 2, . . . , n\}$ ser um conjunto de $n > 2$ instalações retangulares com altura fixa, mas comprimentos diferentes $eu_{eu} > 0$, para $eu \in F$. De Anúncios-tradicionalmente, deixe $c_{ij} = c_{ji} \geq 0$, para $eu\ j \in F$, seja o peso entre as instalações eu e j, que geralmente modela algum custo de transmissão entre eles. Uma solução particular para este problema é um pedido $\pi = \langle \pi(1), \pi(2), . . . , \pi(n) \rangle$ das instalações em F, cujo custo $C(\pi)$ é definido de acordo com a seguinte função objetivo:

$$C(\pi) = \sum_{1 \leq q < r \leq n} c_{\pi(q)\pi(r)} \cdot d_{\pi(q)\pi(r)}, \tag{1}$$

Onde $d_{\pi(q)\pi(r)}$ representa a distância entre os centros das instalações $\pi(q)$ e $\pi(r)$ (ou seja, localizado no pedido π em posições q e r, respectivamente), e é calculado como:

$$d_{\pi(q)\pi(r)} = eu_{\pi(q)}/2 + \sum_{q<s<r} eu_{\pi(s)} + eu_{\pi(r)}/2$$

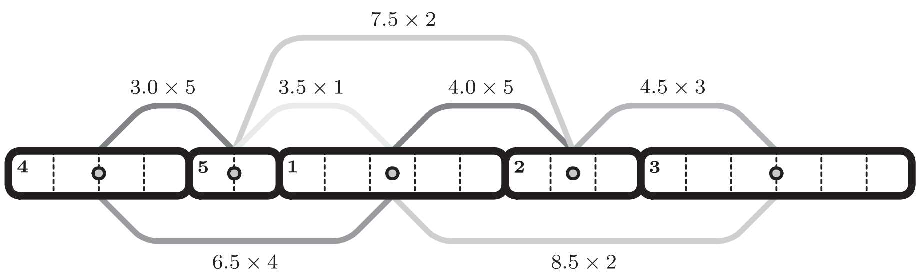
O problema de otimização, portanto, consiste em encontrar uma ordem $\pi?$ que minimiza (1). Formalmente:

$$\pi? = \operatorname*{argmin}_{\pi \in \Pi_n} C(\pi),$$

onde Π_n é o conjunto de permutações do primeiro n inteiros positivos.

Uma instância do SRFLP de tamanho $n = |F|$ é, portanto, definido especificando o Lista eu de comprimentos de instalação (de tamanho n), e um simétrico $n \times n$ matriz de custo c que contém os pesos entre as instalações. A Figura 1 mostra uma instância de tamanho $n = 5$. Nota que o comprimento da instalação (eu_{eu}) variam de 2 (instalação 5) a 6 (instalação 3). Além disso, os elementos diagonais de c são normalmente definidos como 0, pois a função objetivo não os considere, mas outros pesos fora da diagonal também podem ser iguais a 0, indicando que um par não contribui para a função de custo, independentemente da colocação ao longo de uma ordem das instalações particulares.

A Figura 2 ilustra uma solução para a instância descrita na Figura 1. Ela é representada como o pedido $\pi = \langle 4, 5, 1, 2, 3 \rangle$, o que significa que a facilidade 4 é colocada na primeira posição do pedido (ou seja, $\pi(1) = 4$), seguido pela facilidade 5 (ou seja, $\pi(2) = 5$) e assim por diante. A distância entre um par de instalações em uma determinada ordem π envolve computar a distância entre seus centros. No exemplo, a distância entre as instalações 4 e 5 são $d_{45} = (1/2) \times eu_4 + (1/2) \times eu_5 = 3$. Da mesma forma, a distância entre instalação 5 e 3 é $d_{53} = (1/2) \times eu_5 + eu_1 + eu_2 + (1/2) \times eu_3 = 12$. A contribuição de um par de instalações para a função objetivo é calculada como o produto da distância e peso entre as instalações. Por exemplo, a contribuição do



Solution: $\pi = \langle 4, 5, 1, 2, 3\rangle$

Cost: $C(\pi) = (3.0 \times 5) + (6.5 \times 4) + (3.5 \times 1) + (7.5 \times 2) + (4.0 \times 5) + (8.5 \times 2) + (4.5 \times 3) = 110$

Solução: $\pi = \langle 4, 5, 1, 2, 3 \rangle$

Custo: $C(\pi) = (3{,}0 \times 5) + (6{,}5 \times 4) + (3{,}5 \times 1) + (7{,}5 \times 2) + (4{,}0 \times 5) + (8{,}5 \times 2) + (4{,}5 \times 3) = 110$

Figura 2. Exemplo de solução $\pi = \langle 4, 5, 1, 2, 3 \rangle$ para a instância definida na Figura 1, cujo custo é 110. As contribuições (positivas) de cada par de instalações para a função objetivo do problema são ilustradas ao lado de linhas dobradas cuja cor está associada ao valor do peso entre as instalações (em particular, pesos maiores são representados por tons de cinza mais escuros).

par (4, 5) é $d_{45} \times c_{45} = 3{,}0 \times 5 = 15$. Da mesma forma, para as instalações 5 e 3, o contribution é $d_{53} \times c_{53} = 12 \times 0 = 0$. Na figura, ilustramos essas contribuições numéricas ao lado de linhas curvas conectando os pares de instalações associados. Dentro Em particular, o tom de cinza de uma linha depende do peso entre as instalações (pesos maiores são representados por tons de cinza mais escuros). Por fim, por uma questão de clareza, incluímos apenas contribuições diferentes de zero no custo.

Neste artigo, propomos uma adaptação da metodologia greedy randomized adaptive search procedure (GRASP) (Feo e Resende, 1989) para buscar de forma eficiente soluções de alta qualidade para o SRFLP. Em particular, introduzimos vários procedimentos construtivos com uma negociação diferente entre intensificação e diversificação. Além disso, apresentamos uma nova estratégia de busca local rápida que se baseia em uma abordagem híbrida entre os métodos clássicos de busca de primeiro e melhor aperfeiçoamento. Adicionalmente, apresentamos uma estratégia de pós-processamento baseada na reconexão de caminhos (Glover, 1998), onde construímos trajetórias de caminhos curtos por meio de uma abordagem relacionada à distância de Ulam entre permutações (Ulam, 1972). Por último,

O restante deste trabalho está organizado da seguinte forma. A seção 2 descreve as abordagens mais relevantes apresentadas na literatura relacionada. A abordagem GRASP é introduzida na Seção 3, onde nos concentramos na descrição dos algoritmos de busca construtiva e local propostos. A seção 4 descreve várias estratégias para gerar trajetórias no contexto de religação de caminho, enquanto a seção 5 apresenta a combinação proposta de GRASP e religação de caminho. Finalmente, a Seção 6 relata os resultados dos experimentos computacionais, e a Seção 7 resume as conclusões mais relevantes.

2 Revisão da literatura

Existe uma extensa literatura relacionada à família de problemas de localização de instalações. Entre eles, o SRLFP surge atualmente como um dos problemas mais ativos. Pesquisas sobre o estado da arte do SRFLP estão disponíveis em Kothari e Ghosh (2012a); Keller e Buscher (2015). Os pesquisadores desenvolveram algoritmos exatos para resolver o problema que incluem estratégias branch-and-bound (Simmons, 1969), programação dinâmica (Picard e Queyranne, 1981), programação linear inteira mista (Love e Wong, 1976; Heragu e Kusiak, 1991; Amaral, 2006; 2008), ramal e corte (Amaral e Letchford, 2013), corte de planos (Amaral, 2009; Sanjeevi e Kianfar, 2010), programação semide fi nita (Anjos et al., 2005; Anjos e Yen, 2009; Hungerl¨ änder e Rendl, 2013), ou uma combinação das duas últimas abordagens (Anjos e Vannelli, 2008). Atualmente, soluções ótimas são conhecidas para instâncias de no máximo 42 instalações. Para instâncias maiores, pesquisas recentes têm se concentrado no emprego de metaheurísticas eficientes para buscar soluções aproximadas de alta qualidade, uma vez que métodos exatos são atualmente proibitivos do ponto de vista computacional. Finalmente, métodos exatos também foram propostos para uma variante em que todas as instalações têm o mesmo comprimento (Hungerl¨änder, 2014).

De uma perspectiva heurística, os procedimentos de construção são os mais simples. Esses algoritmos consideram ou os pesos entre as instalações (Heragu e Kusiak, 1988; Kumar et al., 1995), ou o comprimento das instalações (Samarghandi e Eshghi, 2010), e obtêm soluções de forma muito eficiente. No entanto, sua qualidade não é aceitável para aplicações reais. Portanto, essas abordagens são geralmente usadas como parte de metaheurísticas mais sofisticadas. Por exemplo, a abordagem de construção proposta em Samarghandi e Eshghi (2010) foi usada dentro de uma estrutura Tabu Search, para uma estratégia multi-start que melhora cada solução construída com uma vizinhança de inserção Lin-Kernigham (Kothari e Ghosh, 2013a), ou como uma semente inicial para o estágio de diversificação da abordagem de scatter search em Kothari e Ghosh (2014b).

Uma variedade de outras metaheurísticas também foi usada para lidar com o SRFLP. Isso inclui algoritmos genéticos (Datta et al., 2011; Kothari e Ghosh, 2014a), enxame de partículas (Samarghandi et al., 2010) e otimização de colônia de formigas (Solimanpur et al., 2005), recozimento simulado (Romero e S´ánchez-Flores, 1990; Heragu e Alfa, 1992), ou pesquisa dispersa (Kumar et al., 2008; Kothari e Ghosh, 2014b). Além disso, algumas abordagens combinam diferentes estratégias metaheurísticas. Por exemplo, o recozimento simulado é acoplado a algoritmos genéticos em Ramkumar e Ponnambalam (2004), enquanto em Kothari e Ghosh (2012b) o relink de caminho é aplicado a soluções geradas pelas abordagens em Kothari e Ghosh (2013a; b; 2014b).

Até onde sabemos, as melhores abordagens para encontrar soluções aproximadas para o SRFLP por meio de metaheurísticas são as apresentadas em Kothari e Ghosh (2014a) e Kothari e Ghosh (2014b). Especificamente, o primeiro trabalho apresenta um algoritmo genético direto que usa o operador de cruzamento parcialmente correspondido (PMX) (ver Larrañ ˜aga et al. (1999) para uma descrição detalhada deste operador), e mutações baseadas em movimentos de inserção. Além disso, melhora as soluções executando um algoritmo de busca local baseado em movimentos de inserção e considerando a melhor estratégia de melhoria. O segundo artigo propõe um método de busca dispersa que toma emprestados componentes do primeiro. Por exemplo, ele também usa o operador PMX para combinar soluções e usa a mesma estratégia exaustiva de busca local. O artigo usa uma abordagem adicional de combinação alternada e propõe dois

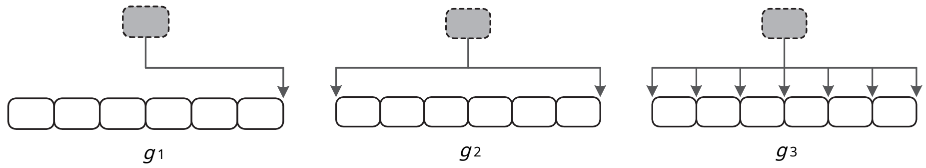


Figura 3. Funções gananciosas. Uma nova instalação pode ser incorporada a uma solução parcial, anexando-a a um extremo do se- quence (g_1), até o melhor dos dois extremos (g_2), ou inserindo-o no melhor local possível (g_3).

estratégias de diversi fi cação para gerar um conjunto inicial de soluções de referência. O algoritmo genético e a abordagem de scatter search foram capazes de obter 42 e 41 das melhores soluções encontradas até agora, respectivamente, nas 43 instâncias de benchmark utilizadas na literatura recente. Isso indica a necessidade de analisar os algoritmos em instâncias maiores e mais desafiadoras, uma vez que os algoritmos freqüentemente encontram as mesmas soluções, que são provavelmente as melhores. Por último, o desempenho ligeiramente melhor do algoritmo genético pode ser devido ao fato de que ele emprega consideravelmente mais tempo de computação.

3 APERTO

A metodologia GRASP é uma metaheurística desenvolvida no final dos anos 1980 (Feo e Resende, 1989) e formalmente introduzida em Feo et al. (1994). Levantamentos recentes e completos do método são apresentados em Resende e Ribeiro (2010; 2014). GRASP é uma metodologia multi-start em que cada iteração consiste em dois estágios. O primeiro executa uma construção gananciosa, aleatória e adaptativa de uma solução. A segunda etapa melhora a solução construída (de acordo com a função objetivo do problema) aplicando um procedimento de busca até atingir um ótimo local. Essas duas etapas são repetidas até que um critério de encerramento seja atendido. O restante desta seção é organizado da seguinte forma. A Seção 3.1 apresenta procedimentos construtivos para o SRFLP, enquanto a Seção 3.2 descreve algoritmos de busca local.

3.1. Métodos construtivos. Um procedimento construtivo GRASP constrói uma solução π do tamanho n para o SRFLP iterativamente, inserindo instalações uma de cada vez (por meio estratégias aleatórias e gananciosas) em uma solução parcial π_p contendo menos que n instalações. Para este propósito, propomos as três funções gulosas ilustradas na Figura 3. O primeiro, denotado como g_1, acrescenta novas instalações unidirecionalmente (por exemplo, da esquerda a direita) para apenas um extremo da sequência parcial. Uma segunda abordagem, chamada g_2, considera anexar uma facilidade a qualquer um dos extremos. Finalmente, uma terceira estratégia, denotada como g_3, contempla a possibilidade de inserir uma instalação em qualquer local ao longo da solução parcial.

A fim de selecionar qual instalação incorporar, bem como sua localização ao longo π_p ao usar g_2 ou g_3, contamos com o fato de que o custo de uma solução parcial com $m \leq n$ as instalações podem ser avaliadas. Em particular, medimos o custo de uma solução de acordo com a função objetivo do problema, mas considerando apenas as facilidades que pertencem à seqüência parcial. Formalmente:

(2) $$C(\pi_p) = \sum_{1 \leq q < r \leq m} c_{\pi_p(q)\pi_p(r)} \cdot d_{\pi_p(q)\pi_p(r)}.$$

Algoritmo 1: Greedy-Random (GR)

1: $CL \leftarrow F$
2: $eu \leftarrow$ SelectRandom (CL)
3: $\pi_p \leftarrow \langle eu \rangle$
4: $CL \leftarrow CL \setminus \{ eu\}$
5: enquanto $CL \neq \emptyset$ Faz
6: $\quad$ $Tamanho \leftarrow \max \{ \lfloor \alpha \cdot | CL | \rfloor, 1\}$
7: $\quad$ $RCL \leftarrow$ SelectBest ($CL, \pi_p, Tamanho$)
8: $\quad$ $eu? \leftarrow$ SelectRandom (RCL)
9: $\quad$ $\pi_p \leftarrow$ IncludeFacility ($\pi_p, eu?$)
10: $\quad$ $CL \leftarrow CL \setminus \{ eu?\}$
11: terminar enquanto
12: Retorna π_p

Algoritmo 2: Random-Greedy (RG)

1: $CL \leftarrow F$
2: $eu \leftarrow$ SelectRandom (CL)
3: $\pi_p \leftarrow \langle eu \rangle$
4: $CL \leftarrow CL \setminus \{ eu\}$
5: enquanto $CL \neq \emptyset$ Faz
6: $\quad$ $Tamanho \leftarrow \max \{ \lfloor \alpha \cdot | CL | \rfloor, 1\}$
7: $\quad$ $RCL \leftarrow$ SelectRandom ($CL, Tamanho$)
8: $\quad$ $eu? \leftarrow$ SelectBest (RCL, π_p)
9: $\quad$ $\pi_p \leftarrow$ IncludeFacility ($\pi_p, eu?$)
10: $\quad$ $CL \leftarrow CL \setminus \{ eu?\}$
11: terminar enquanto
12: Retorna π_p

Por um lado, este custo parcial permite determinar a melhor localização ao longo π_p para uma nova instalação ao usar g_2 ou g_3, ou seja, aquele que minimiza $C(\pi_p)$. No outro Por outro lado, classificando as instalações ainda não incluídas em π_p de acordo com esses custos ótimos, o que permite aplicar estratégias gananciosas e aleatórias para selecionar o instalação que será incorporada a π_p em cada iteração. Por fim, observe que, no contexto das metaheurísticas, g_1 favorece a diversificação (ou seja, aleatoriedade e amplas explorações do espaço de busca), enquanto g_3 favorece a intensificação (ou seja, ganância e exploração de regiões no espaço de busca). Desse modo, g_2 pode ser considerado um compromisso entre as duas abordagens.

Neste artigo, usamos duas famílias de algoritmos construtivos. O primeiro segue o modelo GRASP padrão. O algoritmo 1, denotado como Greedy-Random (GR), mostra o pseudocódigo correspondente. Ele começa criando uma lista de candidatos (CL), que contém as instalações que ainda não foram incluídas na solução parcial em construção. Inicialmente, CL contém todas as instalações (etapa 1). Então, o método seleciona aleatoriamente uma primeira instalação eu a partir de CL (etapa 2), inclui-o na parcial solução π_p (etapa 3) e atualiza a lista de candidatos removendo a instalação escolhida (etapa 4). O método então itera até obter uma solução com $n = | F |$ instalações (etapas 5 a 11). A cada iteração, esta família de algoritmos calcula uma lista de candidatos restrita (RCL) selecionando o melhor $Tamanho$ instalações de CL (etapa 7). Isso é feito classificando previamente as instalações de acordo com as classificações associadas com uma função gananciosa particular. O valor de *Tamanho* depende de um parâmetro $\alpha \in$ [0, 1] (etapa 6). Posteriormente, o algoritmo seleciona uma instalação de *RCL* aleatoriamente (etapa 8), inclui-o na solução parcial (etapa 9) e atualiza a lista de candidatos (etapa 10). O método finalmente retorna a solução completa (observe que π_p conterá exatamente n instalações).

A segunda família de procedimentos construtivos, denotada como Random-Greedy (RG), é baseada em uma estratégia diferente introduzida em Resende e Werneck (2004), que foi aplicada com sucesso recentemente em Campos et al. (2013); Resende et al. (2010); Pantrigo et al. (2012); Duarte et al. (2011). Especificamente, esta construção alternativa troca os estágios gananciosos e aleatórios de uma construção GRASP padrão, conforme ilustrado no Algoritmo 2. Neste caso, *RCL* contém *Tamanho* elementos que são selecionados aleatoriamente de *CL* (etapa 7), enquanto na etapa 8 a melhor instalação é avidamente escolhida a partir de *RCL.*

Por último, observe que α controla a ganância / aleatoriedade dos procedimentos construtivos GRASP. Especificamente, se $\alpha = 0$ os métodos correspondentes são algoritmos puramente gananciosos, enquanto se $\alpha = 1$ eles são procedimentos totalmente aleatórios.

3.2. Pesquisa local. A segunda etapa de um algoritmo GRASP consiste em aprimorar as soluções construídas por meio de um método de busca local. A ideia consiste em aplicar progressivamente alguma estratégia que modifique um ordenamento de instalações de forma a diminuir seu custo de acordo com (1). Este processo de busca, denotado como operador de movimento, é repetido até que nenhuma outra melhoria seja possível (ou seja, ele orienta a busca em direção a um ótimo local).

Em particular, para o SRFLP, definimos dois operadores de movimento diferentes. O primeiro é referido como *trocar.* Dada uma solução $\pi = \langle \pi(1), \ldots, \pi(q), \ldots, \pi(r), \ldots, \pi(n) \rangle$, nós definimos *trocar(* π, q, r*)* como o movimento que troca em π a facilidade $\pi(q)$ em posição n q com a facilidade $\pi(r)$ em posição r, produzindo uma nova solução $\pi' = \langle \pi(1), \ldots, \pi(q-1), \pi(r), \pi(q+1), \ldots, \pi(r-1), \pi(q), \pi(r+1), \ldots, \pi(n) \rangle$. Neste cenário, o conjunto de ordens que podem ser obtidas ao realizar um movimento de troca, normalmente denotado como vizinhança, tem tamanho $n(n-1)/2$.

O segundo operador de movimento é conhecido como *inserir.* Especificamente, dada uma solução π, o movimento *inserir(* π, q, r*)* consiste em remover instalações $\pi(q)$ de sua posição atual q e inserindo-o na posição r. A nova solução π' depende dos valores relativos de q e r. Se $q > r$, então $\pi' = \langle \ldots, \pi(r-1), \pi(q), \pi(r), \pi(r+1), \ldots, \pi(q-1), \pi(q+1), \ldots \rangle$. Em contraste, se $q < r$ então $\pi' = \langle \ldots, \pi(q-1), \pi(q+1), \ldots, \pi(r-1), \pi(r), \pi(q), \pi(r+1), \ldots \rangle$. Neste caso, a vizinhança associada tem tamanho $n(n-1)$.

Existem duas estratégias típicas para explorar uma vizinhança: melhor melhoria e primeira melhoria. O primeiro explora todas as soluções na vizinhança por um procedimento totalmente determinístico, e o melhor movimento (ou seja, aquele que leva a uma solução π' com custo mínimo associado) é aplicado a cada iteração. Abordagens anteriores para lidar com o SRFLP usam a melhor estratégia de melhoria para identificar a melhor troca ou movimento de inserção. Uma implementação ingênua dessas abordagens resultaria em um algoritmo de complexidade computacional $\Theta(n^4)$, uma vez que há um número quadrático de vizinhos, e o custo de avaliar cada um é calculado em $\Theta(n^2)$. No entanto, por meio de escrituração adequada, é possível obter os vizinhos e seus custos em $O(n^3)$ tempo, conforme descrito em Kothari e Ghosh (2012c). Ao usar movimentos de inserção, denotamos este tipo de pesquisa local como LS-BEST.

A primeira estratégia de melhoria explora a vizinhança de uma solução inicial e executa o primeiro movimento que aumenta o custo resultante. O procedimento geralmente escolhe diferentes movimentos aleatoriamente para obter soluções diversas e para quando toda a vizinhança foi explorada e nenhum movimento é capaz de melhorar o custo da solução inicial. No contexto do SRFLP, essa abordagem é ineficiente, pois uma pesquisa puramente aleatória não pode usar a contabilidade. Em particular, observe que o algoritmo é executado em $O(n^4)$ tempo no pior caso, que ocorre, por exemplo, quando para de chegar a um mínimo local.

Apresentamos agora um novo método de pesquisa local híbrido, denominado LS-HYBRID, que usa movimentos de inserção e combina as primeiras e melhores estratégias. Dada uma solução particular, a ideia consiste em selecionar uma instalação aleatoriamente em cada etapa (de forma semelhante à primeira estratégia de melhoria) e, em seguida, encontrar o melhor movimento de inserção apenas para essa instalação (conforme realizado pela melhor estratégia de melhoria). Se o valor da função objetivo não diminuir, o algoritmo escolhe aleatoriamente uma facilidade diferente e procura o novo melhor movimento de inserção para essa facilidade. Este processo é inicializado e repetido conforme ele encontra as melhores soluções. Finalmente, o algoritmo atinge um mínimo local e para quando nenhuma operação de inserção pode melhorar o custo de uma solução anterior.

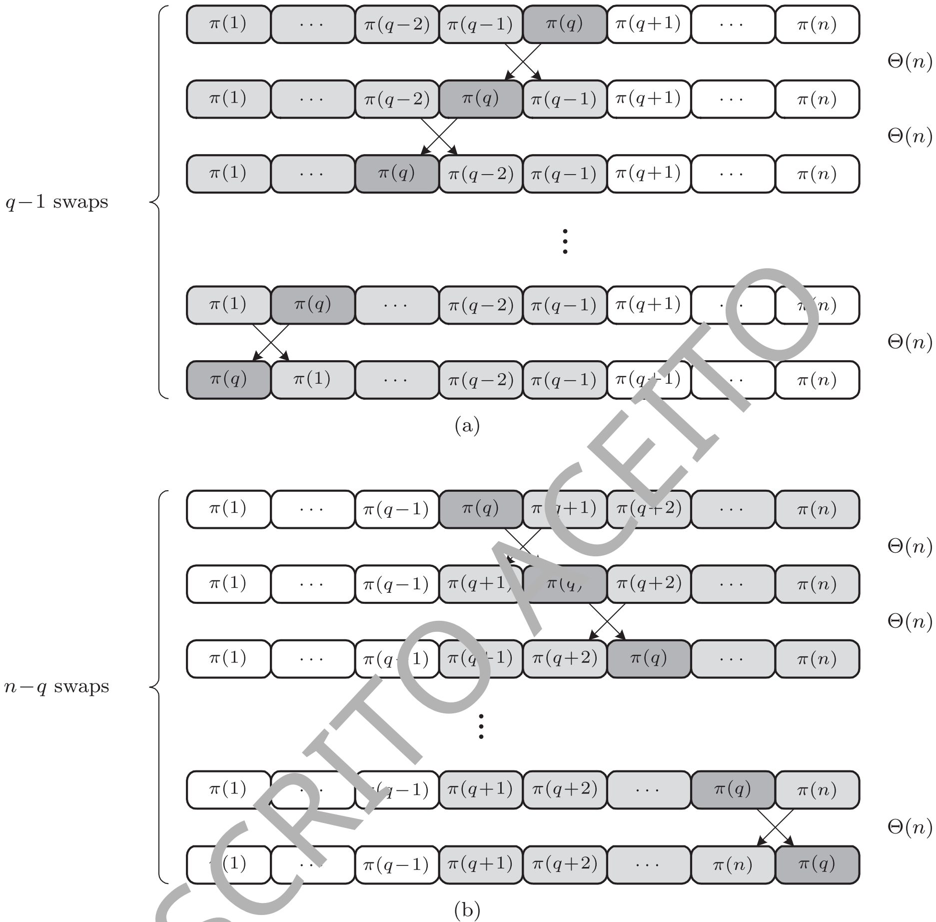
Nosso método de busca local é baseado em movimentos de inserção, uma vez que geralmente apresentam um melhor desempenho na prática do que movimentos de troca (Kothari e Ghosh, 2014a). Para agilizar a busca por meio da escrituração contábil, o método analisa movimentos intermediários de troca de posições consecutivas na busca pelo melhor movimento de melhoria, ao invés de inserir uma facilidade diretamente na posição alvo. Especificamente, dada uma facilidade $\pi(q)$, na posição q, nós trocamos com a facilidade na posição $q - 1$, registrando a mudança associada na função objetivo (denotado como valor de movimento). Então, trocamos as instalações nas posições $q - 1$ e $q - 2$, armazenando novamente o valor de movimento associado. O método procede de forma semelhante, até atingir a posição 1. Simetricamente, a busca local realiza trocas entre posições consecutivas a partir de $q + 1$ para
n A Figura 4 ilustra como a busca local híbrida procede ao realizar um movimento de inserção de instalação $\pi(q)$.

Esta busca local híbrida proposta calcula os valores de movimento de forma incremental, uma vez que a avaliação de um movimento de troca entre posições consecutivas pode ser calculada em tempo linear. Em particular, dada uma solução π e uma instalação $\pi(q)$, o custo de uma solução π' depois de realizar o movimento $trocar(\pi, q, q + 1)$ pode ser calculado como:

$$(3)\quad C(\pi') = C(\pi) + l_{\pi(q+1)} \left(\sum_{s=1}^{q-1} C_{\pi(s)\pi(q)} - \sum_{s=q+2}^{n} C_{\pi(q)\pi(s)} \right) + eu_{\pi(q)} \left(\sum_{s=q+2}^{n} C_{\pi(q+1)\pi(s)} - \sum_{s=1}^{q-1} C_{\pi(s)\pi(q+1)} \right).$$

que roda em $\Theta(n)$ Tempo. O primeiro termo do lado direito da equação identifica o custo da solução π. O segundo ajusta o custo associado à instalação $\pi(q + 1)$, enquanto o último termo corrige o custo para $\pi(q)$. A combinação da composição dos movimentos de troca para obter um movimento de inserção geral, juntamente com o cálculo incremental do valor de custo definido acima, torna a pesquisa local muito eficiente, como mostraremos em nossos resultados computacionais (ver Seção 6). Observe que um movimento de inserção requer $\Theta(n^2)$ tempo, uma vez que cada um dos $n - 1$ operação de swap a ser realizada é executada em tempo linear. Por último, o algoritmo requer $O(n^3)$ tempo ao atingir um mínimo local (ou seja, no pior caso).

(uma)



(b)

Figura 4. Inserir movimentos de uma instalação (na posição *q)* por meio de operações de swap contíguas. As trocas são realizadas para a esquerda (a) e direita (b). Todo o processo requer Θ (n^2) Tempo.

4 Revinculação de caminho

Path relinking (PR) é uma metaheurística originalmente proposta como uma metodologia para integrar estratégias de intensificação e diversificação no contexto da busca tabu (Glover, 1989; 1990). Seu modelo de algoritmo atual é definido em Glover et al. (1998) como estratégia geral de combinação de soluções. O método gera novas soluções modificando iterativamente uma inicial, transformando-a em outra em sua vizinhança, até chegar a uma solução norteadora final. Portanto, o processo constrói um *trajetória* de novas soluções intermediárias com a esperança de que possam eventualmente ser melhores do que as soluções de alta qualidade que estão sendo conectadas.

A escolha do operador de movimentação determina como o caminho é construído. Neste artigo, primeiro descrevemos a abordagem de Kothari e Ghosh (2012b), que denotamos como PR1. Além disso, apresentamos duas estratégias alternativas para construir caminhos entre soluções de alta qualidade, denotadas como PR2 e PR3. Para implementar essas estratégias de forma eficiente, usamos uma representação alternativa de uma solução. Especificamente, se π é uma solução para o SRFLP (onde facilidade $\pi(q)$ está localizado na posição q), então ele também pode ser especificado por meio de sua representação inversa π^{-1}, que indica as posições onde as instalações estão localizadas em π (ou seja, facilidade *eu* é localizado na posição $\pi^{-1}(eu)$ dentro π). Por exemplo, deixe $\pi_x = \langle 2, 3, 1, 4, 5 \rangle$ seja algum inicial ordenando e $\pi_y = \langle 5, 2, 1, 4, 3 \rangle$ uma solução orientadora. A representação inversa de π_x é $\pi^{-1}_x = \langle 3, 1, 2, 4, 5 \rangle$, uma vez que a instalação 1 está localizada na terceira posição em π_x instalação 2 está localizado na primeira posição e assim por diante. Da mesma forma, a representação inversa de π_y é $\pi^{-1}_y = \langle 3, 2, 5, 4, 1 \rangle$. Esta representação alternativa permite que os algoritmos PR para localizar com eficiência uma instalação e sua posição correspondente.

O primeiro algoritmo baseado na metodologia de reconexão de caminho (PR1) é baseado em movimentos de troca, que são realizados em uma ordem (determinística). PR1 analisa cada elemento em π_x e π_y em ordem, digamos, da esquerda para a direita. Se em alguma iteração q (com $1 \leq q \leq n$), a facilidade $\pi_x(q)$ é diferente de $\pi_y(q)$, então um movimento de troca é executado em π_x para que o q-ª instalação em π_x será o mesmo que em π_y. Formalmente, o movimento é definido por $trocar(\pi_x, q, \pi^{-1}_x(\pi_y(q)))$. Considerando as soluções anteriores π_x e π_y, o primeiro movimento trocaria as instalações 2 e 5 em π_x, gerando a solução intermediária $\pi_1 = \langle 5, 3, 1, 4, 2 \rangle$, Onde $\pi_1(1) = \pi_y(1) = 5$. O segundo troca as instalações 2 e 3, o que produz uma nova solução $\pi_2 = \langle 5, 2, 1, 4, 3 \rangle$, cujas duas primeiras instalações coincidem com as da solução orientadora π_y. O subsequente etapas não geram movimentos desde $\pi_2 = \pi_y$.

A abordagem de religação do segundo caminho, denotada como PR2, favorece a diversificação dos caminhos construídos e pode ser considerada uma variante aleatória de PR1. Esta nova abordagem começa identificando o conjunto D de facilita em ambas as soluções que estão localizadas em posições diferentes. Por exemplo, considerando o anterior lutions π_x e π_y, este conjunto é $D = \{5, 2, 3\}$, uma vez que as instalações 1 e 4 estão localizadas em ambas as permutações nas posições 3 e 4, respectivamente. PR2 seleciona uma instalação $eu \in D$ aleatoriamente, cujas posições em π_x e π_y estão $\pi^{-1}_x(eu)$ e $\pi^{-1}_y(eu)$, respectivamente. Subseqüentemente, ele executa o movimento $swap(\pi_x, \pi^{-1}_x(i), \pi^{-1}_y(eu))$, que garante que instalação *eu* estará localizado na mesma posição nas soluções intermediárias e guias. Por exemplo, suponha que PR2 seleciona aleatoriamente a instalação $i = 3$, cuja posição ções em π_x e π_y pode ser determinado com eficiência usando a representação inversa ($\pi^{-1}_x(3) = 2$ e $\pi^{-1}_y(3) = 5$). A fim de situar a instalação 3 na mesma posição em ambas as soluções, PR2 realiza a mudança $swap(\pi_x, \pi^{-1}_x(3), \pi^{-1}_y(3)) = swap(\pi_x, 2, 5)$, produzindo a solução intermediária $\pi_1 = \langle 2, 5, 1, 4, 3 \rangle$. A próxima etapa seria selecionar ou instalação 2 ou 5, gerando um movimento de troca entre as duas primeiras instalações em π_1 isso produziria a solução orientadora.

Propomos uma terceira estratégia de reconexão de caminhos, mais sofisticada, denominada PR3. Está relacionado ao cálculo da distância de Ulam (Ulam, 1972), que mede o número mínimo de movimentos da pastilha necessários para transformar um pedido em outro. Também pode ser entendido como n menos o comprimento da subsequência comum mais longa entre duas ordenações, que pode ser calculada em $O(n^2)$ tempo por um algoritmo de programação dinâmica simples (Sevaux e S¨örensen, 2005). No entanto, uma vez que as soluções para o SRFLP são pedidos do primeiro n inteiros,

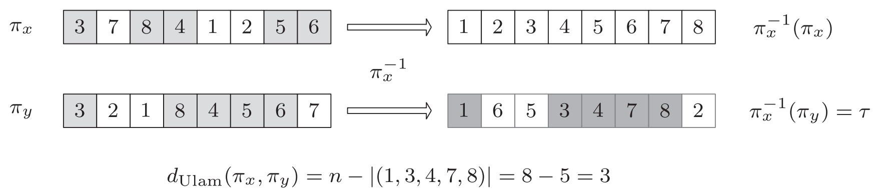


Figura 5. A distância Ulam indica o número mínimo de inserções necessárias para transformar alguns pedidos (π_x) em outro (π_y). Pode ser calculado aplicando primeiro π^{-1}_x para ambos permutações, o que simplesmente altera a rotulagem das instalações. Posteriormente, a maior subsequência crescente $\sigma = (1,3,4,7,8)$ de $\pi^{-1}_x(\pi_y) = \tau$, cujos elementos aparecem sombreados (em cinza escuro), indicates os elementos (3, 8, 4, 5, 6) que compartilham a mesma ordem relativa nas permutações iniciais (mostrado sombreado em cinza claro). A distância de Ulam é $n - |\sigma| = 3$, onde $|\cdot|$ denota comprimento, e a trajetória de religação do caminho pode ser formada inserindo as instalações 1, 2 e 7.

é possível calcular a distância Ulam resolvendo o problema de subsequência crescente mais longo, que pode ser calculado em $O(n$ registro $n)$, como descrevemos abaixo (Hunt e Szymanski, 1977).

A primeira etapa do PR3 consiste em identificar o maior conjunto de elementos em ambas as permutações que preservam a mesma ordem relativa entre eles. Por exemplo, deixar $\pi_x = \langle 3, 7, 8, 4, 1, 2, 5, 6 \rangle$ e $\pi_y = \langle 3, 2, 1, 8, 4, 5, 6, 7 \rangle$ ser duas soluções para alguma instância do SRFLP. Apenas a instalação 3 está localizada na mesma (primeira) posição na ambos os pedidos. No entanto, podemos observar que as instalações 3, 8, 4, 5 e 6 compartilham a mesma ordem relativa em ambas as soluções. Em outras palavras, $\pi^{-1}_x(3) < \pi^{-1}_x(8) < \pi^{-1}_x(4) < \pi^{-1}_x(5) < \pi^{-1}_x(6)$ e $\pi^{-1}_y(3) < \pi^{-1}_y(8) < \pi^{-1}_y(4) < \pi^{-1}_y(5) < \pi^{-1}_y(6)$ (veja a Figura 5). Deixar λ representam este maior conjunto de instalações, que não é o único em geral. Por conveniência de notação, iremos representá-lo como uma lista ($\lambda = (3, 8, 4, 5, 6)$ no exemplo).

Observe que se π_x foram os pedidos $(1, 2,\ldots, n)$ então λ corresponderia ao maior subsequência crescente em π_y. No entanto, desde π_x pode ser qualquer encomenda, é necessário modificar as etiquetas das instalações para obter λ por computação uma subsequência crescente mais longa. Especificamente, facilidade $\pi_x(1)$ teria que ser renomeado como 1, $\pi_x(2)$ como 2 e assim por diante, o que é obtido aplicando π^{-1}_x para a inicial ordenações, conforme mostrado na Figura 5. Portanto, λ pode ser encontrado calculando primeiro a subsequência crescente mais longa na seguinte ordem:

$$\tau = \pi^{-1}_x(\pi_y) = \langle \pi^{-1}_x(\pi_y(1)), \pi^{-1}_x(\pi_y(2)), \ldots, \pi^{-1}_x(\pi_y(n)) \rangle .$$

Observe que τ codifica a localização em π_x das instalações em π_y (ou seja, $\tau(q) = \pi^{-1}_x(\pi_y(q))$ é a posição em π_x do q-ª instalação em π_y). Em outras palavras, indica a ordem relativa em π_x das instalações em π_y. No exemplo $\tau = \langle 1, 6, 5, 3, 4, 7, 8, 2 \rangle$, desde a primeira instalação em π_y está localizada na primeira posição em π_x, o segundo em π_y é localizado na sexta posição em π_x, e assim por diante.

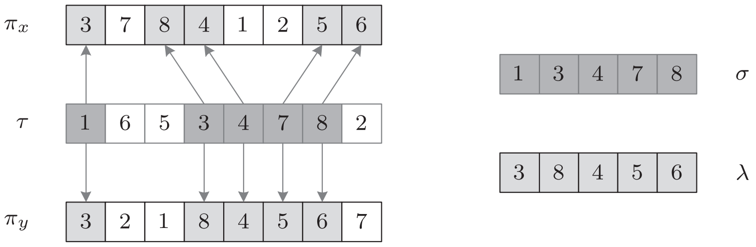


Figura 6. Recuperando λ de duas maneiras diferentes. Por um lado, σ fornece as localizações dos elementos de λ dentro π_x. No outro lado, os elementos de λ estão localizados em π_y nas mesmas posições que os elementos de σ estão localizados em τ.

Posteriormente, vamos σ representam a maior subsequência crescente em τ, que é (1, 3, 4, 7, 8) no exemplo (observe que os elementos em σ não são necessariamente contíguas). Finalmente, λ pode ser recuperado através de σ de duas maneiras diferentes, conforme ilustrado na Figura 6. Por um lado, σ fornece as posições onde os elementos de λ estão localizado em π_x (setas oblíquas para cima), ou seja, $\lambda = \pi_x(\sigma) = (\pi_x(\sigma(1), \pi_x(\sigma(2)), \ldots)$. Observe no exemplo que $\lambda = (3, 8, 4, 5, 6) = \pi_x((1, 3, 4, 7, 8)$. Por outro lado, as posições em τ dos elementos em σ são precisamente as posições onde os elementos de λ estão localizados em π_y (setas para baixo). No exemplo $\lambda = \pi_y(\tau^{-1}(\sigma))$, Onde $\lambda = (3, 8, 4, 5, 6) = \pi_y(\tau^{-1}((1, 3, 4, 7, 8))) = \pi_y((1, 4, 5, 6, 7))$. A distância Ulam é a diferença entre n e o tamanho de σ (qual é o mesmo que o tamanho de λ). No exemplo acima, é: 8 - 5 = 3, o que significa que nós pode transformar π_x para dentro π_y executando apenas três movimentos de inserção. Portanto, PR3 geraria uma trajetória com apenas 2 soluções intermediárias.

Finalmente, o algoritmo seleciona os movimentos de inserção a serem executados aleatoriamente. Em particular, deixe D ser o conjunto de instalações que não estão em λ (no exemplo, $D = \{1, 2, 7\}$). Em primeiro lugar, PR3 seleciona uma instalação de D aleatoriamente e o insere em uma posição adequada em π_x para aumentar o número de instalações que compartilham a mesma ordem relativa em ambas as permutações. Por exemplo, suponha que PR3 escolhe a facilidade 1. Como deve ser inserido entre as instalações 3 e 8, pode ser inserido na posição 2 ou 3. Nosso algoritmo proposto seleciona um desses movimentos possíveis aleatoriamente. Por último, este processo é repetido até que todos os elementos em D foram inseridos corretamente.

5 GRASP com revinculação de caminho

A reconexão de caminhos pode ser adaptada no contexto do GRASP como uma forma de estratégia de pós-otimização (Laguna e Mart´ı, 1999). A combinação de ambas as abordagens opera em um conjunto de soluções de tamanho b que geralmente é chamado de conjunto de elite ($ES = \{ES_1, ES_2, \ldots, ES_b\}$). Normalmente é classificado de acordo com a qualidade do soluções (da melhor, ES_1, para pior, ES_b). A abordagem inicializa primeiro ES com soluções construídas por um procedimento GRASP. Depois, o algoritmo continua gerando mais soluções GRASP que são combinadas com aquelas do conjunto de elite por meio de uma abordagem de religação de caminho. As soluções resultantes substituem as existentes no conjunto de elite, melhorando sua qualidade geral.

Algoritmo 3: GRASP com algoritmo de reconexão de caminho

```
1: eu ← ∅
2: para i = 1 para m Faz
3:     π_eu ← APERTO()
4:     eu ← eu ∪ { π_eu}
5: fim para
6: ES ← SelectBest ( I, b)
7: enquanto tempo de execução < T_max Faz
8:     π_APERTO ← APERTO()
9:     para i = 1 para b Faz
10:        π_PR ← PathRelinking ( π_APERTO, ES_eu)
11:        E se ( π_PR 6 ∈ ES) e ( C (π_PR) ≤ C ( ES_b)) então
12:            j ← argmin_{1 ≤ eu ≤ n} { d (π_PR, ES_eu)}
13:            E se ( j 6 = 1) ou ( C (π_PR) < C ( ES_1) então
14:                ES_j ← π_PR
15:                ES ← ordenar( ES)
16:            fim se
17:        fim se
18:    fim para
19: terminar enquanto
20: Retorna ES_1
```

O algoritmo 3 mostra o pseudocódigo de nosso GRASP proposto com procedimento de religação de caminho. O conjunto de elite é inicialmente preenchido por reter os melhores b soluções geradas por um procedimento GRASP (ver Seção 3) que é executado $m \geq b$ vezes (etapas 2 a 6). Uma vez que nos concentramos principalmente na qualidade das soluções armazenadas em ES nós escolhemos um grande valor para m isso depende do tamanho do problema.

Posteriormente, o algoritmo itera gerando uma nova solução GRASP π_{APERTO} (etapa 8), que será combinada com soluções de elite estabelecidas por meio de um caminho procedimento de religação, até atingir uma quantidade fixa de tempo de execução T_{max} (passos 7 a 19). Em particular, em vez de gerar um caminho entre π_{APERTO} e uma solução selecionada probabilisticamente de ES (como de costume), calculamos todos os caminhos trajetórias entre π_{APERTO} e cada uma das soluções em ES (etapas 9 a 18). Deixar π_{PR} representam a melhor solução encontrada em cada trajetória (etapa 10), que foi adicionalmente otimizada por meio de um dos procedimentos de busca local descritos na Seção ção 3.2. O algoritmo prossegue avaliando π_{PR} a fim de determinar se deve ser incluído no ES. Em primeiro lugar, não pode já estar presente em ES, e os seus custo associado deve ser pelo menos tão grande quanto o custo da pior solução em ES (etapa 11). Se essas condições forem atendidas, o algoritmo prossegue calculando o solução mais próxima em ES (denotado como ES_j) para π_{PR} (etapa 12). Em particular, calculamos a dissimilaridade entre soluções através da distância entre permutações usado anteriormente em Kothari e Ghosh (2014b). Formalmente, essa distância é definida como:

$$d(\pi_x, \pi_y) = \min\{\delta(\pi_x, \pi_y), \delta(\pi_x, \bar{\pi}_y)\},$$

Onde $\bar{\pi}_y$ é a ordem inversa de π_y (ou seja, $\bar{\pi}_y = \langle \pi_y(n), \pi_y(n-1), \ldots, \pi_y(1) \rangle$), e

$$\delta(\pi_x, \pi_y) = \sum_{i=1}^{n} |\pi_x^{-1}(eu) - \pi_y^{-1}(i)|$$

é a “distância de desvio” (Sevaux e S¨örensen, 2005). Finalmente, π_{PR} é introduzido no conjunto de elite, enquanto ES_j é removido (etapa 14), e ES é recolocado (observe que esta operação pode ser realizada em tempo linear (etapa 15)). No entanto, isso ocorre apenas se π_{PR} é a melhor solução encontrada até agora, ou contanto que a solução a ser excluída não seja a melhor no conjunto elite (etapa 13). Portanto, se π_{PR} é melhor que ES_1 isso é sempre admitido em ES. Além disso, se for pior do que ES_1, então ES_1 não será removido do conjunto elite.

Finalmente, a Figura 7 contém um diagrama de fluxo resumido da abordagem. Os círculos sombreados representam soluções para o SRFLP, onde seus custos associados são representados pelo tamanho dos círculos (círculos menores representam melhores soluções). Ob-servir aquele caminho de revinculação é aplicado entre π_{APERTO} e soluções em ES. Ao melhor solução ao longo da trajetória (π_{PR}) é escolhido, melhorado e substitui uma solução em ES dependendo das condições nas etapas 11 e 13 do Algoritmo 3.

6 Resultados computacionais

Nesta seção relatamos os experimentos computacionais realizados para testar a eficiência e eficácia das estratégias propostas. Realizamos todos os experimentos em um computador pessoal com um Intel de quarta geração r Testemunho TM Processador i7-4712HQ de 3,3 GHz e 16 GB de RAM. Todo o código foi escrito em C e compilado com o compilador gcc. Por último, construímos nossas próprias implementações dos métodos em Kothari e Ghosh (2014a) e Kothari e Ghosh (2014b), uma vez que o código / executável não estava disponível.

Executamos experimentos nos três conjuntos de instâncias de benchmark usados na literatura recente sobre o SRFLP, que estão disponíveis publicamente em Anjos (2016). O primeiro foi apresentado em Anjos et al. (2005) e consiste em quatro grupos (para n = 60, 70, 75 e 80) de cinco instâncias. Esses grupos são geralmente conhecidos como Anjos instâncias. Outro conjunto popular, originalmente introduzido em Anjos e Yen (2009), é baseado no problema de Atribuição Quadrática e denominado sko. Neste artigo, usamos os quatro grupos (para n = 64, 72, 81 e 100) de cinco instâncias. Finalmente, um terceiro conjunto, referido como Amaral, é devido a Amaral e Letchford (2013) (ver o documento de trabalho associado) e consiste em três instâncias, cada uma de tamanho n = 110.

Como os estudos mais recentes relatam essencialmente os mesmos resultados nessas instâncias, geramos novas instâncias maiores e mais desafiadoras para comparar algoritmos. Em particular, criamos dois novos conjuntos, Anjos-grande (40 instâncias) e sko-large (40 instâncias), onde comprimentos e custos foram retirados de distribuições que se assemelhavam tanto quanto possível àquelas usadas no Anjos e sko conjuntos de instâncias. Cada conjunto contém 40 instâncias com tamanho consideravelmente maior. Especificamente, consideramos valores cinco instâncias para valores de n em {150, 200, 250, 300, 350, 400, 450, 500}. Esses conjuntos estão disponíveis em Rubio-S´ánchez et al. (2016).

Nesta seção, primeiro descrevemos uma experimentação preliminar com um subconjunto de *Treinamento* instâncias que permitem comparar as diferentes alternativas introduzidas nas seções anteriores e identificar uma em particular como nosso algoritmo final. Este conjunto de treinamento consiste em 40 instâncias de tamanhos 150, 250, 350 e 450, incluídas no Anjos-grande e sko-grande conjuntos. Por último, relatamos outros experimentos computacionais realizados com um diferente *teste* conjunto de instâncias para comparar nosso algoritmo final com métodos de última geração. Em particular, comparamos os resultados com relação a: (i) o algoritmo genético GENALGO descrito em Kothari e Ghosh (2014a), e (ii) a variante de pesquisa dispersa SS-2P relatada em Kothari

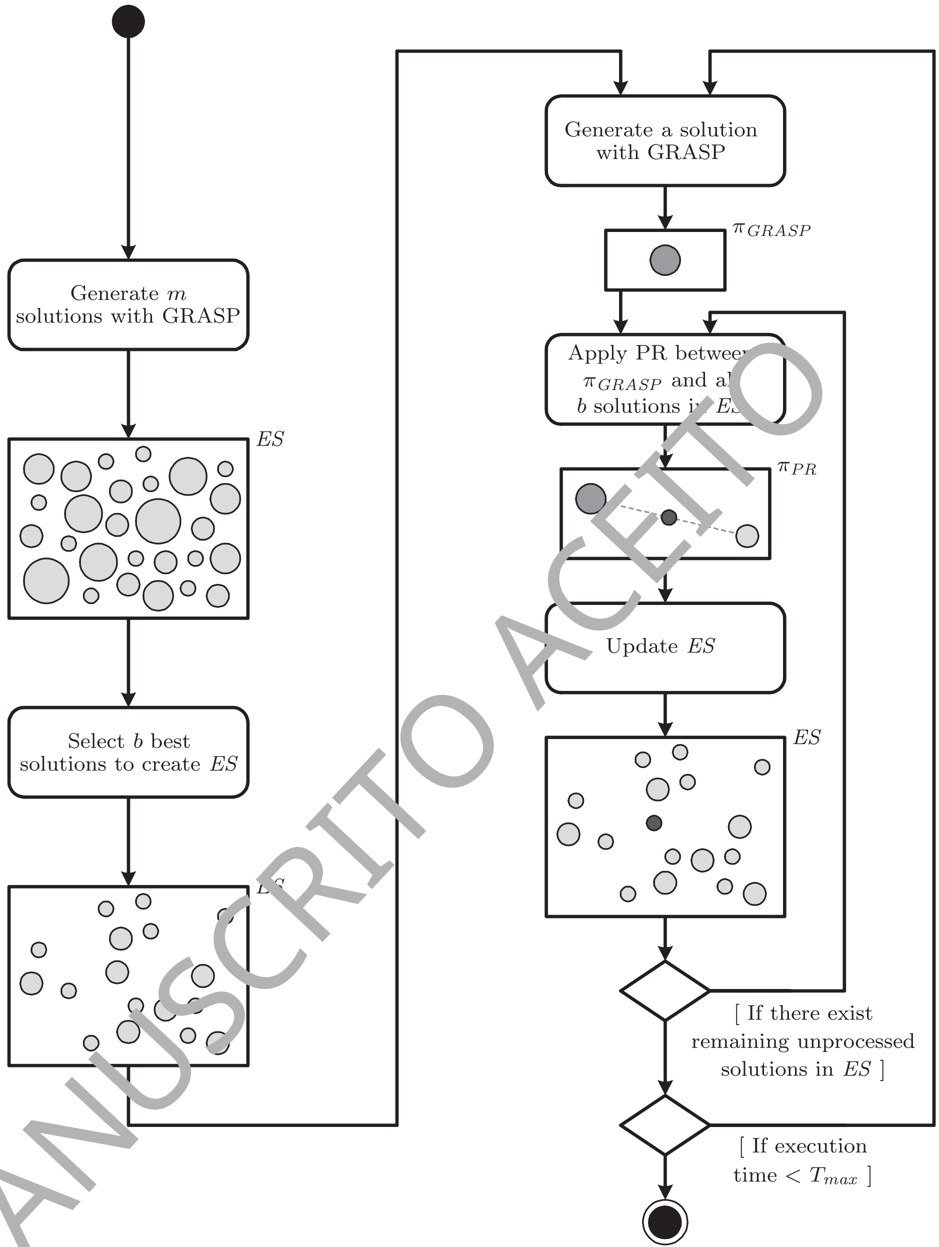


Figura 7. Diagrama de fluxo de nosso GRASP com abordagem de reconexão de caminho.

e Ghosh (2014b), que descobrimos ter um desempenho ligeiramente melhor do que as outras três variantes descritas nesse artigo. Finalmente, o conjunto de teste contém as 40 instâncias restantes, de tamanhos 200, 300, 400 e 500, incluídas no Anjos-grande e sko-grande conjuntos.

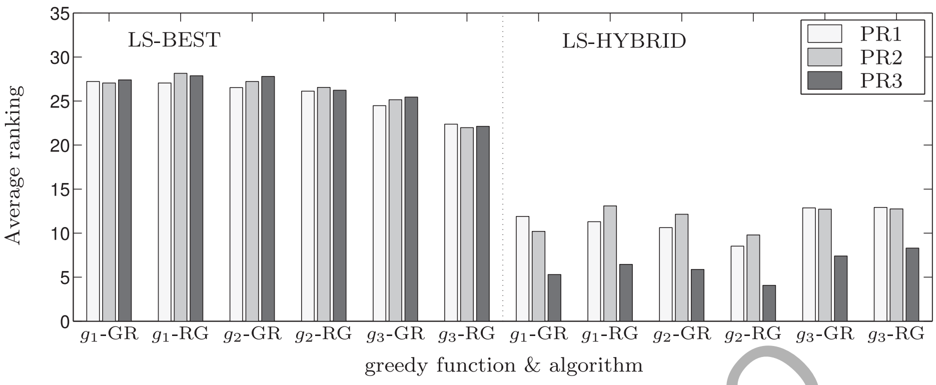


Figura 8. Classificação média de 36 GRASP e algoritmos de reconexão de caminho entre as 40 instâncias de treinamento. As variantes comparadas use as melhores pesquisas locais ou híbridas, as funções gananciosas g_1, g_2 ou g_3, os algoritmos GR ou RG e as estratégias de religação de caminho PR1, PR2 ou PR3.

6.1. Experimentos preliminares. Realizamos uma combinação exaustiva dos elementos algorítmicos descritos nas seções anteriores para examinar seu desempenho nas instâncias de treinamento. Particularmente, executamos um único full-experimento fatorial onde consideramos as funções gulosas g_1, g_2 e g_3, os algoritmos GR e RG, os dois métodos de busca local LS-BEST e LS-HYBRID, e as estratégias de religação de caminho PR1, PR2 e PR3. Não incluímos variações do α parâmetro uma vez que não observamos diferenças relevantes. Em particular, testamos $\alpha = \{0{,}25, 0{,}5, 0{,}75\}$, mas nenhum deles mostrou uma superioridade significativa sobre o resto. Portanto, os resultados relatados consideram apenas um valor padrão de $\alpha = 0{,}5$ como uma troca entre intensificação e diversificação. Além disso, os principais parâmetros do método GRASP com PR dependem do tamanho do problema (n). Especificamente, o número de iterações GRASP iniciais, m, está configurado para $m = d\, n / 2$ e, enquanto o tamanho do conjunto de elite, b, é fixo a $b = d\, n / 20$ e. Além disso, uma vez que nossos experimentos finais foram planejados considerando os limites de tempo de n e 3600 segundos, definimos o tempo máximo de execução para 2 n segundos como um compromisso e para evitar qualquer possível sobreajuste dos parâmetros. Por fim, testamos experimentalmente que pequenas variações desses parâmetros quase não afetam o desempenho dos algoritmos comparados.

A Figura 8 mostra uma comparação de 36 variantes algorítmicas. O X- eixo indica uma função gananciosa e algoritmo de construção particular, enquanto o Y - eixo mede a classificação média de cada variante. A linha pontilhada (no meio do gráfico) separa os algoritmos que usam LS-BEST (lado esquerdo) daqueles que usam LS-HYBRID (lado direito). Finalmente, para cada algoritmo indicado no X- eixo, consideramos três colunas diferentes que correspondem a uma das três estratégias de religação de caminho.

A classificação média é calculada considerando os resultados nas 40 instâncias do conjunto de treinamento. Especificamente, se um algoritmo é o melhor (ou seja, fornece o custo mais baixo) em alguma instância, sua classificação é 1 para essa instância específica. Em vez disso, se fosse o pior, sua classificação seria 36 (uma vez que esse é o número de algoritmos sendo comparados). Além disso, se dois algoritmos fornecem o mesmo custo em uma instância, eles recebem a mesma classificação. Finalmente, executamos cada variante uma vez (por 2 n segundos) em cada instância.

Os resultados mostram claramente que a estratégia de busca local híbrida proposta supera a melhor abordagem de inserção usada nos algoritmos de última geração. Em particular, todas as variantes baseadas em LS-HYBRID superam claramente aquelas baseadas em LS-BEST. Focando apenas em variantes que usam LS-HYBRID, podemos concluir que a estratégia de religação de caminho com base na distância Ulam (PR3) é superior a PR1 (que é descrito no estado da arte) e PR2 (normalmente usado em religação de caminho) . Observe que isso não ocorre para as variantes baseadas no LS-BEST. Isso pode ser parcialmente explicado pelos tempos de computação mais longos exigidos pelo LS-BEST, que pode causar menos iterações de religação de caminho. Finalmente, entre as funções gananciosas

e algoritmos de construção, a combinação de g_2 e RG fornece os melhores resultados. Portanto, definimos nosso algoritmo final para resolver o SRFLP, denotado como GRASP-PR, como a combinação de LS-HYBRID, PR3, g_2, e RG (com $\alpha = 0,5$).

6,2 Comparação com algoritmos de última geração. A comparação final com os métodos atuais de última geração é dividida em três experimentos diferentes. No primeiro, comparamos nossa melhor variante, GRASP-PR, com os algoritmos GENALGO e SS-2P sobre os conjuntos Anjos, sko, e Amaral. Mostramos na Tabela 1 os resultados associados, onde a primeira coluna indica o nome de cada instância (o primeiro número indica seu tamanho) e as demais colunas relatam, para cada algoritmo, o custo da melhor solução encontrada e o tempo de computação associado em segundos. Vale a pena mencionar nosso algoritmo GRASP-PR, ao executar para $n/2$ segundos, encontrou uma solução melhor em uma das instâncias (sko 81 1), enquanto foi capaz de corresponder aos melhores resultados publicados na literatura nas restantes 42 instâncias (obtivemos os mesmos resultados após executar GRASP-PR por uma hora). Em contraste, GENALGO e SS-2P não alcançaram o melhor custo em uma e quatro instâncias (marcadas entre colchetes e com um asterisco), respectivamente. Finalmente, os tempos de execução do GENALGO e SS-2P foram obtidos diretamente dos artigos originais.

É difícil fazer uma comparação justa quando se considera o tempo de CPU, uma vez que GENALGO e SS-2P foram executados em um computador diferente (processador Intel i-5 2500 3,30 GHz com 4 GB de RAM). No entanto, para nossa melhor estimativa, parece que GRASP-PR é mais rápido do que SS-2P em grandes instâncias, e muito mais rápido do que GENALGO quando é executado 200 vezes, como é feito em Kothari e Ghosh (2014a). Para apoiar esta afirmação, implementamos estes dois métodos, designados por GENALGO R e SS-2P R, seguindo as indicações descritas nos artigos originais correspondentes, e na mesma linguagem de programação (C). A Tabela 2 mostra o custo e o tempo de computação para ambos os métodos após executá-los no Anjos, sko e Amaral conjuntos. Nossas implementações alcançam um desempenho semelhante em tempos de computação consideravelmente menores do que aqueles relatados na literatura. Por um lado, GENALGO R encontra os resultados mais conhecidos em todas as instâncias (observe que a versão original falhou em uma delas), enquanto o tempo de computação é reduzido em cerca de 70% em relação aos resultados originais relatados. Por outro lado, o SS2P R encontrou as melhores soluções em todas as instâncias, exceto três (ou seja, uma a mais do que SS-2P), enquanto reduzia o tempo de computação em quase 80%. Embora o aumento na velocidade possa ser devido em parte ao uso de um computador mais rápido, é suficientemente grande para afirmar que nossas implementações são válidas.

Tabela 1. Comparação de desempenho de GENALGO, SS-2P e GRASP-PR em Anjos, sko, e Amaral instâncias.

	GENALGO		SS-2P		GRASP-PR	
Instância	Custo	Tempo	Custo	Tempo	Custo	Tempo
Anjos 60 1	1477834	2110	1477834	41	1477834	30
Anjos 60 2	841776	2126	841776	47	841776	30
Anjos 60 3	648337,5	2200	648337,5	65	648337,5	30
Anjos 60 4	398406	2154	398406	53	398406	30
Anjos 60 5	318805	2102	318805	48	318805	30
Anjos 70 1	1528537	3670	1528537	91	1528537	35
Anjos 70 2	1441028	3556	1441028	97	1441028	35
Anjos 70 3	1518993,5	3658	1518993,5	100	1518993,5	35
Anjos 70 4	968796	3786	968796	95	968796	35
Anjos 70 5	4218002.5	3572	4218002.5	80	4218002.5	35
Anjos 75 1	2393456,5	5008	2393456,5	119	2393456,5	37,5
Anjos 75 2	4321190	4996	4321190	137	4321190	37,5
Anjos 75 3	1248423	4994	1248423	137	1248423	37,5
Anjos 75 4	3941816,5	4974	3941816,5	[illegible]	3941816,5	37,5
Anjos 75 5	1791408	5108	1791408	[illegible]	1791408	37,5
Anjos 80 1	2069097,5	6532	2069097,5	172	2069097,5	40
Anjos 80 2	1921136	6304	1921136	167	1921136	40
Anjos 80 3	3251368	6362	3251368	147	3251368	40
Anjos 80 4	3746515	6336	3746515	182	3746515	40
Anjos 80 5	1588885	[illegible]	1588885	205	1588885	40
sko 64 1	96881	[illegible]	[96890] *	93	96881	32
sko 64 2	634332,5	[illegible]	634332,5	89	634332,5	32
sko 64 3	414323,5	2832	414323,5	68	414323,5	32
sko 64 4	297129	[illegible]	297129	85	297129	32
sko 64 5	501922,5	2800	501922,5	78	501922,5	32
sko 72 1	139150	[illegible]	139150	150	139150	36
sko 72 2	711998	4582	711998	226	711998	36
sko 72 3	1054110,5	4480	1054110,5	112	1054110,5	36
sko 72 4	919586,5	4446	919586,5	132	919586,5	36
sko 72 5	428226,5	4638	[428228,5] *	172	428226,5	36
sko 81 1	[205108,5] *	8154	[205112] *	274	205106	40,5
sko 81 2	521391,5	7326	521391,5	227	521391,5	40,5
sko 81 3	970796	7186	970796	263	970796	40,5
sko 81 4	2031803	7370	2031803	237	2031803	40,5
sko 81 5	1302711	7116	1302711	257	1302711	40,5
sko 100 1	378234	19950	378234	790	378234	50
sko 100 2	2076008,5	17264	2076008,5	468	2076008,5	50
sko 100 3	16145614,5	16588	[16149444] *	530	16145614,5	50
sko 100 4	3232522	16302	3232522	532	3232522	50
sko 100 5	1033080.5	17074	1033080,5	619	1033080,5	50
Amaral 110 1	144296664,5	38398	144296664,5	777	144296664.5	55
Amaral 110 2	86050037	38134	86050037	775	86050037	55
Amaral 110 3	2234743,5	23914	2234743,5	611	2234743,5	55

[·] * Não encontrou o melhor valor.

Mesa 2. Desempenho de nossas implementações de SS-2P e GENALGO em Anjos, sko, e Amaral instâncias.

	GENALGO R		SS-2P R	
Instância	Custo de	Tempo	Custo	Tempo
Anjos 60 1	1477834	585	1477834	10,0
Anjos 60 2	841776	617	841776	11,6
Anjos 60 3	648337,5	678	648337,5	8,2
Anjos 60 4	398406	686	398406	12,3
Anjos 60 5	318805	615	318805	7,3
Anjos 70 1	1528537	1150	1528537	13,9
Anjos 70 2	1441028	1149	1441028	23,0
Anjos 70 3	1518993,5	1159	1518993,5	26,7
Anjos 70 4	968796	1294	968796	22,3
Anjos 70 5	4218002.5	1047	4218002.5	19,2
Anjos 75 1	2393456,5	1518	2393456,5	18,1
Anjos 75 2	4321190	1634	4321190	38,1
Anjos 75 3	1248423	1735	1248423	36,8
Anjos 75 4	3941816,5	1503	3941816,5	25,5
Anjos 75 5	1791408	1657	1791408	35,1
Anjos 80 1	2069097,5	2078	2069097,5	33,5
Anjos 80 2	1921136	1917	1921136	33,5
Anjos 80 3	3251368	1918	3251368	37,0
Anjos 80 4	3746515	1998	3746515	22,8
Anjos 80 5	1588885	2200	1588885	46,2
sko 64 1	96881	1158	96881	14,0
sko 64 2	634332,5	964	634332,5	13,1
sko 64 3	414323,5	995	414323,5	15,1
sko 64 4	297129	1081	297129	22,6
sko 64 5	501922,5	965	501922,5	17,4
sko 72 1	139150	1694	139150	42,1
sko 72 2	711998	1706	711998	45,8
sko 72 3	1054110,5	1716	[1054480.5] *	18,5
sko 72 4	919586,5	1564	919586,5	21,6
sko 72 5	428226,5	1683	428226,5	41,4
sko 81 1	205106	3297	205106	59,0
sko 81 2	521391,5	2620	521391,5	33,6
sko 81 3	970796	2881	970796	47,7
sko 81 4	2031803	2632	[2031826] *	38,9
sko 81 5	1302711	2643	1302711	30,5
sko 100 1	378234	7835	378234	101,1
sko 100 2	2076008,5	6607	2076008,5	143,8
sko 100 3	16145614,5	6324	16145614,5	90,0
sko 100 4	3232522	6262	3232522	177,6
sko 100 5	1033080,5	6861	[1033320,5] *	86,6
Amaral 1.10 1	144296664,5	9899	144296664.5	153,0
Amaral 1.10 2	86050037	9675	86050037	112,5
Amaral 1.10 3	2234743,5	8931	2234743,5	225,3

[·]* Não encontrou o melhor valor.

Tabela 3. Comparação de algoritmos SRFLP em Anjos-grande e sko-grande instâncias, ao executá-los para *n* segundos.

Instância	GENALGO R	SS-2P R	GRASP-PR
Anjos 200 1	305497735	305461895	305461862
Anjos 200 2	178807686,5	178852729,5	178816295,5
Anjos 200 3	61893221	61891688	61891275
Anjos 200 4	127743269	127736617	127736350
Anjos 200 5	89059441,5	89141170,5	89057182.5
Anjos 300 1	1550458661	1552008518	1549663680
Anjos 300 2	956868890,5	957367664,5	955572080.5
Anjos 300 3	308456080.5	308868436,5	308257766.5
Anjos 300 4	603280369.5	603679094.5	60287336[illegible].5
Anjos 300 5	467360444	466172304	46616[illegible]315
Anjos 400 1	5007041358.5	5007312343.5	5000[illegible]2665.[illegible]
Anjos 400 2	2916949529	2917158558	29102[illegible]9558
Anjos 400 3	922299658	922534333	921216[illegible]92
Anjos 400 4	1809796198	1809847357	180[illegible]067255
Anjos 400 5	1404633352.5	1405722365.5	1402779504.5
Anjos 500 1	12318818681	123264[illegible]9609	1[illegible]00427281
Anjos 500 2	7514421872.5	7505998[illegible]55,5	7493143632,5
Anjos 500 3	2485064290	[illegible]	2479334789
Anjos 500 4	4294960811,5	[illegible]	4285937468,5
Anjos 500 5	3689059932,5	3680[illegible]97633,5	3678038149,5
sko_200 1	32331[illegible]4	3231656	3231383
sko_200 2	7758971	7758940	7758937
sko_200 3	[illegible]	12741126	12739045
sko_200 4	20[illegible]63543	20268410	20260611
sko_200 5	268733[illegible].5	26873161,5	26871979,5
sko_300 1	[illegible]1279331	11269514	11251962
sko_300 2	29[illegible]08482	29061358	28993837
sko_300 3	48875875,5	48899319,5	48800528,5
sko_300 4	71313267,5	71342723,5	71194215.5
sko_300 5	86834415,5	86820444,5	86792129,5
sko_400 1	26793018	26790436	26718566
sko_400 2	68058002	68111630	67996135
sko_400 3	116179989	116182660	115942730
sko_400 4	163542874	163544831	163099036
sko_400 5	228268699,5	228153858,5	227787876.5
sko_500 1	53118100	53130663	52952086
sko_500 2	127971045	127664532	127428491
sko_500 3	231573411,5	231487769,5	231044722,5
sko_500 4	341660232	340848573	340390687
sko_500 5	446537443	446702649	445750842

Agora comparamos GRASP-PR com GENALGO R e SS-2P R ao usar as (40) instâncias de teste de tamanhos 200, 300, 400 e 500, no Anjos-grande e sko-grande conjuntos. Para avaliar o desempenho desses três algoritmos em horizontes de tempo curtos e longos, nós os executamos para *n* e 3600 segundos. As Tabelas 3 e 4 mostram os custos associados, onde os melhores valores encontrados para cada uma das instâncias estão destacados em negrito.

Tabela 4. Comparação de algoritmos SRFLP em Anjos-grande e sko-grande instâncias, ao executá-los por uma hora.

Instância	GENALGO R	SS-2P R	GRASP-PR
Anjos 200 1	305497727	305461818	305461818
Anjos 200 2	178807197.5	178852677.5	178816261.5
Anjos 200 3	61892040	61891652	61891275
Anjos 200 4	127743257	127736464	127735691
Anjos 200 5	89059083,5	89141158,5	89057121,5
Anjos 300 1	1550443050	1550822258	1549663657
Anjos 300 2	955749527,5	957249994,5	955572066.5
Anjos 300 3	308372773.5	308701657.5	308257630,5
Anjos 300 4	603268519.5	603678289.5	6028731[illegible].5
Anjos 300 5	466717889	466162354	46616[illegible]264
Anjos 400 1	5004496747,5	5005963158.5	5000[illegible]2142.[illegible]
Anjos 400 2	2916949529	2916726055	[illegible]9102[illegible]6759
Anjos 400 3	921346203	922110067	92121[illegible]55
Anjos 400 4	1809520966	1809564367	180[illegible]061379
Anjos 400 5	1404580939.5	14051524[illegible],5	[illegible]402779472.5
Anjos 500 1	12311278683	123046[illegible]3851	[illegible]300409839
Anjos 500 2	7505406907.5	7501448[illegible]13,5	[illegible]493120635,5
Anjos 500 3	2483990108	2[illegible]831978[illegible]	2479333773
Anjos 500 4	4288172936.5	429[illegible]459[illegible].5	4285937468,5
Anjos 500 5	3685342697.5	3680[illegible]48707.5	3678038066.5
sko 200 1	32331[illegible]8	3231654	3231379
sko 200 2	7758[illegible]950	7758939	7758927
sko 200 3	1[illegible]7402[illegible]6	12741031	12739043
sko 200 4	20[illegible]63483	20268399	20260531
sko 200 5	[illegible]68732[illegible].5	26873153,5	26871976,5
sko 300 1	[illegible]1278708	11263368	11251960
sko 300 2	29[illegible]08472	29061348	28993831
sko 300 3	48828528,5	48894938,5	48800510,5
sko 300 4	71303673.5	71259866,5	71194203.5
sko 300 5	86834403,5	86806304.5	86792128,5
sko 400 1	26780355	26754928	26718485
sko 400 2	68058002	68046421	67996107
sko 400 3	116150616	116088506	115942726
sko 400 4	163359654	163371846	163099026
sko 400 5	228119749,5	228115330,5	227778641.5
sko 500 1	53030465	53117838	52951975
sko 500 2	127631162	127648186	127428477
sko 500 3	231465645,5	231408793,5	231041993,5
sko 500 4	341175620	340848111	340390652
sko 500 5	446498553	446641044	445738609

Nosso algoritmo proposto foi capaz de encontrar as melhores soluções em todas as sko-grande instâncias em ambos os cenários de tempo. Além disso, também obteve os melhores resultados em 19 (de 20) instâncias no conjunto Anjos-grande ao considerar horizontes de curto e longo prazo. Por outro lado, GENALGO R obteve o melhor resultado em apenas uma instância (em ambos os cenários de tempo), enquanto SS-2R-R encontrou o melhor valor (empate com

Tabela 5. Resultados do teste de Friedman aplicado aos dados da Tabela 3 e Tabela 4.

Tempo de execução	Classificação média			p- valor
	GENALGO R	SS-2P R	GRASP-PR	
n segundos	2,45	2,53	1.03	$3{,}9 \times 10^{-13}$
1 hora	2,50	2,46	1.04	$7{,}0 \times 10^{-13}$

Tabela 6. Resultados do teste de sinal estatístico aplicado aos dados da Tabela 3 em relação aos experimentos ao longo n segundos.

Algoritmos comparados	*assinar*	p- valor
GRASP-PR vs. GENALGO R _	39	$3{,}7 \times 10^{-11}$
GRASP-PR vs. SS-2P R _	40	$9{,}1 \times 10^{-13}$

Tabela 7. Resultados do teste de sinal estatístico aplicado aos dados da Tabela 4 em relação às experiências ao longo de 1 hora.

Algoritmos comparados	*assinar*	p- valor
GRASP-PR vs. GENALGO R _	39	$3{,}7 \times 10^{-11}$
GRASP-PR vs. SS-2P R _	39	$1{,}8 \times 10^{-12}$

GRASP-PR) em uma instância ao considerar 3600 segundos. Assim, percebe-se que GRASP-PR apresenta melhor desempenho.

A fim de apoiar a afirmação anterior, aplicamos um teste de Friedman aos dados das Tabelas 3 e 4. Este teste estatístico não paramétrico é semelhante ao teste ANOVA de medidas repetidas e é geralmente usado para detectar diferenças no desempenho do algoritmo no mesmo conjunto de instâncias. O teste classifica cada método em cada instância (linha) e, finalmente, considera os valores das classificações por algoritmos (colunas). Os resultados são mostrados na Tabela 5, onde as colunas do meio relatam a classificação média em todas as instâncias de cada abordagem. A classificação média do nosso método GRASP-PR é muito próximo de 1, o que reflete que geralmente obtém soluções de qualidade superior. O menor p- os valores indicam claramente que há evidência estatística suficiente para con fi rmar que há diferenças entre os três algoritmos.

Confirmada a existência de diferenças entre os métodos, foram realizados testes estatísticos não paramétricos de sinais para detectar diferenças consistentes entre GRASP-PR e os dois métodos anteriores. As Tabelas 6 e 7 resumem os resultados dos testes, onde *assinar* é o valor da estatística do sinal de teste, que indica o número de vezes que GRASP-PR supera o algoritmo de um concorrente. A coluna final relata o p- valores (para testes unilaterais) associados a cada experimento. Dado o baixo p- valores, o teste estatístico claramente suporta a superioridade do GRASP-PR sobre o algoritmo genético descrito em Kothari e Ghosh (2014a), e o procedimento de busca dispersa proposto em Kothari e Ghosh (2014b).

Nos experimentos anteriores, os algoritmos eram executados por um limite de tempo fixo. Para comparar os tempos de convergência, executamos SS-2R R e GENALGO R até

Tabela 8. Tempo de execução necessário para SS-2R R e GENALGO R para combinar ou aprimorar as soluções fornecidas por GRASP-PR após executar o algoritmo por uma hora.

Instância	Tempo de parada (em segundos) SS-2R R	GENALGO R
Anjos 200 1	2045	16087
Anjos 200 2	> 36000	754
Anjos 200 3	> 36000	> 36000
Anjos 200 4	> 36000	> 36000
Anjos 200 5	> 36000	7441
Anjos 300 1	> 36000	> 36000
Anjos 300 2	> 36000	27662
Anjos 300 3	> 36000	> 36000
Anjos 300 4	> 36000	> 36000
Anjos 300 5	7556	> 36000
Anjos 400 1	> 36000	> 36000
Anjos 400 2	> 36000	> 36000
Anjos 400 3	11454	10978
Anjos 400 4	> 36000	> 36000
Anjos 400 5	> 36000	13070
Anjos 500 1	> 36000	33609
Anjos 500 2	> 36000	32365
Anjos 500 3	> 36000	> 36000
Anjos 500 4	> 36000	24413
Anjos 500 5	[illegible]	> 36000
Sko 200 1	> 36000	> 36000
Sko 200 2	> 36000	> 36000
Sko 200 3	> 36000	> 36000
Sko 200 4	> 36000	17074
Sko 200 5	> 36000	> 36000
Sko 300 1	> 36000	> 36000
Sko 300 2	> 36000	> 36000
Sko 300 3	> 36000	> 36000
Sko 300 4	8246	10584
Sko 300 5	> 36000	> 36000
Sko 400 1	> 36000	> 36000
Sko 400 2	> 36000	27070
Sko 400 3	> 36000	> 36000
Sko 400 4	> 36000	13900
Sko 400 5	13691	29891
Sko 500 1	> 36000	33649
Sko 500 2	> 36000	> 36000
Sko 500 3	> 36000	> 36000
Sko 500 4	> 36000	> 36000
Sko 500 5	> 36000	> 36000

atingiram o valor atingido por nosso GRASP-PR após rodá-lo por uma hora nas novas instâncias (Tabela 4 indica os custos associados), ou por no máximo 10 horas. A Tabela 8 mostra os tempos de parada dos algoritmos de última geração, em que células contendo "> 36000" indicam que os métodos não podem corresponder ou aprimorar

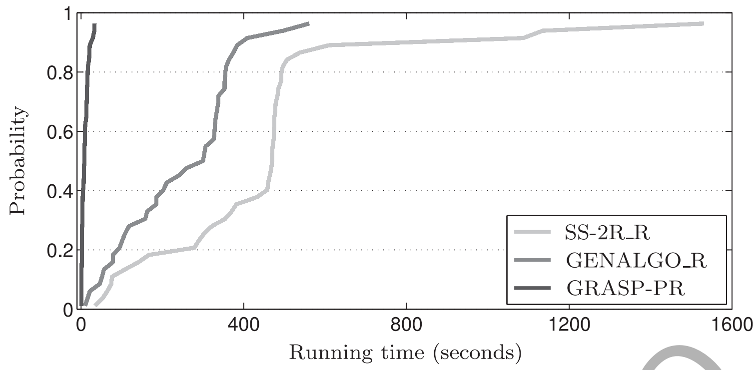


Figura 9. Gráfico de tempo para destino para SS-2R R, GENALGO R e GRASP-PR depois de executá-los 40 vezes na instância Anjos 200 1.

as soluções de GRASP-PR após funcionar por 10 horas. Observe que essa situação ocorreu em 59 dos 80 casos.

Por último, a Figura 9 mostra um gráfico TTT (tempo para atingir o objetivo) Aiex et al. (2006) resultante de 40 execuções com diferentes sementes aleatórias de SS-2R R, GENALGO R e GRASP-PR sobre a instância Anjos 200 1. Este gráfico mostra a probabilidade de atingir um valor de custo fixo (em particular, 305788558 para a instância) em um determinado momento, e ilustra o comportamento geral dos três algoritmos. É evidente que nossa abordagem converge para soluções de maior qualidade mais rapidamente do que os métodos de última geração.

7 Conclusões

Este artigo descreveu um algoritmo para encontrar soluções de alta qualidade para o SRFLP com base no acoplamento de um procedimento GRASP com uma abordagem de religação de caminho. Uma vez que algoritmos de última geração alcançam essencialmente os mesmos resultados nas instâncias usadas em toda a literatura, geramos novas instâncias de maior tamanho para comparar algoritmos. Em geral, nosso método proposto foi capaz de encontrar soluções melhores para essas instâncias de grande porte, em comparação com os métodos anteriores. As principais características do algoritmo são vários procedimentos de construção GRASP, uma nova estratégia de busca local rápida e uma abordagem relacionada à distância Ulam para construir trajetórias de religação de caminho. Além disso, nosso algoritmo não apenas obteve as melhores soluções relatadas anteriormente na literatura de forma eficiente,

Antes de iniciar o teste competitivo, realizamos uma única experimentação fatorial completa para determinar a contribuição dos vários elementos que projetamos. Acreditamos que o leitor pode considerá-los muito úteis, uma vez que lições valiosas podem ser aprendidas com eles e aplicadas a outros problemas. A extensa comparação final entre o GRASP proposto com abordagem de reconexão de caminho e os dois métodos mais bem identificados na literatura (um algoritmo genético (Kothari e Ghosh, 2012b) e um método de pesquisa dispersa (Kothari e Ghosh, 2012a)), revela que nosso O algoritmo é capaz de superar os métodos atuais de última geração em horizontes de curto e longo prazo. Em particular, nossa abordagem encontra os resultados mais conhecidos em instâncias usadas anteriormente em um tempo de computação consideravelmente mais curto. Além disso, ele produz soluções de alta qualidade em 39 de 40 instâncias ao executar os algoritmos para *n* segundos e 38 de 40 instâncias ao executá-los por uma hora. A superioridade do nosso método é ainda apoiada pelo baixo *p*- valores (abaixo de 10 - 11) associado a testes não paramétricos para detectar diferenças estatisticamente significativas entre os algoritmos.

Pesquisas futuras sobre o problema podem examinar outras metaheurísticas complexas (por exemplo, abordagens bioinspiradas como Ant Colony Optimization ou Arti fi cial Bee Colony), variantes da abordagem proposta ou algoritmos de teste em instâncias diferentes. Por exemplo, as matrizes de peso das instâncias desse problema são bastante esparsas (ou seja, elas contêm um grande número de zeros). Portanto, o desempenho desses algoritmos em instâncias com matrizes de peso denso ainda não foi estudado.

Finalmente, os melhores pedidos e custos encontrados para as instâncias no Anjos, sko e Amaral conjuntos, bem como no Anjos-grande e sko-grande conjuntos de teste (para *n* = 200, 300, 400 e 500) estão disponíveis em Rubio-S´ánchez et al. (2016).

Agradecimentos

Esta pesquisa foi parcialmente financiada pelo Ministério espanhol da “Econom´ıa y Competitividad” com subvenções ref. TIN2015-65460-C2-2-P, TIN2015-66731-C2-1-R e “Comunidad de Madrid” com ref. S2013 / ICE-2894.

Referências

Renata M. Aiex, Mauricio GC Resende e Celso C. Ribeiro. Plots Ttt: um perl programa para criar gráficos de tempo para atingir o objetivo. *Cartas de Otimização,* 1 (4): 355–366, 2006. ISSN 1862-4480. doi: 10.1007 / s11590-006-0031-4.

Andr´é RS Amaral. Sobre a solução exata de um problema de layout de instalação. *europeu Journal of Operational Research,* 173 (2): 508–518, 2006. ISSN 0377-2217. doi: 10.1016 / j.ejor.2004.12.021.

Andr´é RS Amaral. Uma abordagem exata para o layout unidimensional da instalação problema. *Pesquisa Operacional,* 56 (4): 1026–1033, agosto de 2008. doi: 10.1287 / opre.1080.0548.

Andr´é RS Amaral. Um novo limite inferior para o problema de layout de instalação de linha única. *Matemática Aplicada Discreta,* 157 (1): 183–190, janeiro de 2009. ISSN 0166-218X. doi: 10.1016 / j.dam.2008.06.002.

Andr´é RS Amaral e Adam N. Letchford. Uma abordagem poliédrica para o sin- Problema de layout de instalação de linha de gle. *Programação Matemática,* 141 (1-2): 453–477, outubro de 2013. ISSN 0025-5610. doi: 10.1007 / s10107-012-0533-z.

Miguel F. Anjos, 2016. URL www.miguelanjos.com/flplib.

Miguel F. Anjos e Anthony Vannelli. Soluções de computação globalmente ideais para problemas de layout de linha única usando programação semide fi nita e planos de corte. *INFORMS Journal on Computing,* 20 (4): 611–617, maio de 2008. doi: 10.1287 / ijoc. 1080.0270.

Miguel F. Anjos e Ginger Yen. Soluções provavelmente quase ótimas para grandes problemas de layout de instalação de linha única. *Software de métodos de otimização,* 24 (4-5): 805– 817, agosto de 2009. ISSN 1055-6788. doi: 10.1080 / 10556780902917735.

Miguel F. Anjos, Andrew Kennings e Anthony Vannelli. Um op- semidefinito abordagem de sincronização para o problema de layout de linha única com dimensões desiguais. *Otimização Discreta,* 2 (2): 113–122, junho de 2005. ISSN 1572-5286. doi: 10.1016 / j.disopt.2005.03.001.

V. Campos, R. Mart´ı, J. Sanchez-Oro e A. Duarte. GRASP com PR para o problema de orientação. *Jornal da Sociedade de Pesquisa Operacional,* 2013. doi: 10.1057 / jors.2013.156.

Dilip Datta, Andr´é RS Amaral e José Rui Figueira. Instalação de linha única problema de layout usando um algoritmo genético baseado em permutação. *European Journal of Operational Research,* 213 (2): 388–394, 2011. ISSN 0377-2217. doi: 10.1016 / j.ejor.2011.03.034.

A. Duarte, R. Mart´ı, MGC Resende e RMA Silva. GRASP com heurísticas de reconexão de caminho para o problema de largura de banda. *Redes,* 58 (3): 171–189, 2011.

TA Feo, MGC Resende e SH Smith. Um procedimento de busca adaptativa aleatória gananciosa para o conjunto máximo independente. *Pesquisa Operacional,* 42: 860–878, 1994. Thomas A. Feo e Mauricio GC Resende. Uma heurística probabilística para um com problema de cobertura de conjunto supostamente difícil. *Cartas de pesquisa operacional,* 8: 67-71, 1989. doi: 10.1016 / 0167-6377 (89) 90002-3.

F. Glover. Pesquisa tabu - parte 1. *ORSA Journal on Computing,* 1 (2): 190–206, 1989.

F. Glover. Pesquisa tabu - parte 2. *ORSA Journal on Computing,* 2 (1): 4-32,1990.

F. Glover, CC Kuo e KS Dhir. Algoritmos heurísticos para o problema de diversidade máxima. *Journal of Information and Optimization Sciences,* 19 (1): 109-132, 1998.

Fred Glover. Um modelo para pesquisa dispersa e reconexão de caminho. Dentro *Pa- selecionado pessoas da Terceira Conferência Europeia sobre Evolução Arti fi cial,* páginas 3–54. Springer-Verlag, 1998. ISBN 3-540-64169-6.

Sunderesh S. Heragu e Attahiru Sule Alfa. Análise experimental de um simulado algoritmos baseados em annealing para o problema de layout. *European Journal of Operational Research,* 57 (2): 190–202, 1992. ISSN 0377-2217. doi: 10.1016 / 0377-2217 (92) 90042-8.

Sunderesh S. Heragu e Andrew Kusiak. Problema de layout da máquina inflexível sistemas de manufatura. *Pesquisa Operacional,* 36 (2): 258-268, março de 1988. doi: 10.1287 / opre.36.2.258.

Sunderesh S. Heragu e Andrew Kusiak. Modelos eficientes para o layout da instalação problema. *European Journal of Operational Research,* 53 (1): 1-13, 1991. ISSN 0377-2217. doi: 10.1016 / 0377-2217 (91) 90088-D.

Philipp Hungerl¨ änder. Layout de instalação equidistante de linha única como um caso especial de layout de instalação de uma única linha. *International Journal of Production Research,* 52 (5): 1257–1268, 2014. ISSN 0020-7543. doi: 10.1080 / 00207543.2013.828163.

Philipp Hungerl¨ änder e Franz Rendl. Um estudo e pesquisa computacional de métodos para o problema de layout de instalação de linha única. *Op- computacional timização e aplicativos,* 55 (1): 1–20, maio de 2013. ISSN 0926-6003. doi: 10.1007 / s10589-012-9505-8.

James W. Hunt e Thomas G. Szymanski. Um algoritmo rápido para computação por mais tempo subseqüências comuns. *Comunicações da ACM,* 20 (5): 350–353, maio de 1977. ISSN 0001-0782. doi: 10.1145 / 359581.359603.

Birgit Keller e Udo Buscher. Modelos de layout de linha única. *European Journal of Pesquisa Operacional,* 245 (3): 629–644, 2015. ISSN 0377-2217. doi: 10.1016 / j.

ejor.2015.03.016. Aparecer.

Ravi Kothari e Diptesh Ghosh. O problema de layout de instalação de linha única: estado da arte. *OPSEARCH,* 49 (4): 442–462, dezembro de 2012a. ISSN 0030-3887. doi: 10.1007 / s12597-012-0091-4.

Ravi Kothari e Diptesh Ghosh. Revinculação de caminho para layout de instalação de linha única. Technical Report 2012-05-01, Indian Institute of Management, maio de 2012b.

Ravi Kothari e Diptesh Ghosh. Pesquisa tabu para o layout de instalação de linha única problema usando vizinhanças exaustivas de 2 opções e de inserção. Technical Report 2012-01-03, Indian Institute of Management, janeiro de 2012c.

Ravi Kothari e Diptesh Ghosh. Heurística de lin-kernighan baseada em inserção para um único layout de instalação de linha. *Computadores e Pesquisa Operacional,* 40 (1): 129–136, janeiro de 2013a. ISSN 0305-0548. doi: 10.1016 / j.cor.2012.05.017.

Ravi Kothari e Diptesh Ghosh. Pesquisa tabu para o layout de instalação de linha única problema usando vizinhanças exaustivas de 2 opções e de inserção. *European Journal of Operational Research,* 224 (1): 93–100, 2013b. ISSN 0377-2217. doi: 10.1016 / j. ejor.2012.07.037.

Ravi Kothari e Diptesh Ghosh. Um algoritmo genético eficiente para linha única layout das instalações. *Cartas de Otimização,* 8 (2): 679–690, fevereiro de 2014a. ISSN 1862-4472. doi: 10.1007 / s11590-012-0605-2.

Ravi Kothari e Diptesh Ghosh. Um algoritmo de busca dispersa para a única linha problema de layout da instalação. *Journal of Heuristics,* 20 (2): 125–142, abril de 2014b. ISSN 1381-1231. doi: 10.1007 / s10732-013-9234-x.

K. Ravi Kumar, George C. Hadjinicola e Ting li Lin. Um procedimento heurístico para o problema de layout de instalação de linha única. *European Journal of Operational Research,* 87 (1): 65–73, 1995. ISSN 0377-2217. doi: 10.1016 / 0377-2217 (94) 00062-H. Satheesh Kumar, Asokan P. Kumanan e S. Varma. Algoritmo de pesquisa dispersa para problema de layout de linha única no FMS. *Avanços em Engenharia e Gestão de Produção,* 3 (4): 193–204, 2008. ISSN 1854-6250.

M. Laguna e R. Mart´ı. GRASP e reconexão de caminho para minimização de cruzamento de linha reta de 2 camadas. *INFORMS Journal on Computing,* 11 (1): 44–52, 1999.

P. Larrañ˜aga, CMH Kuijpers, RH Murga, I. Inza e S. Dizdarevic. Algoritmos genéticos para o problema do caixeiro viajante: uma revisão das representações e operadores. *Artif. Intell. Rev.* 13 (2): 129-170, abril de 1999. ISSN 0269-2821. Robert F. Love e Jsun Y. Wong. Na resolução de uma alocação de espaço unidimensional problema com programação inteira. *INFOR: Sistemas de Informação e Pesquisa Operacional,* 14: 139–144, janeiro de 1976.

JJ Pantrigo, R. Mart´ı, A. Duarte e EG Pardo. Pesquisa de dispersão para o problema de minimização da largura de corte. *Anais de Pesquisa Operacional,* 199 (1): 285–304, 2012. Jean-Claude Picard e Maurice Queyranne. No espaço unidimensional problema de alocação. *Pesquisa Operacional,* 29 (2): 371-391, abril de 1981. doi: 10.1287 / opre.29.2.371.

AS Ramkumar e SG Ponnambalam. Projeto de layouts de linha única para sistemas de manufatura flexíveis usando algoritmo genético e algoritmo de recozimento simulado. Dentro *Conferência IEEE de 2004 sobre Cibernética e Sistemas Inteligentes,* volume 2, páginas 1143–1147, dezembro de 2004. doi: 10.1109 / ICCIS.2004.1460751.

MGC Resende, R. Mart´ı, M. Gallego e A. Duarte. GRASP e reconexão de caminho para o problema de diversidade máximo-mínimo. *Computadores e Pesquisa Operacional,* 37 (3): 498–508, 2010.

Mauricio GC Resende e Celso C. Ribeiro. Busca adaptativa aleatória gananciosa procedimentos: avanços, hibridizações e aplicações. Em M. Gendreau e J.-Y. Potvin, editores, *Manual de metaheurísticas,* volume 146 de *Série Internacional em Pesquisa Operacional e Ciência de Gestão,* páginas 283–319. Springer US, 2ª edição, 2010.

Mauricio GC Resende e Celso C. Ribeiro. GRASP: Greedy randomized adap- procedimentos de busca ativa. Em EK Burke e G. Kendall, editores, *Metodologias de pesquisa,* páginas 287–312. Springer US, 2014.

Mauricio GC Resende e RF Werneck. Uma heurística híbrida para a p-mediana problema. *Journal of Heuristics,* 10 (1): 59–88, 2004. ISSN 1381-1231.

David Romero e Adolfo S´ánchez-Flores. Métodos para o espaço unidimensional problema de alocação. *Computadores e Pesquisa Operacional,* 17 (5): 465–473, junho de 1990. ISSN 0305-0548. doi: 10.1016 / 0305-0548 (90) 90051-8.

Manuel Rubio-S´ánchez, Micael Gallego, Francisco Gort´ázar e Abraham Duarte, 2016. URL www.optsicom.es/srflp/.

Hamed Samarghandi e Kourosh Eshghi. Um algoritmo tabu eficiente para o único problema de layout de instalação de linha. *European Journal of Operational Research,* 205 (1): 98–105, 2010. ISSN 0377-2217. doi: 10.1016 / j.ejor.2009.11.034.

Hamed Samarghandi, Pouria Taabayan e Farzad Firouzi Jahantigh. Uma partícula otimização de enxame para o problema de layout de instalação de linha única. *Computadores & Engenharia Industrial,* 58 (4): 529–534, 2010. ISSN 0360-8352. doi: 10.1016 / j.cie. 2009.11.015.

Sujeevraja Sanjeevi e Kiavash Kianfar. Um estudo poliédrico da formulação tripla para problema de layout de instalação de linha única. *Matemática Aplicada Discreta,* 158 (16): 1861–1867, 2010. ISSN 0166-218X. doi: 10.1016 / j.dam.2010.07.005.

Marc Sevaux e Kenneth S¨örensen. Medidas de distância de permutação para memética algoritmos com gestão de população. Dentro *MIC2005, Proceedings of 6th Metaheuristics International Conference,* páginas 832–838, 2005.

Donald M. Simmons. Alocação de espaço unidimensional: Um algoritmo de ordenação. *Pesquisa Operacional,* 17 (5): 812–826, outubro de 1969. doi: 10.1287 / opre.17.5.812.

M. Solimanpur, Prem Vrat e Ravi Shankar. Um algoritmo de formiga para o problema de layout de linha única em sistemas de manufatura flexíveis. *Computadores e Pesquisa Operacional,* 32 (3): 583–598, 2005. ISSN 0305-0548. doi: 10.1016 / j.cor.2003.08.005.

SM Ulam. Algumas ideias e perspectivas em biomatemática. *Revisão anual de biofísica e bioengenharia,* 1: 277–292, 1972. ISSN 0084-6589. doi: 10.1146 / annurev.bb.01.060172.001425.

(M. Rubio-Sá´nchez) Departamento de Ciência da Computação e Estatística, Universidad Rey Juan Carlos, Espanha.

Endereço de e-mail: manuel.rubio@urjc.es

(M. Gallego) Departamento de Ciência da Computação e Estatística, Universidade Rey Juan Carlos, Espanha.

Endereço de e-mail: micael.gallego@urjc.es

(F. Gortazar) Departamento de Ciência da Computação e Estatística, Universidade Rey Juan Carlos, Espanha.

Endereço de e-mail: francisco.gortazar@urjc.es

(A. Duarte) Departamento de Ciência da Computação e Estatística, Universidade Rey Juan Carlos, Espanha.

Endereço de e-mail: abraham.duarte@urjc.es

MANUSCRITO ACEITO