UM ALGORITMO GENÉTICO COM DESCONSTRUÇÃO E RECONSTRUÇÃO PARCIAL DA SOLUÇÃO PARA O PROBLEMA DE LAYOUT DE FACILIDADES COM ÁREAS DIFERENTES

Frederico Galaxe Paes
Instituto Federal Fluminense – IFF
Av. Souza Mota, 350, Pq. Fundão, 28060-010, Campos dos Goytacazes, RJ
fpaes@iff.edu.br

Artur Alves Pessoa
Departamento de Engenharia de Produção – Universidade Federal Fluminense
Rua Passo da Pátria 156, São Domingos, 24210-240, Niterói, RJ
artur@producao.uff.br

Thibaut Vidal
Departamento de Informática – PUC–Rio
Rua Marquês de São Vicente, 225, Gávea, 22451-900, Rio de Janeiro, RJ
vidalt@inf.puc-rio.br

RESUMO

Este artigo apresenta um algoritmo genético (GA) baseado em uma estratégia gulosa de localização para resolver o problema de layout de facilidades com áreas diferentes (PLFAD), onde um conjunto de facilidades retangulares tem que ser localizadas, sem sobreposição, em um espaço ilimitado. Dados as possíveis dimensões dos retângulos e os fluxos entre eles, o objetivo é encontrar as suas dimensões e posições de modo a minimizar a soma dos produtos dos fluxos entre as facilidades pelas distâncias entre os seus centróides. É apresentado também um método de decomposição que visa aprimorar isoladamente partes da melhor solução corrente, similarmente ao método de Taillard e Voss (2002), para tentar escapar de mínimos locais. A abordagem proposta foi capaz de produzir boas soluções, superando a melhor solução conhecida da literatura (BKS) em 5 das 8 instâncias testadas, em um tempo de processamento competitivo. Usando a nova estratégia de localização, a abordagem proposta obteve um bom desempenho, especialmente na resolução de instâncias de médio e grande porte com mais de 50 facilidades. Para este tamanho de instâncias, métodos exatos são atualmente impraticáveis.

PALAVRAS CHAVE: Layout, Algoritmo Genético, Metaheurística.

ABSTRACT

This paper presents a genetic algorithm (GA) based in a greedy strategy of placement, to solve the unequal area facility layout problem (UA-FLP), where a set of rectangular facilities has to be placed, without overlapping, in an unlimited floor space. Given the possible rectangle dimensions and the flows among them, the objective is to find their dimensions and locations in order to minimize the sum of the distances between their centroids weighted by the corresponding flows. We also introduce a decomposition method, which aims to improve parts of the current best solution, in a similar way as Taillard e Voss (2002), hence helping to escape from local minimums. The proposed approach was able to produce good solutions, improving 5 out of 8 best known solutions(BKS) from the literature. The overall CPU time remains similar to previous algorithms. Using the new decomposition strategy, the approach performs especially well for medium- and large-scale instances with more than 50 facilities. For this size of instances, exact methods are currently impracticable.

KEY WORDS: Layout, Genetic Algorithm, Metaheuristic.

1 Introdução

O Problema de Layout de Facilidades (PLF) é um conhecido problema de otimização combinatória, que busca determinar a localização de facilidades retangulares no plano de modo que não haja sobreposição entre elas, enquanto algum objetivo é otimizado, em geral minimizar o custo de manuseio de material. Uma boa localização das facilidades contribui para uma eficiência global das operações e pode reduzir significativamente as despesas operacionais (Drira *et al*., 2007). Os PLFs podem ser classificados como estáticos ou dinâmicos, com as facilidades podendo ter áreas iguais ou diferentes bem como dimensões flexíveis ou rígidas. Além disso, o espaço onde as facilidades serão localizadas pode ser ilimitado ou limitado. Uma revisão de diferentes abordagens registradas na literatura para resolver PLFs pode ser encontrada em Drira *et al*. (2007).

Neste artigo, é abordada a variante estática do PLF, com área disponível ilimitada e contínua, envolvendo tanto facilidades rígidas como flexíveis e com áreas diferentes, conhecido como PLFAD. Dado um fluxo de material entre cada par de facilidades, o problema tem como função objetivo minimizar o custo de manuseio de material, calculado como a soma dos produtos entre as distâncias entre os centróides e os fluxos correspondentes. Como restrições, temos as limitações das dimensões das facilidades flexíveis, bem como garantir que não haja sobreposição entre as facilidades. O PLFAD é difícil de ser resolvido de forma exata pois apresenta características similares ao problema quadrático de alocação (PQA) (Sahni e Gonzalez, 1976). Portanto, métodos analíticos, heurísticas e metaheurísticas têm sido utilizados para resolver este problema.

Dentre as heurísticas encontradas na literatura para o problema, destacamos o algoritmo de busca *Cluster Boundary* (CBA) apresentado em Imam e Mir (1998), que procura a posição ótima de cada novo bloco através de uma busca uni-dimensional por partes na fronteira formada pelo cluster de blocos previamente localizados. Testes computacionais realizados demonstram a eficiência da técnica para instâncias de referência. Em Kado (1996) foram desenvolvidos seis tipos de algoritmos genéticos usando estrutura de representação do tipo *slicing-tree*. Em Dunker *et al*. (2003) foi desenvolvido um abordagem co-evolucionária utilizando algoritmo genético. Os autores apresentam operadores de mutação e crossover melhores, agrupando os departamentos no intuito de resolverem problemas maiores. Um algoritmo de *busca tabu* com uma representação de *slicing-tree* e incorporando uma curva de limite é proposto em Scholz *et al*. (2009) para resolver PLFAD com facilidades fixas e flexíveis. O algoritmo proposto incorporou quatro tipos de movimentos (na vizinhança) para encontrar melhores soluções. Posteriormente em McKendall e Hakobyan (2010), através de uma modificação feita no CBA para resolver problemas de layout de facilidades dinâmico (PLFD), obteve uma heurística de construção chamada de *Boundary Search* (BSH). Após construir uma solução com o BSH, os autores utilizaram a heurística busca tabu para melhorar a solução e encontraram bons resultados para algumas instâncias de referência do problema estático (PLFE). Uma abordagem híbrida de otimização por enxame de partículas e busca local é proposta em Kulturel-Konak e Konak (2011) para resolver o PLFAD usando uma estrutura de baias flexíveis relaxadas (RFBS). Mais recentemente, Gonçalves e Resende (2014) publicaram um relatório técnico onde propuseram uma abordagem híbrida combinando um *biased random-key genetic algorithm* – BRKGA, uma nova estratégia de localização para o posicionamento das facilidades e um modelo de programação linear para ajuste fino das soluções. Dentre 28 instâncias de referência da literatura, os autores conseguiram superar a melhor solução conhecida em 19 instâncias, comprovando a qualidade da abordagem proposta. Além disso, para a variante do problema tratada neste artigo, onde a área disponível é ilimitada, o modelo de programação linear não era utilizado e os custos de todas as melhores soluções da literatura foram reduzidas por percentuais que variam entre 1,86% e 11,05%. Acreditamos que a abordagem apresentada por Gonçalves e Resende (2014) é a mais adequada para o PFLAD com área ilimitada dentre as encontradas na literatura e, por isso, desenvolvemos nosso trabalho em cima desta abordagem visando melhorar ainda mais a qualidade da solução obtida sem deteriorar significativamente o tempo computacional.

Deste modo, este artigo traz as seguintes contribuições: (i) apresentar uma estratégia de decomposição de uma parcela da melhor solução para tentar escapar de ótimos locais, similar ao algoritmo de Taillard e Voss (2002); (ii) integrar este método com um algoritmo genético (GA) que utiliza um procedimento conhecido de avaliação para a construção gulosa da solução, chamado de *Empty Maximal-Spaces* (EMS), do mesmo modo como foi feito em Gonçalves e Resende (2014); (iii) Realizar uma extensiva análise experimental em instâncias da literatura. Em particular, o método foi capaz de superar os resultados da literatura em instâncias de médio e grande porte com mais de 50 facilidades, e com menor tempo de processamento, conforme revelaram os experimentos computacionais.

O restante do artigo é organizado como segue. A seção 2 apresenta a definição do problema. A Seção 3 descreve um GA incluindo a geração da população inicial e os operadores. A Seção 4 introduz a estratégia de localização das facilidades e o gerenciamento dos espaços máximos vazios (EMV). A Seção 5 descreve o modo como é feita a decomposição da melhor solução S^*. Na seção 6 é apresentado o pseudo-código do algoritmo proposto. Na Seção 7 são comparados e discutidos os resultados computacionais com outros resultados relevantes da literatura e finalmente, na Seção 8 são apresentadas as conclusões.

2 Definição do Problema

Seja o problema de localizar os centróides (x_i, y_i) de n facilidades retangulares de áreas diferentes em um plano ilimitado, sem sobreposição. Cada facilidade $i = 1, \ldots, n$ é definida por sua largura l_i, seu comprimento h_i, sua área $A_i = l_i \times h_i$ e o *aspect ratio* $AR_i = l_i/h_i$, dado pelo quociente entre sua largura e sua altura. Por razões práticas, assume-se que $AR_i^{min} \leq AR_i \leq AR_i^{max}$. Desta forma, um layout fica determinado pelas coordenadas dos centróides e pelas dimensões (largura e altura) de cada facilidade i, enquanto que a área e os *aspect ratios* mínimo e máximo são dados de entrada do problema. A função custo a ser minimizada é:

$$Custo = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} f_{ij} d_{ij} \tag{1}$$

Na equação (1), f_{ij} é o fluxo de material e d_{ij} é a medida da distância entre os centróides (x_i, y_i) e (x_j, y_j), das facilidades i e j respectivamente. De acordo com a métrica utilizada, a medida dessas distâncias pode ser calculada utilizando uma das seguintes equações:

1.Distância Retilínea (DR): $$d_{ij} = |x_i - x_j| + |y_i - y_j| \tag{2}$$

2.Distância Quadrado Euclidiano (DQE): $$d_{ij} = (x_i - x_j)^2 + (y_i - y_j)^2 \tag{3}$$

3.Distância Euclidiana (DE): $$d_{ij} = [(x_i - x_j)^2 + (y_i - y_j)^2]^{\frac{1}{2}} \tag{4}$$

O algoritmo proposto neste artigo, resolve instâncias do PLFAD cuja métrica de distância pode ser qualquer uma das equações (2) – (4). Um resumo das instâncias de referência da literatura utilizadas com suas respectivas métricas, bem como uma descrição das características das facilidades de cada instância, é apresentado na Tabela 1.

3 Algoritmo Genético (GA) para o PLFAD

Um algoritmo genético (GA) é uma conhecida metaheurística pertencente à classe dos algoritmos evolucionários (AE), que imita o processo de evolução natural usando técnicas tais como herança, mutação, seleção e *crossover* e é frequentemente usada para gerar boas soluções em problemas de otimização. GAs trabalham com uma família de soluções, conhecida como a população atual $P(t)$, da qual nós obtemos a próxima geração $P(t+1)$ de soluções. Quando o algoritmo é

Instância	Métrica	Descrição
L020	R	Instâncias com 20, 28, 50, 100 e 125 facilidades respectivamente.
L028	QE	Dimensões rígidas, isto é, AR_{min}=AR_{max}.
L050	E	
L100	R	
L125B	R	
L062	R	Instância com 62 facilidades permitindo rotação e de dimensões rígidas.
L075	R	Instâncias com 75 e 125 facilidades respectivamente.
L125A	R	Algumas facilidades possuem dimensões flexíveis, isto é, $AR_{min} \neq AR_{max}$.

Tabela 1: Instâncias de referência da literatura do caso irrestrito

projetado corretamente, nós obtemos progressivamente melhores soluções de uma geração para a outra. Desta forma, ele aumenta as chances de alcançar uma solução ótima global sem cair em uma solução ótima local. Alguns artigos, tais como Cohoon *et al*. (1991) e Tam (1992), apresentam aplicações bem sucedidas de GAs em PLFs. Assim, GA pode ser uma ferramenta promissora para resolver PLFs (Cohoon *et al*. (1991); Dunker *et al*. (2003); Kado (1996)).

3.1 População Inicial $P(0)$

O tamanho da população representa, indiretamente, o número de layouts candidatos que serão processados no intuito de produzir o melhor layout ao fim de todas as gerações. No algoritmo proposto o tamanho da população permanece fixo durante todas as gerações. Um cromossomo será representado por uma permutação de n inteiros positivos, onde n é o número de facilidades, que serão localizadas no plano por meio de um procedimento construtivo. O processo de geração da população inicial $P(0)$ se dá da seguinte forma:

Passo 1: Gere um array com n inteiros representando as facilidades a serem inseridas;

Passo 2: Para cada facilidade $i = 1, \ldots, n$, calcule o quociente A_i/FT_i, onde A_i é a área da facilidade i e FT_i é a soma dos fluxos entre a facilidade i e as demais facilidades $j = 1, \ldots, n$, com $i \neq j$. Em seguida, ordene o array em ordem crescente com relação ao valor do quociente;

Passo 3: Perturbe a ordem do array fazendo n trocas entre suas posições. A partir do 1º elemento do array, selecione aleatoriamente um elemento de um bloco de tamanho α ($\alpha = 5, 10$ ou ∞), remova-o, desloque todos os elementos à sua esquerda dentro do bloco, uma posição para a direita e insira o elemento removido na 1ª posição do bloco. Este processo continua até o n-ésimo elemento do array;

Passo 4: Repita este procedimento até que toda população inicial $P(0)$ seja gerada.

A ideia é permitir que as facilidades com menor área sejam inseridas no início, com maior frequência ($\alpha = 5$), frequência média ($\alpha = 10$) ou baixa frequência ($\alpha = \infty$, quando o indivíduo é gerado de modo totalmente aleatório). O Passo 3 assemelha-se, de certo modo, a uma lista de candidatos restrita (LCR) utilizada pela metaheurística GRASP na fase de construção, onde o parâmetro α pode ser interpretado como o tamanho da LCR.

3.2 Seleção e Crossover

A seleção é o primeiro operador do GA aplicado à população. No algoritmo proposto, a seleção dos pais para recombinação é feita de maneira aleatória com probabilidade uniforme, não sendo levado em consideração a aptidão do indivíduo como é feito em outros métodos como, por exemplo, o método da roleta viciada. Desta forma, permite-se que indivíduos menos aptos tenham mais chances de serem selecionados.

O *crossover*, por sua vez, é o processo pelo qual dois indivíduos são misturados de modo aleatório para produzir um filho. Ao contrário dos métodos tradicionais que só recombinam pais com uma probabilidade P_{cross}, em nossa abordagem sempre que dois pais forem selecionados serão submetidos ao processo de recombinação para gerar um novo filho. É importante notar que, em

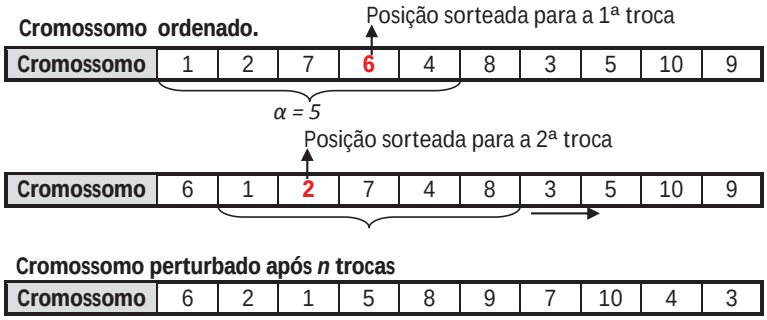


Figura 1: Perturbação do cromossomo para $\alpha = 5$.

função da representação dos cromossomos, deve-se optar por um operador de *crossover* que evite a geração de elementos repetidos no filho, bem como procure preservar no filho a ordem relativa dos genes nos pais.

Embora na literatura existam diversos tipos de *crossover* baseado em ordem, em nossa abordagem optou-se pelo *Position-based Crossover* (PX), um operador bem conhecido proposto por Syswerda e Palmucci (1991) que procura preserva a sequência dos genes no filho. O PX é um tipo de variação do *Order Crossover* (OX) onde os genes selecionados não estão dentro de um único bloco. O modo como o operador PX trabalha é descrito no pseudo-código a seguir:

Passo 1: Escolha aleatoriamente um pai atribuindo a mesma probabilidade a cada um dos dois indivíduos que serão recombinados. Selecione N_{herd} genes deste pai aleatoriamente, onde $N_{herd} = \frac{n}{2} + \frac{N_{dif}}{4}$, n representa o número de genes do cromossomo e N_{dif} o número de alelos diferentes nos dois pais;

Passo 2: Copie o conteúdo destes genes para os genes correspondentes no filho;

Passo 3: Remova os genes que foram selecionados no **passo 1**, no segundo pai. A sequência resultante contém os genes que o filho precisa;

Passo 4: Copie o conteúdo na mesma ordem da sequência resultante, da esquerda para a direita, para as posições vazias no filho.

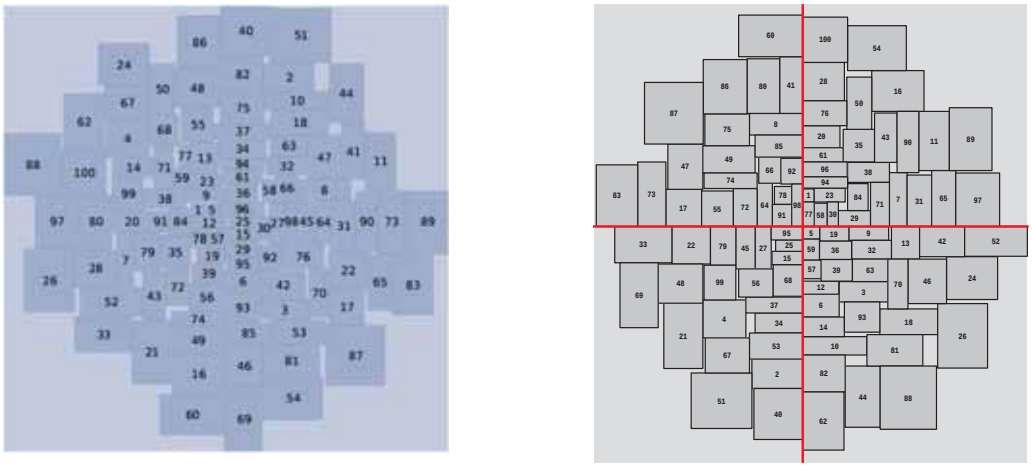
Esta estratégia baseia-se na observação de que no início das gerações, quando os indivíduos ainda são muito diferentes, gerar um filho que possua 75% do pai selecionado ajuda a evitar um tempo excessivo até a convergência. Por outro lado, quando a população começar a convergir e os indivíduos ficarem muito parecidos, o valor de N_{herd} se aproximará de 50% de n e, desta forma, selecionando-se 50% de cada pai evita-se a geração de muitos filhos repetidos.

4 Estratégia de Localização da Facilidade na Construção da Solução

A estratégia de localização utilizada neste artigo segue quase a mesma estratégia de Gonçalves e Resende (2014), onde um algoritmo construtivo guloso localiza as facilidades, uma de cada vez segundo a ordem em que aparecem no cromossomo, em uma posição no plano que produza o menor custo parcial. A principal diferença de nossa abordagem é a transformação do plano contínuo em um plano cartesiano de modo que as facilidades só podem ser inseridas em um dos quatro quadrantes, não sendo permitido a interseção entre uma facilidade e um dos eixos imagináros, como mostrado na Figura 2(b).

4.1 Espaços Máximos e o Processo de Diferença

Para localizar as facilidades no plano, é gerada uma lista $\mathcal{L}$ de espaços máximos vazios (EMV) (*empty maximal-spaces* – EMS), que são os lugares disponíveis para inserção da facilidade sem que haja sobreposição com as facilidades já alocadas. Um EMV e contido em $\mathcal{L}$ é representado



(a) BRKGA: Custo=478910,09 Gonçalves e Resende (2014).

(b) GADR: Custo=472157,67; T=31,42s.

Figura 2: Comparação entre as estratégias para a instância L100.

pelos seus vértices (x^e_{min}, y^e_{min}) e (x^e_{max}, y^e_{max}). Neste processo, só serão testados aqueles EMVs onde a facilidade que está sendo inserida se ajusta. Para gerar e atualizar os EMVs é utilizada uma técnica conhecida como *difference process* (DP), desenvolvida por Lai e Chan (1997). Além disso, um tratamento especial deve ser dado à primeira facilidade inserida, que dependerá da instância tratada. No momento da inserção dessa facilidades o espaço disponível está dividido em quatro EMVs, como mostra a Figura 3(a), devido à divisão do espaço em quadrantes. Assim, podemos classificar a inserção da primeira facilidade em um dos seguintes casos:

1. Se as facilidades da instância forem rígidas, i.e. $AR_{min} = AR_{max}$, e não admitirem rotação, a primeira facilidade é sempre inserida no EMV1 de modo que o lado referente a sua largura esteja sobre o eixo OX e o lado referente a sua altura esteja sobre o eixo OY, como mostrado na Figura 3(b);

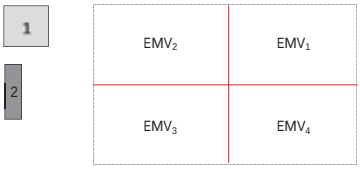
2. Se a instância possuir facilidades não-rígidas, i.e. $AR_{min} \neq AR_{max}$, a inserção da primeira facilidade seguirá a seguinte estratégia: se a primeira facilidade for rígida insere como em (1); caso contrário, compara-se com a segunda facilidade calculando ambas as dimensões com o AR_{min} e depois com o AR_{max}. O AR que produzir a menor distância horizontal ou vertical entre os centroides, determinará as dimensões da primeira facilidade. Desde que $AR_i = \frac{l_i}{h_i}$, as dimensões da facilidade i são respectivamente determinadas por

$$l_i = \sqrt{A_i \times AR_i} \qquad \text{e} \qquad h_i = \frac{A_i}{l_i} \tag{5}$$

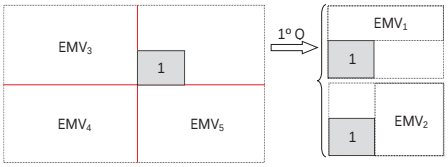
3. Se a instância permitir rotação das facilidades, a primeira facilidade pode ser inserida tanto com o maior lado na vertical sobre o eixo OY, como com o maior lado na horizontal sobre o eixo OX.

Na Figura 3, é ilustrado um exemplo de como são atualizados os EMVs através do DP. Assuma que existam duas facilidades a serem inseridas. Inicialmente, o plano é dividido em quatro EMVs e a primeira facilidade será sempre inserida no EMV1, segundo um dos casos descritos anteriormente (Figura 3(a)). Após a inserção da facilidade 1, dois novos EMVs são gerados pelo DP (Figura 3(b)) e a lista $\mathcal{L}$ de EMVs é atualizada de modo a eliminar EMVs muito estreitos ou aqueles que estão totalmente inscritos em outros EMVs (Lai e Chan, 1997), ficando com cinco EMVs. A

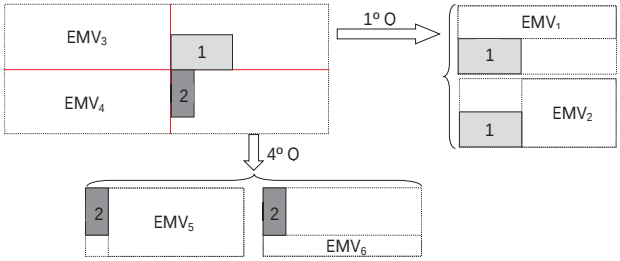
facilidade 2 é então inserida no EMV5, dois novos EMVs são gerados pelo PD e novamente a lista $\mathcal{L}$ é atualizada ficando com seis EMVs conforme visto na Figura 3(c). Este processo continua até que todas as facilidades sejam inseridas.



(a) Facilidades a serem inseridas e espaços máximos vazios iniciais candidatos.



(b) Após inserir a facilidade 1, dois novos EMVs são gerados pelo *difference process*, EMV1 e EMV2.



(c) Após inserir a facilidade 2, dois outros novos EMVs são gerados pelo *difference process*, EMV5 e EMV6..

Figura 3: Exemplo de geração de novos EMVs com o *difference process*(DP).

4.2 Otimizando a Inserção da Facilidade no EMV

Para localizar uma facilidade i em um determinado EMV, devemos calcular o ótimo irretrito (OI) de acordo com a métrica usada (Heragu, 1997). O OI é o ponto (x_I^*, y_I^*) que otimiza a localização de uma única facilidade, mas devido à sobreposição este ponto pode não estar acessível. Para contornar este problema, a facilidade i é localizada inicialmente no centro geométrico do EMV selecionado, dado pelo ponto $\left(\frac{EMVx_{min}+EMVx_{max}}{2}, \frac{EMVy_{min}+EMVy_{max}}{2}\right)$. A posição final da facilidade i é obtida por mover seu centróide o mais próximo possível do OI sem ultrapassar as fronteiras do EMV. Primeiro, tenta-se encontrar uma posição mais próxima possível do OI movendo verticalmente e depois movendo horizontalmente ou vice-versa (Gonçalves e Resende (2014)). Algumas vezes não é possível calcular o OI, devido ao fluxo entre a facilidade i que está sendo inserida e todas as outras facilidades já alocadas ser igual a zero. Quando isso ocorre, assume-se que o OI é dado pelo centro geométrico de todas as facilidades já dispostas, calculado pela expressão (6), onde M é o conjunto das facilidades já alocadas.

$$(OI_x, OI_y) = \left(\frac{1}{|M|}\sum_{m\in M} x_m, \frac{1}{|M|}\sum_{m\in M} y_m\right) \tag{6}$$

Se a instância tratada possuir facilidades com $AR_{min} \neq AR_{max}$, suas dimensões serão obtidas através do AR otimizado, dado pelo modelo não-linear (7) – (10), onde as variáveis contínuas Δl e Δh representam, respectivamente, as variações sofridas pelas coordenadas temporárias x_0 e y_0 do

centróide da facilidade inserida, de modo a otimizar a função custo parcial C (11).

$$\text{Min} \quad C_x(x_0, y_0)\Delta l + C_y(x_0, y_0)\Delta h \tag{7}$$

s.a:

$$(l + 2\Delta l)(h + 2\Delta h) = A \tag{8}$$

$$\Delta l_{mín} \leq \Delta l \leq \Delta l_{máx} \tag{9}$$

$$\Delta h_{mín} \leq \Delta h \leq \Delta h_{máx} \tag{10}$$

A função objetivo (7) é uma aproximação de primeira ordem da função custo parcial C no ponto (x_0, y_0), uma vez que C não é diferenciável nos centróides. No modelo, $C_x(x_0, y_0)$ e $C_y(x_0, y_0)$ são, respectivamente, as coordenadas do subgradiente $g \in \partial C(x_0, y_0)$, avaliadas em (x_0, y_0). A função custo parcial contabiliza o custo associado às distâncias entre a facilidade i sendo inserida e todas as facilidades já alocadas. As novas dimensões da facilidade inserida ficam determinadas fazendo $l_i = l + \Delta l^*$ e $h_i = h + \Delta h^*$, onde o par $(\Delta l^*, \Delta h^*)$ denota uma solução ótima para o modelo (7) – (10), que pode ser resolvido analiticamente utilizando relaxação lagrangeana.

5 Estratégia de Desconstrução e Reconstrução

Uma vez que o GA tenha convergido, inicia-se um processo de busca local similar ao algoritmo de Taillard e Voss (2002). Enquanto este realiza a busca local em uma vizinhança da solução utilizando uma heurística específica, nossa estratégia realiza uma busca na vizinhança da melhor solução encontrada S^*, utilizando o próprio GA.

A estratégia funciona da seguinte maneira: assim que o algoritmo convergir pela primeira vez, é gerada uma nova população de tamanho $nPop$ formada por soluções vizinhas da melhor solução até o momento ($P(t+1) \subset N(S^*)$), perturbando-se de modo idêntico ao Passo 3 do pseudo-código da Seção 3.1, apenas as facilidades com coordenadas no primeiro quadrante $x > 0$ e $y > 0$, mantendo as demais facilidades fixas. A partir daí, o GA atua recombinando os indivíduos (vizinhos de S^*) desta nova população, geração após geração, até a convergência da população. Se alguma solução melhor for encontrada ao logo desse processo, atualiza-se a solução corrente e uma nova vizinhança é gerada. Novamente, todo o processo descrito anteriormente é repetido perturbando-se apenas os genes cujas facilidades possuem coordenadas no segundo quadrante ($x < 0$ e $y > 0$), depois no terceiro quadrante ($x < 0$ e $y < 0$) e por último no quarto quadrante ($x > 0$ e $y < 0$). Portanto, toda vez que o algoritmo convergir em uma determinada geração, o procedimento de desconstrução e reconstrução atua em um determinado quadrante.

É importante notar que o esforço computacional necessário para pesquisar toda a vizinhança $N(S^*)$ de uma solução é bem maior devido ao seu tamanho. Por outro lado, perturbando-se apenas a região da solução contida em um dos quatro quadrantes produzirá uma vizinhança bem menor do que a anterior reduzindo, desta forma, o esforço computacional da busca. Tal fato, reflete diretamente no tempo necessário para realizar a busca local nos quadrantes, que é bem menor que no primeiro caso.

6 Pseudocódigo do Algoritmo

A seguir, são mostrados os pseudocódigos dos algoritmos utilizados para resolver o PLFAD. O algoritmo GADR (Algoritmo 1) recebe como parâmetros o tamanho da população $nPop$, o tamanho do bloco que imita a LCR para gerar os indivíduos da população inicial α, o Critério Guloso CG, que avalia o melhor local para a inserção de uma determinada facilidade no momento da construção de uma solução e o parâmetro N_{iter} que controla o número de vezes que cada quadrante da melhor solução será desconstruido e reconstruido. Os critérios gulosos (CG) utilizados pelo algoritmo serão discutidos na Seção 7. O algoritmo GA (Algoritmo 2), por sua vez, recebe como parâmetros a população de uma determinada geração $P(t)$, CG e $nPop$.

Na linha 3 do GADR é gerada uma população inicial $P(0)$ com $nPop$ indivíduos, conforme descrito no pseudocódigo da Seção 3.1. Na linha 4, para cada cromossomo, uma solução é construída

inserindo-se n facilidades, na ordem em que aparecem no cromossomo, na posição com o menor valor retornado pela função definida pelo parâmetro CG. Para avaliar este indivíduo, é utilizada como função de aptidão a equação (1). O GA é chamado na linha 5, onde os indivíduos de $P(0)$ serão recombinados para gerar $nPop$ filhos. Após todos os filhos terem sido gerados, teremos uma população com $2 \times nPop$ indivíduos, dentre os quais, serão selecionados $nPop$ indivíduos com o menor custo (linha 8 do Algoritmo 2). Esse processo se repete até a convergência da população.

Algoritmo 1 GADR($nPop$, α, CG, N_{iter})

1: {Fase 1 do algoritmo}
2: $t \leftarrow 0$;
3: **Gere** a população inicial $P(t)$ com base no parâmetro α ;
4: **Construa e avalie** $P(t)$ usando o parâmetro CG ;
5: GA($P(t), CG, nPop$);
6: {Fase 2 do algoritmo}
7: **para** iter $\leftarrow 1$ **até** N_{iter} **faça**
8: **para** q $\leftarrow 1$ **até** 4 **faça**
9: **Gere** $P(t)$ com $nPop$ indivíduos, perturbando-se o quadrante q de S^* com base no parâmetro α ;
10: **Desconstrua e reconstrua** o quadrante q de cada indivíduo em $P(t)$ com base no parâmetro CG ;
11: GA($P(t), CG, nPop$);
12: **fim para**
13: **fim para**
14: **Retorne** S^* ;

Algoritmo 2 GA($P(t)$, CG, $nPop$)

1: **enquanto** (*P(t) não convergiu*) **faça** {Loop principal. Controla o número de gerações }
2: $t = t + 1$;
3: **para** i $\leftarrow 1$ **até** $nPop$ **faça** {Gera descendentes através de Recombinação}
4: **Selecione** aleatoriamente pais p_1 e p_2 de $P(t)$;
5: $d \leftarrow$ cruzamento(p_1, p_2);
6: **Construa e avalie** o descendente d usando CG e respeitando as facilidades fixadas ;
7: **fim para**
8: **Selecione** indivíduos sobreviventes em $P(t)$;
9: **fim enquanto**

Após a convergência de $P(t)$, inicia-se a busca local através do procedimento de desconstrução e reconstrução dos quadrantes de S^*, executado pelos loops das linhas 7 e 8. Inicialmente, S^* é perturbada apenas no quadrante $q = 1$, mantendo-se as facilidades localizadas nos demais quadrantes fixas. Apenas as facilidades localizadas em $q = 1$ serão perturbadas no cromossomo (Seção 3.1). Este processo se repete até obtermos $nPop$ vizinhos de S^*. Caso seja encontrado algum indivíduo com solução melhor, S^* é atualizada.

Dando sequência ao processo evolutivo, o algoritmo genético é novamente chamado na linha 11, filhos desta nova população são gerados e a população irá evoluir até sua convergência. Observe que o número de vezes que cada quadrante será perturbado, bem como o GA será executado, dependerá do parâmetro N_{iter}. Portanto, um valor alto para N_{iter} pode afetar o desempenho do algoritmo.

7 Experimentos Computacionais

No intuito de avaliar a performance de nossa abordagem utilizando algoritmo genético, foram realizados diversos experimentos computacionais. Os algoritmos foram codificados em C++ e os experimentos conduzidos em uma máquina Intel Core $i7 - 3770$, com $3,4$ GHz e $11,7$ GB de memória RAM em um sistema operacional Linux. A performance do algoritmo foi testada em 8 instâncias da literatura do caso irrestrito e cada solução comparada com outras abordagens utilizadas para resolver o mesmo problema. Para que pudéssemos identificar os valores dos parâmetros que levassem a uma boa calibragem do algoritmo, foram testados os seguintes valores:

- **Tamanho da população**($nPop$): foram utilizadas populações com $150, 200, 300$ e 500 indivíduos;
- **População inicial**(α): para o parâmetro α descrito na Seção 3.1, foram utilizados os valores $5, 10$ e ∞;
- **Critério Guloso**(CG): critério de escolha do melhor local para inserção. Foram utilizadas duas funções distintas, uma baseada no produto do fluxo pela distância (11), e a outra uma soma ponderada de (11) com o produto da distância de i até a origem d_{iO} e a soma dos fluxo entre i e as m facilidades já alocadas (12). Seja M o conjunto das facilidades já alocadas e i a facilidade sendo inserida, então

$$C_i = \sum_{j \in M} f_{ij} d_{ij} \tag{11}$$

$$DC_i = \left(\frac{n - |M| - 1}{n - 1}\right) d_{iO} \sum_{j \in M} f_{ij} + \left(1 - \left(\frac{n - |M| - 1}{n - 1}\right)\right) \sum_{j \in M} f_{ij} d_{ij} \tag{12}$$

Desta forma, combinando os 4 possíveis valores de $nPop$, com os 3 possíveis valores de α mais 2 possíveis valores de CG, foram geradas 24 versões a serem analisadas. Cada versão rodou 10 vezes, encerrando após cada um dos quatro quadrantes ter sido desconstruído e reconstruído duas vezes ($N_{iter} = 2$, obtido após alguns testes). A versão selecionada foi chamada de Versão Padrão (VP) com os seguintes parâmetros de referência: $nPop = 200$, $\alpha = 5$ e $CG = C$. A Tabela 2 compara os resultados da VP com os melhores resultados da literatura, obtidos de Gonçalves e Resende (2014). A última coluna da tabela (BKS) apresenta os custos das melhores soluções conhecidas, sendo as duas primeiras instâncias obtidas pelo BRKGA e as demais (marcadas com $*$) obtidas por uma das 24 versões testadas. Observe que nossa abordagem conseguiu superar a melhor solução da literatura, com um tempo de processamento menor ou próximo, para as instâncias L050, L062, L075, L100 e L125A.

Instância	**VIP-PLANOPT**		**BRKGA**		**GADR (nPop=200; α=5; CG=C)**					**BKS**
	C(mín)	T(s)	C(mín)	T(s)	C(mín)	C(méd)	C(máx)	T*(s)	TT(s)	
L020	1157	0,3	1130,00	0,48	1151,00	1172,15	1185,00	0,27	0,38	**1130,00**
L028	6447,25	1,5	6014,07	0.95	6131,75	6293,99	6604,77	0,64	0,68	**6014,04**
L050	78244,68	7	69404,64	6,26	**69221,38**	69566,11	69835,55	15,04	15,40	**68903,33***
L062	3939362	4996	3685136,02	9,07	**3678026,00**	3696483,55	3720726,00	11,60	12,02	**3657352,00***
L075	34396,38	13	31482,84	11,60	**30752,79**	**31162,37**	31664,14	15,87	16,58	**30574,54***
L100	538193,10	14	478910,09	57,03	**472157,67**	**474233,62**	475609,29	27,07	28,25	**470722,32***
L125A	288774,60	110	256860,78	83,58	**249456,31**	**251487,93**	253391,59	84,32	87,16	**247412,38***
L125B	1084451	70	943140.06	118,65	944565,60	946879,20	951385,32	64,24	66,37	**937516,83***
Gap(%)	11,26		1,33		**0,91**	1,97	3,28			
Gap-LG(%)	13,41		1,77		**0,58**	1,22	1,93			
TM(s)		651,48		35,95				**27,38**	**28,36**	

Tabela 2: Comparação entre VP e os melhores resultados da literatura para as instâncias testadas.

As três últimas linhas da Tabela 2 apresentam, os Gaps médios entre os custos obtidos por cada abordagem e o BKS (Gap(%)), os mesmos Gaps médios excluindo as instâncias com $n < 50$ (Gap-LG(%)) e os tempos médios em segundos TM, para todas as instâncias testadas. Observe que ambos os Gaps em negrito referentes ao custo mínimo C(mín), dados respectivamente por $0,91\%$ e $0,58\%$, bem como o tempo médio da melhor solução T$^* = 27,38$s e o tempo total médio TT=$28,36$s, são menores que as demais abordagens. Vale ressaltar que Gonçalves e Resende (2014) fornecem apenas os custos mínimos obtidos pelo método BRKGA.

7.1 Análise dos Resultados

Na Tabela 3, são analisados os Gaps(%) médios entre os custos mínimos C(mín) e os BKSs (fornecidos na Tabela 2) de cada instância com $n \geq 50$ facilidades, bem como seus tempos totais médios em segundos TTM, para as 24 versões geradas. Observe que a versão padrão possui um Gap(%)=$0,58$ menor que todas as versões com $nPop = 150$ e $nPop = 200$, empatando apenas com a versão $nPop = 200; \alpha = 10; CG = DC$. As versões com população superior a 200 indivíduos possuem Gaps menores ou iguais a $0,58$, indicando soluções de melhor qualidade, mas com um tempo de processamento médio bem superior que os $28,36$ segundos apresentado pela versão padrão. Desta forma, a versão selecionada parece apresentar um equilíbrio entre qualidade e tempo de processamento das soluções.

	Tamanho da população							
	150		200		300		500	
	Gap(%)	TTM(s)	Gap(%)	TTM(s)	Gap(%)	TTM(s)	Gap(%)	TTM(s)
α=∞; CG=C	0,75	45,57	0,64	65,53	0,41	141,51	0,25	298,75
α=5; CG=C	0,95	17,43	**0,58**	**28,36**	0,53	61,03	0,28	126,03
α=10; CG=C	0,64	32,81	0,64	45,58	0,53	100,53	0,21	198,05
α=∞; CG=DC	0,84	55,17	0,67	75,57	0,47	158,06	0,24	311,50
α=5; CG=DC	0,90	21,96	0,77	42,49	0,58	61,64	0,05	125,81
α=10; CG=DC	0,70	46,91	0,58	40,34	0,44	93,69	0,31	208,07

Tabela 3: Comparação entre os Gaps(%) e os tempos totais médios TTM(s) das versões testadas.

Ao resolver uma instância do PLFAD, o algoritmo proposto o faz em duas fases distintas. A fase 1, que antecede a etapa de desconstrução e reconstrução e é executada até a convergência da população e a fase 2, que será executada até que o algoritmo complete o procedimento de desconstrução e reconstrução nos quadrantes.

Instância	**Fase 1**			**Fase 2**			**Gap(méd)**	**Gap(mín)**	**TTM**
	C(méd)	C(mín)	TTM(s)	C(méd)	C(mín)	TTM(s)	**%**	**%**	**%**
L020	1178,8	1167,00	0,18	1172,15	1151,00	0,38	-0,56	-1,37	111,11
L028	6513,77	6188,29	0,52	6293,99	6131,75	0,68	-3,37	-0,91	30,77
L050	69826,74	69505,71	11,47	69566,11	69221,38	15,40	-0,37	-0,41	34,26
L062	3738541,85	3718150,50	8,15	3696483,55	3678026,00	12,02	-1,12	-1,08	47,48
L075	32043,81	31290,96	12,27	31162,37	30752,79	16,58	-2,75	-1,72	35,13
L100	476363,59	474794,94	21,43	474233,62	472157,67	28,25	-0,45	-0,56	31,82
L125A	255108,15	252776,86	57,82	251487,93	249456,31	87,16	-1,42	-1,31	50,74
L125B	953793,29	947527,42	44,93	946879,20	944565,60	66,37	-0,72	-0,31	47,72

Tabela 4: Análise do impacto da fase 2 sobre a fase 1 na versão padrão.

Deste modo, procurou-se medir o impacto da fase 2 sobre a fase 1 da versão padrão, através da qualidade da solução obtida após a busca de desconstrução e reconstrução que é executada na fase 2, sobre a solução obtida ao final da fase 1, bem como o tempo necessário para alcançar este objetivo. A qualidade das soluções obtidas na fase 2 ficam evidenciadas, uma vez que tanto o Gap(mín), quanto o Gap(méd), da fase 2 com relação à fase 1, indicam soluções com custos inferiores para todas as instâncias. Além disso, o aumento do tempo de processamento para as instâncias com $n \geq 50$ não é muito superior a 50%. Isto revela que o custo computacional de aplicar a fase 2, é compensado pela qualidade das soluções encontrada pelo algoritmo.

8 Conclusões e Perspectivas de Pesquisa

Neste artigo, foi apresentado um algoritmo genético baseado em uma estratégia gulosa de localização das facilidades, para resolver instâncias de referência da literatura do problema de

layout de facilidades com áreas diferentes. Além disso, foi utilizada uma estratégia de busca de desconstrução e reconstrução em uma parcela da melhor solução para tentar escapar de ótimos locais. A abordagem proposta foi capaz de produzir boas soluções, superando a melhor solução conhecida da literatura (BKS) em 5 das 8 instâncias testadas, em um tempo de processamento competitivo. Devido às características da estratégia de localização, onde a inserção das facilidades é feita nos quadrantes, a abordagem proposta parece ser uma boa alternativa para a resolução de instâncias com $n \geq 50$, algo difícil de ser obtido atualmente através de métodos exatos.

Referências

Cohoon, J. P., Hegde, S. U., Martin, W. N. e Richards, D. S. (1991), Distributed genetic algorithms for the floorplan design problem. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, v. 10, n. 4, p. 483–492.

Drira, A., Pierreval, H. e Hajri-Gabouj, S. (2007), Facility layout problems: A survey. *Annual Reviews in Control*, v. 31, n. 2, p. 255–267.

Dunker, T., Radons, G. e Westkämper, E. (2003), A coevolutionary algorithm for a facility layout problem. *International Journal of Production Research*, v. 41, n. 15, p. 3479–3500.

Gonçalves, J. F. e Resende, M. G. *A biased random-key genetic algorithm for the unequal area facility layout problem*, 2014.

Heragu, S. *Facilities design*. PWS Publishing, Boston, MA, 1997.

Imam, M. H. e Mir, M. (1998), Cluster boundary search algorithm for building-block layout optimization. *Advances in Engineering Software*, v. 29, n. 2, p. 165–173.

Kado, K. *An investigation of genetic algorithms for facility layout problems*. Tese de doutorado, University of Edinburgh, 1996.

Kulturel-Konak, S. e Konak, A. (2011), A new relaxed flexible bay structure representation and particle swarm optimization for the unequal area facility layout problem. *Engineering Optimization*, v. 43, n. 12, p. 1263–1287.

Lai, K. e Chan, J. W. (1997), Developing a simulated annealing algorithm for the cutting stock problem. *Computers & Industrial Engineering*, v. 32, n. 1, p. 115 – 127.

McKendall, A. R. e Hakobyan, A. (2010), Heuristics for the dynamic facility layout problem with unequal-area departments. *European Journal of Operational Research*, v. 201, n. 1, p. 171–182.

Sahni, S. e Gonzalez, T. (1976), P-complete approximation problems. *Journal of the ACM (JACM)*, v. 23, n. 3, p. 555–565.

Scholz, D., Petrick, A. e Domschke, W. (2009), Stats: a slicing tree and tabu search based heuristic for the unequal area facility layout problem. *European Journal of Operational Research*, v. 197, n. 1, p. 166–178.

Syswerda, G. e Palmucci, J. The application of genetic algorithms to resource scheduling. *ICGA*, p. 502–508, 1991.

Taillard, É. D. e Voss, S. Popmusic–partial optimization metaheuristic under special intensification conditions. *Essays and surveys in metaheuristics*, p. 613–629. Springer, 2002.

Tam, K. Y. (1992), Genetic algorithms, function optimization, and facility layout design. *European Journal of Operational Research*, v. 63, n. 2, p. 322–346.