**Universidade de Santa Cruz do Sul**

**Departamento de Informática**

**Ciência da Computação**

**Programa Institucional de Bolsas de Iniciação Científica – PIBIC**

**Projeto:**

“**completar**”

Relatório de atividades do bolsista

Bolsista: Leonardo Pellegrini Silva

Orientador: João Carlos Furtado

**Santa Cruz do Sul, Janeiro de 2018**

**ÍNDICE**

**1. Dados de identificação**

**2. Introdução**

**3. Revisão de literatura**

**4. Atividades Desenvolvidas e não Desenvolvidas**

**5. Metodologia**

**6. Resultados Obtidos**

**7. Discussões/Conclusões**

**8. Referências**

**9. Auto-Avaliação do Bolsista**

**10. Avaliação do Bolsista pelo Orientador**

**11. Anexos**

**12. Assinaturas**

**1. Dados de Identificação**

1.1 Do Coordenador

a. Nome: João Carlos Furtado

b. Unidade Acadêmica: UNISC – Universidade de Santa Cruz do Sul, Departamento de Informática

1.2 Do Bolsista

a. Nome: Leonardo Pellegrini Silva

b. Período de vigência da bolsa: De 01/10/2017 a 31/07/2018

c. Curso: Ciência da Computação

1.3 Do Projeto

a. Nome : **completar**

b. Período de vigência do projeto: **completar**

c. Área do conhecimento (UNISC): **completar**

d. Linha de Pesquisa: **completar**

e. Data do Relatório: Janeiro de 2018.

**2. Introdução**

O projeto tem como objetivo a otimização de resultados do Problema de *layout* de instalações em linha única (*Single Row Facility Layout Problem*  ou  *SRFLP*). O Problema é considerado NP-difícil [10], foi inicialmente proposto por Simmons [36] e consiste no seguinte: Considere um conjunto de instalações retangulares que variam apenas em seus comprimentos. Cada instalação se comunica com todas as outras instalações, e o custo desta comunicação é o produto da intensidade de transmissão entre um par de instalações e a distância entre elas.

A intensidade de transmissão pode ser visualizada como o número de vezes que o par de instalações precisa se comunicar, e a distância entre este par é medido como a distância entre seus centróides. O custo de comunicação entre um par de instalações é alto se o par possui uma alta intensidade de transmissão, em outras palavras, se o par se comunica frequentemente, ou se os centros das instalações estejam localizados um longe do outro. O custo total de tansmissão é a soma do custo de transmissão entre todos os pares de instalações.

O *SRFLP* já foi utilizado para modelar inúmeras aplicações práticas, como o arranjamento de quartos em hospitais, departamentos em uma empresa ou supermercados [36], designação de arquivos em discos cilíndricos no armazenamento de um computador ou *layouts* de armazéns.

**3. Revisão de Literatura**

3.1. Definição Formal

O *SRFLP* é formalmente definido como:

Dado: Um Conjunto *F* = {1, 2, …, *n*} de *n* > 2 instalações, onde uma instalação *j* possui comprimento, e intensidades de transmissão para cada par (*i, j*) de instalações, *, i ≠ j*.

Objetivo: Encontrar a permutação Π =de instalações em *F* que minimize a expressão

onde seja a distância entre os centróides das instalaçõese quando as instalações em *F* estão ordenadas pela permutação Π.

3.2. Hierarquia de Problemas

O *SRFLP* pode ser colocado em uma hierarquia de problemas lidados na literatura, alguns destes membros são casos especiais de outros membros.

Um exemplo é o *Problema de Alocação Espacial* (*Space Allocation Problem* ou *SAP*)*,* onde seu objetivo é o de dar forma e orientar instalações em uma planta para alcançar uma dada medida de eficiência. Este não é estritamente um problema de localização de instalação, porque em um problema de localização de instalação existe um conjunto de localizações pré-definido, em que tais localizações são separadas por uma distância fixa e somente é necessário determinar quais instalações serão localizadas em determinadas localizações. No *SAP* as localizações não são definidas.

O *SRFLP* é um caso especial do *SAP* onde as instalações possuem comprimentos fixos e serão arranjanar de uma maneira linear. É um problema de ordenação com instalações de comprimento não-uniforme arranjadas em uma única linha. Este problema também é conhecido como o Problema de Alocação Espacial Unidimensional (*One Dimensional Space Allocation Problem* ou *ODSAP*).

O *Problema de* *Arranjo Mínimo Linear* *(MinLa*) é um caso especial do *SRFLP* onde os comprimentos de todas as instalações são unidades e o custo de cada par de instalações é. Para uma leitura mais profunda do *MinLa*, podem ser referênciados Amaral [3], Días e outros [13] e Petit [30].

O *Problema de Ordenação Linear Generalizada* (*Generalized Linear Ordering Problem* ou *GLOP*) é muito relacionado ao *SRFLP,* onde são dadas *n* instalações, *n* localizações ao longo de uma linha, e a transmissão intensa entre cada par de instalações, e o objetivo é encontrar relações de um-a-um entre *n* instalações para *n* localizações para minimar o custo total de transmissão.

3.3. Métodos de Solução Exatos

Métodos de solução exatos para o *SRFLP* possuem altos gastos computacionais e altos requisitos de memória. Nesta seção serão descritos altuns destes métodos utilizados na literatura.

3.3.1. *Branch and Bound* Combinacional

Técnicas *Branch and Bound* (*BB*)são muito caras computacionalmente quando aplicadas em instâncias pequenas do *SRFLP*, então não foi realizada muita pesquisa em obter uma técnica *BB* para instâncias maiores que 11.

Este algoritmo é categorizado por sua construção passo-a-passo na ordenação das instalações, já que uma permutação completa das instalações não é possível até os estágios mais avançados do algoritmo. Por este motívo esta técnia é conhecida como *Branch and Build Up*.

3.3.2. Programação Matemática

Love e Wong [27] formularam o *SRFLP* como um programa linear de inteiros mistos e o resolveram usando o código *MIP* da *IBM.* O código era capaz de resolver instâncias *SRFLP* de 10 instalações ou menos. Para instâncias maiores, severas execuções eram necessárias para a obtenção da solução, acompanhado de uma heirística de julgamento para fixação de posições de algumas instalações.

Uma nova formulação de programação linear de inteiros mistos do *SRFLP* foi proposta por Amaral [1], resolvendo instâncias de até 15 instalações de [18, 27, 36] com mais eficiência do que estudos reportados na literatura. Outra formulação de programação linear de inteiros misto foi proposta por Amaral [2]. Esta formulação foi obtiva através de uma série de transformações matemáticas em um modelo de programação quadrática 0-1 apresentadas no mesmo *paper*.

Códigos baseados nesta formulação foram ainda mais eficientes que aqueles baseados em antigas programações lineares de inteiros mistos, podendo resolver instâncias do *SRFLP* com até 18 instalações que foram propostas em [18, 27, 36] em um tempo razoável.

3.3.3. Programação Dinâmica

Picard e Queyranne [31] reportaram um algoritmo de programação dinâmica para resolver o *SRFLP*. Embora o algoritmo tenha sido aplicado com sucesso em instâncias de até 15 instalações, o alto requerimento de memória desta técnica desestimularam o seu uso para a comunidade de pesquisa.

3.3.4. *Branch and Cut*

Amaral e Letchford [5] conquistaram o primeiro estudo poliédrico profundo do *SRFLP,* e introduziram um *Polytope* chamado de *Polytope* de distância definido como

que é um casco convexo de *n*!/2vetores de distância *d* válidos. A distância estre os centróides das instalações *i* e *j* no *layout* foram denotadas pore o comprimento do vetoronde *n* é o tamanho da instância.

Um algoritmo de *Branch and Cut* utiliza uma combinação de regras de ramificação e *Cutting Planes* para reduzir a região factível de um problema de programação matemático. Amaral e Letchford [5] sugeriram uma regra de ramificação especializada para evitar o uso de variáveis binparias adicionais e usaram a heurística primal baseada na escalação multi-dimensional para obter permutações factíveis para o *SRFLP*. Este método resolveu instâncias de até 30 instalações tiradas de [8, 17, 36] com tempos de execução razoáveis.

3.3.5. Aproximação de *Cutting Planes*

Amaral [a4] sugeriu uma programação linear baseada na abordagem de *Cutting Planes* para o *SRFLP*. Foi encontrado um novo limite inferior usando esta abordagem de *Cutting Planes* que se igualou ao custo de uma solução ótima em cada uma das instâncias do *SRFLP* com até 35 instalações. Esta aproximação de Amaral [4] requeriu tempos significativamente menores que o de Anjos e Vanelli [8], que resolveram instâncias com até 30 instalações.

3.3.6. Programação semidefinida

Programação semidefinida (*SDP*) se refere a classe de otimização de problemas onde a função linear de uma matriz variavél *X* é otimizada sujeita a restrições lineares nos elementos de *X* e uma restrição adicional que *X* seja positivo semidefinida. O relaxamento do *SDP* foi apresentado por Anjos e outros [7] e foi melhorado por Anjos e Vannelli [8] usando desigualdades triangulares como *Cutting Planes*. Instâncias com até 30 instalações foram retiradas de [17, 28, 32, 36] e suas soluções globais ótimas foram obtidas.

Este relaxamento possuia restrições lineares de *O(n3)* e requeria modelos grandes para serem resolvidos. Para poder superar a dificuldade do grande tempo computacional requerido, um relaxamento alternativo do *SDP* com restrições lineares de *O(n2)* foi apresentado por Anjos e Yen [9]. Este relaxamento alternativo conseguia resolver problemas com até 100 instalaçõescom o custo de uma deteriozação marginal das soluções obtidas por Anjos e outros [7].

Um estudo recente por Hungerlänger e Rendl [20] melhorou ainda mais o relaxamento do *SDP* de Anjos e outros [7] e Anjos e Yen [9]. Foi utilizada uma combinação de métodos de otimização para lidar com relaxamentos mais fortes, porém, mais caros. Este método foi construído na técnica sugradiente de otimização e lidou com cortes de desigualdade triangular e os cortes *LS* através da dualidade de Lagrange. Quando comparado com o estudo de Anjos e Yen [9], o estudo de Hungerlänger e Rendl [20] reduziu os vãos conhecidos entre os melhores limites inferiores e as melhores permutações para as instâncias do *SRFLP* com 60 a 100 instalações.

3.4. Métodos de Solução Heurísticos

Visto que algoritmos exatos para o *SRFLP* são computacionalmente caros, eles são aplicados em instâncias relativamente pequenas, com até 42 instalações. Pesquisadores utilizam heurísticas para resolver instâncias maiores do *SRFLP*. Estas são muito mais rápidas que algoritmos exatos, mas não garantem soluções ótimas. Serão revisadas nesta seção as principais heurísticas utilizadas na literatura, sendo estas divididas entre heurísticas de construção e heurísticas de melhoria.

3.4.1. Heurísticas de Contrução

Uma heurística de contrução quando aplicada no *SRFLP* progessivamente contrói uma sequência de instalações até que a permutação completa seja adquirida. Usualmente uma instalação é fixada em cada iteração da heurística, mas, em algumas heurísticas de contrução para o *SRFLP*, múltiplas instalações são designinadas para determinadas localizações em uma única iteração da heurística.

Heragu e Kusiak [18] foram os primeiros a apresentar uma heurística de contrução para o *SRFLP*. Em cada iteração, a heurística modificava uma matriz de fluxo ajustada que era computada baseada na matriz de transmissão e a solução era contruída desse modo, um par de instalações é selecionado, e então este era conectado como parte da solução. Esta heurística foi aplicada em nove instâncias com até 20 instalações [28] e soluções ótimas foram encontradas em seis dessas instâncias.

Ravi Kumar e outros [32] apresentaram outra heurística gulosa. Esta heurística ignora os comprimentos das instalações, e tenta designar instalações com a maior intensidade de transmissão entre instalações para localizações adjacentes na solução.Ela difere de outras heurísticas da literatura por permitir a designação de mair de uma instalação para uma sequência existente de instações em qualquer iteração na heurística. Ela foi testada em instâncias com até 30 instalações de [16, 17, 28, 32, 36] e obteve soluções melhores que as soluções conhecidas naquele tempo em um tempo significativamente menor.

Djellab e Gourgand [14] proporam uma heurística de contrução de dois passos baseada em inserção. Esta heurística foi aplicada para instâncias de até 30 instalações de [16, 32]. Foi encontrado que todas as instâncias, exceto uma, obtiveram ou o melhor resultado na literatura, ou superaram todos os outros métodos disponíveis na literatura até 2001.

3.4.2. Heurísticas de melhoria

Heurísticas de melhoria começam com uma ou mais permutações de instalações ou iterativamente melhoram a solução inicial de instalações até que a solução não possa ser melhorada ou um critério de pausa é alcançado. A classificação de heurísticas de melhoria é baseada se ela funciona para melhorar uma única solução a cada iteração (heuríticas baseadas em uma única solução) ou se ela usa uma população de soluções e as combina para gerar nova soluções a cada iteração (heurísticas baseadas em população).

No geral, as soluções obtidas usando heurísticas de melhoria não garantem uma solução ótima, ou até mesmo um limite na qualidade da solução. No caso do *SRFLP*, heurísticas como a Busca Tabu ou o *Simulated Annealing*, que são normalmente implementadas como heurísticas baseadas em uma única solução têm sido também implementadas como heurísticas baseadas em população, com a diferença de que em cada iteração, apenas uma das permutações é escolhida e mudada pela heurística. Nesta seção serão revisadas as heurísticas de melhoria sugeridas na literatura para resolver o *SRFLP*.

3.4.2.1. “Algoritmo Heurístico” em Heragu e Kusiak [16]

Heragu e Kusiak [16] apresentaram a primeira abordagem de uma heurística de melhoria para resolver o *SRFLP*. Eles transformaram uma formulação não linear, como pode ser vista em [16], em um modelo sem restrições usando um método de penalidade e o resolveram usando o que eles chamaram de o “Algoritmo Heurístico”.

A solução resultante por esta heurística foi melhorada por uma trocar de pares gulosa. Esta abordagem foi usada heuristicamente para resolver instâncias do *SRFLP* com até 30 instalações. Soluções ótimas foram obtidas para instâncias de até 10 instalações.

3.4.2.2 Busca Tabu

De Alvarenga e outros [12] apresentaram a primeira implementação da Busca Tabu para instâncias pequenas do *SRFLP*. O algoritmo começava com uma solução inicial aleatória, uma lista tabu vazia e designava o limite inferior e os valores limiares do problema para as melhores soluções encontradas na literatura para determinada instância.

Em cada iteração do algoritmo, era efetuada uma busca na vizinhança 2-opt da solução atual para obter a solução de menor custo cujo par de instalações permutadas não estava na lista tabu ou cujo custo era menor que o valor limiar. Se o custo desta solução fosse menor que o último melhor custo obtivo pelo algoritmo, então esta solução era determinada como a melhor atual. Em seguida, a lista tabu era atualizada com o par de instalações permutadas para obter o vizinho 2-opt, e o menor limite e valor limiar eram designados como a melhor solução se o custo fosse melhor que a solução obtida pelo algoritmo em todas as suas iterações. Isto compede a uma iteração do algoritmo da lista tabu. O algoritmo termina se não se não há melhora no custo da solução para um número de iterações especificado pelo usuário.

Este algoritmo foi aplicado em instâncias do *SRFLP* de até 30 instalações escolhidas de [16, 17, 36]. O algoritmo resultou em soluções ótimas para três instâncias de tamanhos 4, 10 e 11 e superou as melhores soluções previamente conhecidas nas seis instâncias restantes.

3.4.2.3. *Simulated Annealing*

Romero e Sánchez-Flores [33] foram os primeiros a aplicar o *Simulated Annealing* (*SA*)no *SRFLP*. Eles compararam a performance de severas implementações do *SA* nas quais diversas combinações de estrátegias de inserção e troca elementares foram estudads para encontrar um vizinho de uma solução, e reportaram que aquela inserção baseada em vizinhança é melhor do que a troca baseada em vizinhança.

Eles também deselvolveram quatro métodos amplos, dois baseados no *SA* e dois baseados em métodos e melhoria local, e criaram 70 variações destes algoritmos para resolver o *SRFLP*. As soluções finais obtidas por todas as variantes foram aperfeiçoadas usando um método de melhoria local 3-opt chamado 3-LOC. Romero e Sánches-Flores concluíram que métodos de melhorias locais são melhores para problemas pequenos, enquanto o *SA* funciona para instâncias com tamanhos em torno de 40 ou 50.

Outra abordagem do *SA* para resolver o *SRFLP* foi feita por De Alvarenga e outros [12]. Eles propuseram um algoritmo do *SA* baseado na estrutura da vizinhança 2-opt. O algoritmo começa com uma solução inicial aleatória e um valor de temperatura inicial relativamente alto. A cada vez que a condição de equilíbrio é alcançada, a temperatura atual é reduzida por um parâmetro *α* até que a temperatura final seja obtida. O algoritmo foi aplicado em nove instâncias do *SRFLP* com tamanho de até 30 instalações escolhidas de [16, 17, 36]. O algoritmo resultou em soluções ótimas para cada uma das três instâncias de tamanhos 4, 10 e 11 e superou as melhores soluções previamente conhecidas nas seis instâncias restantes.

3.4.2.4. Otimização da Colônia de Formigas

Solimanpur e outros [37] propuseram um algoritmo de formigas para o *SRFLP*. Em cada iteração do algoritmo um novo conjunto de formigas era criado. Cada formiga era um agente que escolhe movimentos. Um movimento designa uma instalação para uma posição. Para cada movimento, uma formiga possui uma lista para armazenar quais instalações a mesma já designou.

Todos os movimentos realizados pela formiga são armazenados na memória da formiga. Esta memória é usada no final de cada iteração para atualizar os níveis de trilha, os quais são análogos aos depósitos de feromônios. Movimentos são realizados durante o processo heurístico. O fator conhecido como a desejabilidade de um movimento é computado para cada movimento. O nível de trilha para cada movimento indica com que frequência aquele movimento em particular foi selecionado por outras formigas no algoritmo. A desejabilidade e o nível de trilha dos movimentos são combinados para computar a probabilidade da seleção de um determinado movimento por uma formiga.

A solução gerada por cada formiga era aperfeiçoada localmente usando a estrutura 2-opt. Após o aperfeiçoamento os níveis de trilha são atualizados, as formigas são destruídas, e a próxima iteração começa. O número total de iterações é definido pelo usuário para controlar o tempo de execução do algoritmo. O número de formigas de cada iteração também é especificado pelo usuário.

Implementações ingênuas deste algoritmo requerem tempo *O(n4)*e são muito caras computacionalmente. Solimanpur e outros [37] sugeriram uma maneira efetiva da implementação do algoritmo em um tempo *O(n3)*. Eles aplicaram o algoritmo em instâncias do *SRFLP* com até 30 instalações obtidas de [10, 28, 32, 36] e observaram que o algoritmo proposto obteve a melhor solução para todas as instâncias. O algoritmo encontrou soluções melhores que Heragu e Kusiak [18] e Ravi Kumar e outros [32].

3.4.2.5. Busca Dispersa

Kumar e outros [25] aplicaram a busca dispersa no *SRFLP.* O algoritmo de busca deles usava: um método de geração de diversificação para gerar diversas soluções; um método de aperfeiçoamento para produzir uma solução melhor usando uma heurística; um conjunto referência contendo soluções boas e diversas; um método de geração de subconjuntos que combina as solulções do conjunto referência formando diferentes tipos de subconjuntos; e um método combinacional que combina as soluções em cada subconjunto para gerar novas soluções.

Em [25] a geração de diversificação e o método de geração de subconjuntos foram adotados de [15] e a heurística de inserção esquerda e direita foi utilizada para resolver o primeiro conjunto de problemas também solucionados em [37] e obtiveram os mesmos resultados que em [37], exceto quando o tamanho do problema era 30, então Kumar e outros [25] apresentaram um custo menor que o custo em [37].

3.4.2.6. Otimização do Enxame de Partículas

Samarghandi e outros [35] aplicaram a otimização do enxame de partículas no *SRFLP*. Eles usaram uma encodificação/decodificaçõ fatorial para mapear o espaço facível discreto do *SRFLP* em um espaço contínuo. O sistema numérico fatorial[25] foi usado para estabelecer um mapeamento de um-a-um entre o conjunto de todas as permutações das instalações do *SRFLP* e o conjunto de todas os númerosfatoriais e então o número decimal correspondente para cada permutação foi encontrado, assim associando um número único para cada permutação no algoritmo de otimização de enxame de partículas.

Eles então usaram o mapeamento inverso para recuperar a requerida permutação correspondente a melhor solução. No começo do algoritmo, partículas que, na implementação deles números que correspondem as permutações, são geradas aleatóriamente do intervalo [0, *n*! - 1] para que a diversificação no processo de busca seja assegurada. Velocidades das partículas inicialmente aleatoriamente geradas também são geradas aleatoriamente em um intervalo apropriado para se assegurar de que as partículas não saiam da região facível.

A cada iteração, novas posições das partículas são computadas baseadas na posição das mesmas, e as partículas são então movidas para suas novas posições. A solução atual e a melhor solução obtida são então atualizadas. Cada partícula do algoritmo passa por um processo de intensificação após um número fixo de iterações. Este processo de intensificação é realizado aplicando uma busca local utilizando a estrutura da vizinhança 2-opt. Este algoritmo termina após uma condição especificada pelo usuário e a melhor solução encontrada pelo algoritmo será exibida.

Samarghandi e outros [35] aplicaram este algoritmo nos conjuntos de problemas usandos em [37] e obtiveram as mesmas soluções que essas reportadas em [37]. Eles também usaram o algoritmo para resolver instâncias do *SRFLP* de [1, 8] com até 30 instalações nas quais heurísticas nunca haviam sido aplicadas antes. O algoritmo obteve soluções ótimas para oito das doze instâncias.

3.4.2.7. Algoritmo Genético

A única aplicação de algoritmos genéticos para resolver o *SRFLP* foi feita por Datta e outros [11]. Neste algoritmo, solução na população são representadas como a permutação das instalações. A população inicial é inicializda de quatro maneiras para garantir a diversificação da população. O operador de seleção uma seleção de torneio binário para formar uma piscina de acasalamento. Após a criação da piscina, o operador de *Crossover* seleciona duas soluções pais da piscina de acasalamento.

A nova solução (criança) é gerada da seguinte maneira: Inicialmente nenhuma das instalações da criança são alocadas para posições particulares. Um pai é escolhido aleatoriamente. Uma instalação é escolhida aleatoriamente desse pai. O algoritmo tenta designar esta instalação na criança na mesma posição que é encontrada no pai. Se a posição já tiver sido ocupada por outra instalação, esta então é guardada na piscina de reserva. Quando todas as instalações que não estiverem na piscina de reserva forem designadas para a criança, as instalações na piscina de reserva são então designadas aleatoriamente para as posições restantes. Esta operação de *Crossover* é usada para gerar o mesmo número de crianças que número de soluções na piscina de acasalamento. Cada criança é então modificada usando um operador de mutação. Os dois conjuntos de soluções, a piscina de acasalamento e o conjunto de crianças são então unidos, e um operador de seleção de elite escolhe os melhores 50% destas soluções para formar a próxima geração. O algoritmo encerra quando um número de gerações especificado pelo usuário for produzido, e a melhor solução encontrada pelo algoritmo é exibida.

Datta e outros [11] usaram o algoritmo genético para resolver 14 instâncias de [4, 7] com até 30 instalações cujas soluções ótimas eram conhecidas, e 20 instâncias de [7] de 60 a 80 instalações cujas soluções ótimas não eram conhecidas. Os primeiros 14 problemas foram resolvidos de maneira ótima e o algoritmo melhorou o melhor custo para nove dos próximos 20 problemas.

**4. Atividades Desenvolvidas e não Desenvolvidas**

Atividades Desenvolvidas:

Foram desenvolvidos alguns métodos de obtenção de soluções para o *SRFLP*, assim como heurísticas de melhoria, estes métodos serão descritos detalhadamente na próxima seção. Também foi estudada a literatura e métodos clássicos de obtenção de soluções e heurísticas construtivas e de melhoria.

O programa criado foi testado utilizando as instâncias de tamanho 60, 70, 75 e 80, criadas por Anjos [7], e as instâncias de tamanho 42, 49, 56, 64, 72, 81 e 100,criadas por Anjos e Yen [9]. Algumas das soluções da literatura foram alcançadas pelo programa em poucas instâncias. Os resultados serão discutidos em detalhes na 6a seção.

Atividades não desenvolvidas:

Apesar do programa já ter alcançado algumas das soluções da literatura, o mesmo não está perto de ser concluído, pois melhorias ainda podem ser efetuadas tanto na geração da solução inicial, quanto nas heurísticas de otimização.

**5. Metodologia**

Foram desenvolvidos quatro métodos principais de obtenção da solução inicial, além de três heurísticas de otimização, sendo que uma destas heurísticas foi descontinuada pois outra oferece uma lógica similar e resultados muito melhores que os resultados desta heurística. Nesta seção serão descritos os métodos utilizados, seus funcionamentos e análise do que já foi feito e do que pode ser melhorado.

5.1. Geradores de Solução Inicial

5.1.1. *Default PathFinder* (*DPF*)

Neste primeiro algoritmo, *n* instalações com o maior fluxo entre si são encontradas e inseridas sequencialmente na solução, sendo *n* um parâmetro que pode ser definido pelo usuário. Este algoritmo pode ser melhorado para que os pares de instalações sejam inseridos da melhor maneira possível, não apenas sequencialmente.

5.1.2. *MaxFlow PathFinder* (*MPF*)

Neste algoritmo, um fluxo total é utilizado para realizar a inserção das instalações na solução, tal fluxo geral é a soma do fluxo de uma instalação com todas as outras instalações. Deste modo, cada instalação possui um fluxo total, e então, as instalações são ordenadas de maneira crescente por este mesmo fluxo. Na próxima etapa, as instalações com menor fluxo são inseridas nas extremidades da solução, de modo que as instalações de maior fluxo total estarão localizadas no centro da sequência de instalações.

Este algoritmo pode ser melhorado verificando se as instalações adjacentes possuem o melhor fluxo de sua vizinhança, sendo o tamanho desta vizinhança um parâmetro definido pelo tamanho da instância. Se não possuírem, estas serão permutadas para que a melhor sequência seja encontrada.

5.1.3. *Relation PathFinder* (*RPF*)

Neste algoritmo, a criação da solução inicial é feita da seguinte forma: um par de instalações com o melhor fluxo entre si é escolhido e inserido na solução, então, uma próxima instalação é selecionada, se a mesma possuir o melhor fluxo com uma das instalações já escolhidas, ela será anexada com esta instalação e inseridas na solução. Se esta possuir o melhor fluxo com uma instalação que não esteja na solução, ambas serão unidas e anexadas da melhor maneira nas instalações das extremidades da solução, e assim sucessivamente, até que a solução esteja completa.

Este algoritmo pode ainda ser melhorado se a inserção de instalações também ocorrer entre instalações já definidas na solução, em outras palavras, a cada nova instalação selecionada, permitir que a mesma possa ser inserida entre duas instalações já posicionadas na solução inicial.

5.1.4. *MaxFlow Relation PathFinder* (*MRPF*)

Este algoritmo, dado o seu nome, mistura conceitos do *MaxFlow PathFinder* e do *Relation PathFinder.* Ele começa estabalecendo um fluxo total para cada instalação, como visto anteriormente no *MPF*, e insere as instalações, sucessivamente, do maior para o menor fluxo total, em grupos de tamanho *n*, sendo *n* um parâmetro informado pelo usuário, e então realiza a soma dos fluxos totais e define este novo valor como o fluxo máximo do grupo.

Conforme cada grupo é formado, estes são inseridos em uma lista de grupos de maneira crescente, em outras palavras, os grupos de menor fluxo máximo estão no começo desta lista, e, os grupos de maior fluxo máximo, estarão no final. Após esta etapa de formação, as instalações são então permutadas para que a melhor sequência das mesmas seja encontrada.

Este método executa diversas operações, sendo que estas foram testadas com diversos valores para *n*, no momento não creio que ela possa ser melhorada além do que já foi melhorado.

5.2. Heurísticas

5.2.1. *Heuristic 1*

Esta primeira heurística divide a solução em grupos de *n* elementos, sendo *n* um valor definido pelo usuário, e, sucessivamente, os permuta baseado no melhor fluxo local, sendo este a ordenação atual, utilizando uma ordenação inversa do primeiro ou do segundo grupo ou trocando ambos os grupos de posição. Este processo é repetido dobrando-se o valor de *n* para cada iteração até que o valor de *n* seja maior que a metade do tamanho da instância.

Esta heurística pode ser melhorada caso sua lógica de permutação de grupos seja baseada no resultado da Função Objetiva dos mesmos, e não apenas no melhor fluxo local. Mas, mesmo que esta mudança seja efetuada, não creio que as soluções produzidas terão a mesma qualidade das soluções produzidas utilizando a *Heuristic 3*.

5.2.2. *Heuristic 2*

Esta heurística foi a mais utilizada em grande parte do projeto, servindo agora apenas como um complemento para a *Heuristic* *3*, pois ela possui um alto gasto computacional e temporal, utilizando soluções não otimizadas, a mesma demora muito mais tempo do que a *Heuristic 3* para obter resultados piores.

Sobre a lógica da mesma, esta percorre a solução tentando apenas inserir cada instalação em todos os lugares possíveis e procura pela melhor solução baseada no resultado da Função Objetiva, caso uma melhora possa ser feita, a próxima instalação a ser testada é a primeira da solução, e não próxima instalação em relação a instalação testada atualmente. Caso uma melhora não seja possível, a próxima instalação em relação a atual será testada, e assim consecutivamente até que nenhuma melhora possa ser feita. Esta heurística permite que uma instalação que já tenha sido movida possa ser testada novamente, de maneira que a melhor solução possa ser encontrada.

Esta heurística pode ser melhorada se for mesclada com a *Heuristic 3*, pois enquanto esta apenas tenta inserir uma instalação entre outras duas intalações, ou antes do início da solução, ou após o final da solução, a outra apenas troca o posicionamento entre instalações.

5.2.3. *Heuristic 3*

Esta última heurística irá, sucessivamente, verificando se a troca de uma instalação com todas as outras instalações da solução poderá implicar em uma melhora do resultado baseado na Função objetiva, caso esta melhora exista, a troca é efetuada e a próxima instalação a ser testada será a primeira da solução, este processo é similar ao processo da *Heuristic 2.*

Tanto esta heurística quanto a anterior são as heurísticas utilizadas para a obtenção dos resultados deste programa, ambas estão agora localizadas dentro de uma função recursiva (*hRecursion*) que as executa intercaladamente com o objetivo de obter a melhor solução.

Esta heurística pode ser melhorada da maneira descrita previamente, caso esta seja mesclada com a *Heuristic* *2*, para que ambos o processo de inserção e troca sejam executados para maximizar a performance do programa e uma ótima solução seja obtida.

**6. Resultados Obtidos**

Após a execução do programa em sua última versão, utilizando todos os geradores de solução inicial: *Default PathFinder* (*DPF*)*;* *MaxFlow PathFinder* (*MPF*)*; Relation PathFinder* (*RPF*)*;* e *MaxFlow Relation PathFinder* (*MRPF*). Todos eles com a função *hRecursion* sendo executada logo em seguida, os melhores resultados foram então selecionados e comparados com os resultados da literatura, como pode ser visto nas tabelas a seguir:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Instância | Nº de Instalações | Resultado da Literatura | Resultado do *DPF* | Resultado do *MPF* | Resultado do *RPF* | Resultado do *MRPF* |
| Anjos\_60\_1 | 60 | **1.477.834** | 1.479.271 | 1.477.840 | 1.479.271 | 1.479.231 |
| Anjos\_60\_2 | 60 | **841.776** | 847.724 | 850.724 | 841.872 | **841.776** |
| Anjos\_60\_3 | 60 | **648.337,5** | 651.341,5 | 650.275,5 | 649.872,5 | 651.735,5 |
| Anjos\_60\_4 | 60 | **398.406** | 399.490 | 398.456 | 402.092 | 402.681 |
| Anjos\_60\_5 | 60 | **318.805** | 318.855 | 318.855 | 325.401 | 324.942 |
| Anjos\_70\_1 | 70 | **1.528.537** | 1.532.955 | 1.530.147 | **1.528.537** | 1.530.209 |
| Anjos\_70\_2 | 70 | **1.441.028** | 1.441.515 | 1.442.306 | 1.442.306 | 1.443.917 |
| Anjos\_70\_3 | 70 | **1.518.993,5** | 1.544.460,5 | **1.518.993,5** | 1.519.721,5 | **1.518.993,5** |
| Anjos\_70\_4 | 70 | **968.796** | 969.467 | **968.796** | 969.467 | 968.830 |
| Anjos\_70\_5 | 70 | **4.218.002,5** | 4.231.971,5 | 4.219.184,5 | 4.218.003,5 | 4.226.960,5 |
| Anjos\_75\_1 | 75 | **2.393.456,5** | 2.413.963,5 | 2.416.686,5 | 2.415.678,5 | 2.402.206,5 |
| Anjos\_75\_2 | 75 | **4.321.190** | 4.325.544 | 4.322.283 | 4.329.296 | 4.321.389 |
| Anjos\_75\_3 | 75 | **1.248.423** | 1.251.204 | 1.252.432 | 1.248.607 | 1.248.607 |
| Anjos\_75\_4 | 75 | **3.941.816,5** | 3.945.409,5 | 3.950.040,5 | 3.951.256,5 | 3.947.605,5 |
| Anjos\_75\_5 | 75 | **1.791.408** | 1.792.804 | 1.810.969 | 1.800.908 | 1.792.808 |
| Anjos\_80\_1 | 80 | **2.069.097,5** | 2.071.227,5 | 2.070.680,5 | 2.070.923,5 | 2.070.076,5 |
| Anjos\_80\_2 | 80 | **1.921.136** | 1.921.212 | 1.921.202 | 1.922.852 | **1.921.136** |
| Anjos\_80\_3 | 80 | **3.251.368** | 3.262.162 | 3.259.213 | 3.262.162 | 3.282.988 |
| Anjos\_80\_4 | 80 | **3.746.515** | 3.770.396 | 3.748.514 | 3.749.027 | 3.752.961 |
| Anjos\_80\_5 | 80 | **1.588.885** | 1.595.095 | 1.598.654 | 1.595.092 | 1.598.206 |

Tabela 1. Comparação dos resultados do programa com os resultados da literatura encontrados em [11, 26, 34].

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Instância | Nº de Instalações | Resultado da Literatura | Resultado do *DPF* | Resultado do *MPF* | Resultado do *RPF* | Resultado do *MRPF* |
| sko\_42\_1 | 42 | **25.525** | 25.770 | 25.532 | 25.878 | 25.987 |
| sko\_42\_2 | 42 | **216.120,5** | **216.120,5** | 216.289,5 | 216.470,5 | 216.209,5 |
| sko\_42\_3 | 42 | **173.267,5** | 175.667,5 | 175.616,5 | 174.760,5 | 173.663,5 |
| sko\_42\_4 | 42 | **137.615** | 137.987 | 137.839 | 138.275 | 139.904 |
| sko\_42\_5 | 42 | **248.238,5** | 252.515,5 | 251.893,5 | 251.675,5 | **248.238,5** |
| sko\_49\_1 | 49 | **40.967** | 41.062 | 41.311 | 41.173 | 41.106 |
| sko\_49\_2 | 49 | **416.178** | 420.653 | 416.273 | 418.707 | 421.830 |
| sko\_49\_3 | 49 | **324.512** | 328.673 | 324.550 | 327.605 | 330.714 |
| sko\_49\_4 | 49 | **236.755** | 237.972,5 | 237.360,5 | 239.503,5 | 237.446,5 |
| sko\_49\_5 | 49 | **666.143** | 674.512 | 675.957 | 679.306 | 681.154 |
| sko\_56\_1 | 56 | **64.024** | 64.395 | 64.472 | 64.472 | 64.442 |
| sko\_56\_2 | 56 | **496.561** | 499.333 | 496.790 | **496.561** | 500.295 |
| sko\_56\_3 | 56 | **170.449** | 170.581 | 173.040 | 171.271 | 171.369 |
| sko\_56\_4 | 56 | **313.388** | 317.066 | 317.262 | 318.607 | 313.841 |
| sko\_56\_5 | 56 | **592.294,5** | 592.335,5 | 592.466,5 | 592.443,5 | 592.316,5 |
| sko\_64\_1 | 64 | **96.881** | 97.874 | 98.504 | 97.659 | 97.654 |
| sko\_64\_2 | 64 | **634.332,5** | 645.325,5 | 642.236,5 | 640.203,5 | 645.272,5 |
| sko\_64\_3 | 64 | **414.323,5** | 422.079,5 | 416.918,5 | 420.392,5 | 416.897,5 |
| sko\_64\_4 | 64 | **297.129** | 300.232 | 298.716 | 299.789 | 298.645 |
| sko\_64\_5 | 64 | **501.922,5** | 506.698,5 | 504.557,5 | 507.517,5 | 505.159,5 |
| sko\_72\_1 | 72 | **139.150** | 143.026 | 143.626 | 140.949 | 141.971 |
| sko\_72\_2 | 72 | **711.998** | 717.731 | 715.434 | **711.998** | 712.390 |
| sko\_72\_3 | 72 | **1.054.110,5** | 1.065.797,5 | 1.061.445,5 | 1.061.700,5 | 1.057.479,5 |
| sko\_72\_4 | 72 | **919.586,5** | 922.083,5 | 928.923,5 | 930.889,5 | 930.105,5 |
| sko\_72\_5 | 72 | **428.226,5** | 431.051,5 | 431.960,5 | 430.057,5 | 433.650,5 |
| sko\_81\_1 | 81 | **205.106** | 207.793 | 207.376 | 207.001 | 206.207 |
| sko\_81\_2 | 81 | **521.391,5** | 521.570,5 | 526.408,5 | 525.407,5 | 525.848,5 |
| sko\_81\_3 | 81 | **970.796** | 979.050 | 972.751 | 977.589 | 972.979 |
| sko\_81\_4 | 81 | **2.031.803** | 2.042.925 | 2.043.269 | 2.044.810 | 2.032.689 |
| sko\_81\_5 | 81 | **1.302.711** | 1.313.872 | 1.308.471 | 1.304.383 | 1.305.704 |
| sko\_100\_1 | 100 | **378.234** | 385.789 | 386.518 | 382.100 | 384.090 |
| sko\_100\_2 | 100 | **2.076.008,5** | 2.083.663,5 | 2.080.260,5 | 2.079.114,5 | 2.108.913,5 |
| sko\_100\_3 | 100 | **16.145.598** | 16.307.000 | 16.262.532 | 16.316.380 | 16.240.737 |
| sko\_100\_4 | 100 | **3.232.522** | 3.251.013 | 3.249.749 | 3.250.217 | 3.250.280 |
| sko\_100\_5 | 100 | **1.033.080,5** | 1.039.985,5 | 1.034.709,5 | 1.035.896,5 | 1.044.390,5 |

Tabela 2. Compação dos resultados do programa com os resultados da literatura encontrados em [5, 6, 19, 21, 22, 23, 24, 29].

**7. Discussões / Conclusões**

Após a realização da pesquisa da literatura, da criação dos algoritmos neste relatório apresentados e da execução dos testes nas instâncias, foi possível perceber o quanto a matemática está relacionada com este problema, o quanto a lógica por trás da permutação e combinação é muito importante tanto na questão computacional, quanto temporal. Pequenas alterações no código-fonte demonstram grandes custos computacionais, mas uma melhora na solução final. Este problema, como problemas familiares a ele, necessitam de um profundo estudo para a criação de novos algoritmos que consigam criar soluções ainda melhores que as soluções da literatura. Embora poucas soluções da literatura tenham sido alcançadas, as não alcançadas estão apenas a alguns passos de serem alcançadas, ou até mesmo superadas. Creio que melhorias ainda podem ser efetuadas neste próximo semestre, tanto as descritas neste relatório, quanto novas maneiras de criação ou melhorias da solução.

**8. Referências**

1. Amaral, A.R.S.: On the exact solution of a facility layout problem. Eur. J. Oper. Res. 173(2), 508–518 (2006)

2. Amaral, A.R.S.: An exact approach to the one-dimensional facility layout problem. Oper. Res. 56(4), 1026–1033 (2008)

3. Amaral, A.R.S.: A mixed 0–1 linear programming formulation for the exact solution of the minimum linear arrangement problem. Optim. Lett. 3, 513–520 (2009)

4. Amaral, A.R.S.: A new lower bound for the single row facility layout problem. Discrete Appl. Math. 157(1), 183–190 (2009)

5. Amaral, A.R.S., Letchford, A.N.: A polyhedral approach to the single row facility layout problem. Math. Program. (2012). 10.1007/s10107-012-0533-z

6. Amaral, A.R.S., & Letchford, A.N. (2013). A polyhedral approach to the single row facility layout problem. Mathematical Programming, 141(1–2), 453–477.

7. Anjos, M.F., Kennings, A., Vannelli, A.: A semidefinite optimization approach for the singlerow layout problem with unequal dimensions. Discrete Optim. 2(2), 113–122 (2005)

8. Anjos, M.F., Vannelli, A.: Computing globally optimal solutions for single-row layout problems using semidefinite programming and cutting planes. INFORMS J. Comput. 20(4), 611– 617 (2008)

9. Anjos, M.F., Yen, G.: Provably near-optimal solutions for very large single-row facility layout problems. Optim. Methods Softw. 24(4–5), 805–817 (2009)

10. Beghin-Picavet, M., Hansen, P.: Deux problèmes daffectation non linéaires. RAIRO, Rech. Opér. 16(3), 263–276 (1982)

11. Datta, D., Amaral, A.R.S., Figueira, J.R.: Single row facility layout problem using a permutation-based genetic algorithm. Eur. J. Oper. Res. 213(2), 388–394 (2011)

12. de Alvarenga, A.G., Negreiros-Gomes, F.J., Mestria, M.: Metaheuristic methods for a class of the facility layout problem. J. Intell. Manuf. 11, 421–430 (2000)

13. Díaz, J., Petit, J., Serna, M.: A survey of graph layout problems. ACM Comput. Surv. 34(3), 313–356 (2002)

14. Djellab, H., Gourgand, M.: A new heuristic procedure for the single-row facility layout problem. Int. J. Comput. Integr. Manuf. 14(3), 270–280 (2001)

15. Glover, F.: A template for scatter search and path relinking. Lect. Notes Comput. Sci. 1363, 13–54 (1998)

16. Heragu, S.S., Alfa, A.S.: Experimental analysis of simulated annealing based algorithms for the layout problem. Eur. J. Oper. Res. 57(2), 190–202 (1992)

17. Heragu, S.S., Kusiak, A.: Machine layout problem in flexible manufacturing systems. Oper. Res. 36(2), 258–268 (1988)

18. Heragu, S.S., Kusiak, A.: Efficient models for the facility layout problem. Eur. J. Oper. Res. 53, 1–13 (1991)

19. Hungerländer, P., Rendl, F.: A computational study for the single-row facility layout problem(2011). Disponível em: <www.optimization-online.org/DB\_FILE/2011/05/3029.pdf>.

20. Hungerländer, P., Rendl, F.: Semidefinite relaxations of ordering problems. Math. Program. B (2011)

21. Kothari, R., & Ghosh, D. (2013a). Tabu search for the single row facility layout problem using exhaustive 2-opt and insertion neighborhoods. European Journal of Operational Research, 224(1), 93–100.

22. Kothari, R., & Ghosh, D. (2013b). Insertion based Lin-Kernighan heuristic for single row facility layout. Computers and Operations Research, 40(1), 129-136.

23. Kothari, R., & Ghosh, D. (2014a). An efficient genetic algorithm for single row facility layout. Optimization Letters, 8(2), 679–690.

24. Kothari, R., & Ghosh, D. (2014b). A scatter search algorithm for the single row facility layout problem. Journal of Heuristics, 20(2), 125–142.

25. Kumar, S., Asokan, P., Kumanan, S., Varma, B.: Scatter search algorithm for single row layout problem in fms. Adv. Produc. Engineer. Manag. 3(4), 193–204 (2008)

26. Lian, K., Zhang, C., Gao, L., & Shao, X. (2011). Single row facility layout problem using an imperialist competitive algorithm. In Proceedings of the 41st International Conference on Computers & Industrial Engineering

27. Love, R.F., Wong, J.Y.: On solving a one-dimensional space allocation problem with integer programming. INFOR 14(2), 139–144 (1976)

28. Nugent, C.E., Vollman, T.E., Ruml, J.: An experimental comparison of techniques for the assignment of facilities to locations. Oper. Res. 16(1), 150–173 (1968)

29. Ozcelik, F. (2012). A hybrid genetic algorithm for the single row layout problem. International Journal of Production Research, 50(20), 5872–5886.

30. Petit, J.: Experiments on the minimum linear arrangement problem. J. Exp. Algorithmics 8,Article 2.3 (2003)

31. Picard, J.C., Queyranne, M.: On the one-dimensional space allocation problem. Oper. Res. 29(2), 371–391 (1981)(pp. 578–586). Los Angeles, CA, USA.

32. Ravi Kumar, K., Hadjinicola, G.C., Lin, T.L.: A heuristic procedure for the single-row facility layout problem. Eur. J. Oper. Res. 87(1), 65–73 (1995)

33. Romero, D., Sánchez-Flores, A.: Methods for the one-dimensional space allocation problem. Comput. Oper. Res. 17(5), 465–473 (1990)

34. Samarghandi, H., & Eshghi, K. (2010). An efficient tabu algorithm for the single row facility layout problem. European Journal of Operational Research, 205(1), 98–105.

35. Samarghandi, H., Taabayan, P., Jahantigh, F.F.: A particle swarm optimization for the single row facility layout problem. Comput. Ind. Eng. 58(4), 529–534 (2010)

36. Simmons, D.M.: One-dimensional space allocation: an ordering algorithm. Oper. Res. 17(5), 812–826 (1969)

37. Solimanpur, M., Vrat, P., Shankar, R.: An ant algorithm for the single row layout problem in flexible manufacturing systems. Comput. Oper. Res. 32(3), 583–598 (2005)

**9. Auto-avaliação do bolsista**

O tempo neste projeto permitiu-me expandir meu conhecimento sobre diversos aspectos da computação, lógica, ciência e matemática, principalmente durante a construção do código-fonte utilizado, onde foi necessária a criação de lógicas complexas computacionalmente para a execução de uma ordem específica de instruções. Acredito que estou cumprindo as funções a mim designadas neste projeto durante meu tempo trabalhando nele e espero poder superar as espectativas neste próximo semestre.

**10. Avaliação do bolsista pelo orientador**

**11. Anexos**

**12. Assinaturas**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Bolsista: Leonardo Pellegrini Silva Orientador: João Carlos Furtado