Uma Heurística Baseada em GRASP + Path-Relinking para o problema de Localização de Instalações sem Capacidade

Elias B. Ferreira, Raul W. M. Costa

[1]Departamento de Ciência da Computação – Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte – MG – Brazil

eliasbf@gmail.com, raul.wagner@live.com

Abstract. *This work presents a comparative study between the Tabu Search algorithm, and two metaheuristics (GRASP + Path-Relinking) for solving the Uncapacited Locating Facilities Problem (UFLP). The instances provided by Bilde-Krarup were used to perform the tests, in which we evaluated three metrics: quality of the solution, time of execution and robustness of the solutions between the instances of the set. We concluded that, despite the better execution time, the Tabu Search method returned worse solutions than GRASP + Path-Relinking. The "pure"GRASP method did not show an improvement in the quality of the solutions, despite the significant increase in the execution time in relation to the Tabu Search.*

Resumo. *Este trabalho apresenta um estudo comparativo entre o algoritmo Tabu Search e duas metaheurísticas(GRASP+Path-Relinking) para solução do problema de localização de instalações sem capacidade (UFLP). Foram usadas as instâncias fornecidas por Bilde-Krarup para execução dos testes, em que avaliamos três métricas: qualidade da solução, tempo de execução e robustez das soluções entre as instâncias do conjunto. Concluímos que, apesar do melhor tempo de execução, o método Tabu Search retornou soluções piores do que o GRASP+Path-Relinking. O método GRASP "puro"não apresentou uma melhora na qualidade das soluções, apesar do aumento significativo no tempo de execução em relação ao Tabu Search.*

1. Introdução

O Problema de localização de instalações sem capacidade (UFLP, do inglês Uncapacitated facility location problem) é um problema clássico da área de otimização combinatória. Esse problema lida com a seleção de locais para abrir novas instalações, dentro de uma gama finita de possíveis locais, em que cada localidade possui um custo fixo para abertura de uma nova instalação. Essa seleção deve ser feita visando minimizar o custo total de abertura das instalações e o custo de conexão entre uma dada instalação e os clientes que essa instalação atende.

Pela própria definição do problema tratado, é fácil perceber que ele possui aplicações em várias áreas como logística, planejamento de centros de distribuição/varejo, localização de hospitais, localização de unidades de saúde, redes de comunicação, localização de aterros sanitários, etc. A título de exemplo, [Ahmadi-Javid et al. 2017] observaram uma correlação entre a má localização de unidades de saúde e o aumento da

mortalidade em uma região. Isso demonstra a relevância econômica e social do problema de localização de instalações sem capacidade.

Como um problema de otimização combinatória, o problema UFL pode ser formalmente definido da seguinte maneira:

- **Entrada**: conjunto de cliente D, conjunto de possíveis locais para abertura de instalações F, distância $d : D \times F \rightarrow \Re^+$ e custo de abertura $f : F \rightarrow \Re^+$.
- **Soluções viáveis**: um subconjunto F', tal que $\emptyset \neq F' \subseteq F$.
- **Função objetivo**: custo de abertura mais custo de conexão, i.e., $\sum_{i \in F'} f(i) + \sum_{j \in D} d(j, a(j))$, com $a(j)$ instalação mais próxima de j em F'.
- **Restrições**: cada cliente deve ser atendido por somente uma instalação.
- **Objetivo**: encontrar solução de custo mínimo.

Apesar de sua importância, o problema UFL pertence à classe de problemas NP-difícil e, por isso, não temos conhecimento de um método exato que garanta uma solução eficiente para instâncias de qualquer tamanho. Para contornar esse problema, é comum recorrer a métodos heurísticos a fim de aproximar das soluções ótimas.

Nesse trabalho, foi implementado e comparado dois métodos metaheurísticos. O primeiro método é baseado no algoritmo ***Greedy Randomized Adaptive Search Procedure (GRASP)*** com o uso do método de ***Path Relinking (PR)*** e o segundo, que servirá como baseline, é uma implementação do algoritmo *Tabu Search* que apresenta resultados promissores na literatura. Os critérios de comparação considerados são: tempo de execução, qualidade dos resultados obtidos e robustez dos resultados obtidos.

Para execução dos testes, foram selecionadas instâncias do conjunto gerado por [Bilde and Krarup 1977]. Esse conjunto, que é bastante usado na literatura, consiste em 220 instâncias distintas, onde cada instância apresenta um conjunto de possíveis locais para as instalações, com o custo de cada local, um conjunto de clientes que devem ser atendidos e o custo para conectar cada instalação à cada cliente.

2. Trabalhos Relacionados

Várias metodologias foram propostas para resolver o Problema de localização de instalações sem capacidade. Está fora do escopo deste trabalho revisar a riqueza de resultados para esse problema, porém serão pontuados alguns métodos que usam diferentes estratégias para resolução de problemas de otimização combinatória. Dentre as estratégias que estes trabalhos relacionados abrangem estão métodos exatos, de aprimoramento iterativo e aprimoramento de populações.

O algoritmo *Tabu Search* é um método clássico para problemas de otimização combinatória. Sua ideia principal consiste em realizar uma busca local no espaço de vizinhos de uma solução inicial. Para evitar ótimos locais previamente visitados, é usada uma estrutura chamada lista tabu. Essa estrutura armazena, em sua forma mais básica, uma lista com as n últimas soluções visitadas. As soluções nessa lista são consideradas tabu e não podem ser visitadas novamente, enquanto estiverem na lista.

Laurent M. e Pascal Van H. [Laurent and Pascal 2004] apresentam uma implementação simples, porém robusta e eficiente, do algoritmo de Tabu Search. Em sua implementação, cada local candidato a abertura de uma instalação é um elemento de um vetor, onde 1 representa que uma instalação foi construída no local, e 0 que não

foi construída. Dessa forma, a lista tabu é definida como um conjunto de posições do vetor de localizações que devem ficar travados, ou seja, não podem mudar seu status. Essa implementação apresentou resultados promissores quanto comparado a outras três metodologias que serviram de benchmark.

Uma das metodologias que serviram de base para esse estudo foi desenvolvida por Kratica et al. [Kratica et al. 2001]. A metodologia empregada foi um algoritmo genético com um selecionador baseado em ranking, em que, por volta de $1/3$ da população, de 500 indivíduos, é substituída em cada iteração. Novos indivíduos são gerados a partir de operações de *cross-over* e mutações. O algoritmo usa ainda alguns refinamentos como *fitness adaptation* e técnicas que usam caches para otimização do tempo de execução.

Outro estudo importante foi desenvolvido por Erlenkotter [Erlenkotter 1978]. Esse estudo aplica um algoritmo de *branch-and-bound* baseado em uma estratégia de *dual descent*. O algoritmo, nomeado DUALOC, apresentou uma boa performance para instâncias pequenas, mas falhou em retornar bons resultados em instâncias maiores, como é esperado para a aplicação de um algoritmo exato nesse problema.

O algoritmo **greedy randomized adaptive search procedure**, também conhecido como GRASP, foi proposto por Feo et al. [Feo and Resende 1995], visando se aproveitar do fato de que algoritmos de busca local são sensíveis à solução inicial recebida. A estratégia básica adotada pelo método é realizar buscas locais em múltiplas soluções iniciais aleatórias geradas por heurísticas construtivas. Esse método, apesar de simples, apresenta bons resultados para vários problemas de otimização [Festa and Resende 2011].

Dentre as metodologias citadas, foi observado que o *Tabu Search* se destaca pela qualidade dos resultados obtidos, tempo de execução, assim como a simplicidade da implementação proposta por Laurent M., Pascal Van H. [Laurent and Pascal 2004]. Portanto, esse algoritmo foi selecionado para servir de baseline para a metodologia que estamos propondo para o Problema de localização de instalações sem capacidade nesse projeto.

3. Estratégia de solução

3.1. Metaheurística GRASP

Algoritmo proposto por [Feo and Resende 1995], o GRASP (**greedy randomized adaptive search procedure**) é constituído de duas fases: a primeira fase a ser realizada é a construtiva, onde uma solução viável para o problema é criada e a segunda trata-se de uma fase de busca em vizinhança (busca local), que atua com o objetivo de melhorar a solução obtida na fase construtiva. O GRASP é um algoritmo iterativo que, normalmente, utiliza como critério de parada o número de iterações. Após cada iteração, um ótimo local é encontrado e a melhor soluções de todas as iterações é retornada.

A cada iteração um conjunto de soluções viáveis são geradas de forma gulosas e aleatória e armazenadas em uma lista restrita de candidatos (LRC). A capacidade da LRC é definida pelo parâmetro α, de forma que este valor igual a 0 (zero), implica em soluções puramente gulosas, enquanto que se o valor for igual a 1 as soluções geradas são puramente aleatórias.

A segunda fase do GRASP, fase de busca local, ocorre no final de cada iteração e nada mais é que uma tentativa de melhoria da solução obtida. Utilizando uma estratégia

de intensificação esse algoritmo explora a vizinhança em busca de um ótimo local. Por exemplo, no problema UFLP iremos ativar (ou desativar) uma instalação e verificar se houve redução no custo total, este passo se repete até que haja n soluções sem melhoria.

3.2. Metaheurística Path-Relinking

A heurística *path-relinking* foi descrita originalmente por [Glover 1997] e consiste em explorar o trajeto entre duas soluções, uma denominada solução inicial e outra chamada solução guia. A cada iteração é realizado um movimento na solução inicial para aproxima-la da solução guia. Portanto, inicialmente calcula-se a diferença simétrica entre as duas, ou seja, são identificados os movimentos que estão na solução guia e não na solução inicial, em seguida, cada movimento da diferença simétrica é aplicado à solução inicial e, havendo melhoria no custo total deste novo caminho, o mesmo é escolhido. Este processo continua até que todos os movimentos sejam aplicados. É uma estratégia que pode ser compreendida como um balanço entre intensificação e diversificação, pois explora o espaço de soluções entre as soluções inicial e guia.

3.3. Solução

A primeira etapa da solução é a fase de preparação, que constitui na leitura de cada instância e armazenamento nas estruturas de dados apropriadas, conforme descrito a seguir:

- Estrutura de dados com custo de abertura: corresponde ao custo de abertura de cada uma das instalações para uma dada instância do problema.
- Estrutura de dados com o custo de conexão: corresponde a distância dos clientes a cada uma das instalações. Portanto, é uma matriz $n \times m$ (instalações x clientes).

Após a preparação das instâncias, inicia-se a atividade de implementação da solução. Esta fase é composta de duas etapas. Inicialmente o algoritmo GRASP é implementando e, em seguida o *Path-Relinking* é implementado. As duas fases podem ser resumidas da seguinte forma:

- **Fase 1, GRASP**: tem por objetivo construir uma solução inicial de forma aleatorizada (etapa construtiva) e, em seguida, buscar melhora-la por meio da etapa de busca local. Esta fase é executada por k iterações com parâmetro de intensificação e diversificação alpha configurado para 40% e as x melhores soluções encontradas são retornadas.
- **Fase 2, Path-Relinking (PR)**: nessa fase, a heurística PR irá analisar e processar as x melhores soluções (soluções elites) retornadas pela heurística GRASP (fase 1). Dessa forma, as $x-1$ soluções são comparadas com a melhor solução (solução *guia*) para fins de melhoria. Retorna a melhor solução alcançada.

O algoritmo 1 detalha as características da solução. O conjunto de possíveis instalações F e a quantidade de clientes D são recebidos pela função. A linha 1, faz a chamada à fase construtiva aleatorizada e a linha 2 a fase de busca local. Na fase de busca local são retornadas as x melhores soluções (soluções elites), sendo que a melhor delas é escolhida como a **solução guia** (S_g).

Para cada solução elite retornada o método *Path-relinking* é executado (linhas 3 a 10). O laço de repetição da linha 3 mantém a execução da heurística *Path-relinking*

enquanto houver soluções elite no conjunto. Inclusive, essa é uma adaptação ao método *Path-relinking*, ou seja, aplica-lo a várias instâncias de soluções elite e não a apenas uma - essa adaptação apresentou uma melhora de 2% nas soluções elite. A linha 4 escolhe a melhor solução elite e atribui a S_i (solução inicial), em seguida, remove está solução do conjunto (linha 5). A linha 6 calcula a diferença simétrica entre as soluções S_g e S_i, ou seja, quais os movimentos necessários para transforma a solução inicial S_i na solução guia S_g. Em seguida, entre as linhas 7 a 10, cada movimento da diferença simétrica é aplicado na solução inicial S_i e, a cada movimento, uma nova solução vizinha é gerada e comparada para identificar se houve redução no custo da solução, se a resposta for positiva, essa solução se torna a nova solução guia S_g (linha 9), e será utilizada na próxima iteração. Ao fim da execução a solução de menor custo encontrada será retornada.

Algorithm 1: $GRASPePathRelinkingSolution$

```
   Data: D: conjunto de clientes
   Data: F: conjunto de instalações
   Data: alfa
   Data: n: quantidade de melhores soluções
   Result: Melhor solução encontrada
1  melhoresSolucoes = ConstrucaoGulosaAleatoria(alfa, n);
2  melhoresSolucoes, s_g = BuscaLocal(melhoresSolucoes);
3  while melhoresSolucoes ≠ ∅ do
4      s_i = melhoresSolucoes[0];
5      melhoresSolucoes = melhoresSolucoes \{s_i} ;
6      Δ = CalcularDiferençaSimetrica(s_g, s_i);
7      while Δ ≠ ∅ do
8          AbreOuFechaInstalacao(S_i, Δ_i);
9          AtualizaMelhorSolucao(S_g);
10         Δ = Δ \ {Δ_i}
```

4. Experimentos Computacionais

Os experimentos computacionais que avaliamos foram gerados usando uma maquina com as seguintes características: processador Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz, 16 Gb de RAM, SSD 250 Gb, Sistema Operacional Ubuntu 18.04 e o g++ 7.5.0.

Na comparação entre os algoritmos propostos analisamos três critérios distintos através da execução de testes em um conjunto de instâncias. Os critérios analisados foram:

1. A qualidade da solução, medida pela aproximação percentual que a implementação chegou do valor ótimo de cada instância;
2. O tempo de execução de cada implementação, medido através da média de 100 execuções, para cada algoritmo, no conjunto de instâncias selecionadas;
3. A robustez da solução, que foi medida pela variância do erro entre o resultado obtido e o valor do resultado ótimo de cada instância;

Durante a realização dos experimentos, usamos o conjunto de instâncias Bilde-Krarup [Bilde and Krarup 1977]. Esse conjunto é composto por 220 instâncias, que são segmentadas em 22 classes distintas, portanto com 10 instâncias pertencendo a cada

classe. Para uma dada classe, todas as instâncias dessa classe mantém um numero constante de locais e clientes, por exemplo, todas instancias da classe D3 possuem 30 locais e 80 clientes. As instâncias desse conjunto foram geradas de maneira artificial usando distribuições estatísticas.

Em todas instâncias, o custo de distância entre uma instalação e um cliente foi gerado a partir de uma distribuição uniforme discreta, com valores entre 0 e 1000. Para definir o custo de se abrir uma instalação em um determinado local, foram usadas duas distribuições estatísticas: uma distribuição uniforme discreta entre 0 e 1000 e uma função de custo constante para todos locais, onde cada classe de instâncias adotou uma única função. A organização das funções usadas em cada classe, assim como o tamanho das instâncias de cada classe podem ser vistos na Tabela 1.

Tabela 1. Característica de cada classe de instâncias. O campo size define a quantidade de locais disponíveis e o número de clientes que devem ser atendidos. [Bilde and Krarup 1977]

Problem class	Size	Openning costs	Connection costs
B	50/100	Discret uniform(1000,10000)	Discret uniform(0,1000)
C	50/100	Discret uniform(1000,2000)	Discret uniform(0,1000)
Dq*	30/80	Identical, q*1000	Discret uniform(0,1000)
Eq*	50/100	Identical, q*1000	Discret uniform(0,1000)
(*)q= 1...10			

Foi levantado como hipótese que a implementação do método GRASP, com o *Path-Relinking* poderia superar a qualidade dos resultados obtidos pelo Tabu Search para o problema de UFL. Essa análise foi feita levando em consideração a qualidade do resultado, tempo de execução e robustez das soluções obtidas.

A implementação do método *Tabu Search* foi feita baseada no trabalho desenvolvido por [Laurent and Pascal 2004]. Uma ideia geral do algoritmo pode ser visto abaixo:

Algorithm 2: $TabuSearch$

Data: D: conjunto de clientes
Data: F: conjunto de instalações
Data: alfa
Data: n: quantidade de melhores soluções
Result: Melhor solução encontrada

```
melhoresSolucoes = ConstrucaoGulosaAleatoria(alfa, n);
melhoresSolucoes, s_g = BuscaLocal(melhoresSolucoes);
while iteracoesSemMelhora ≤ maxIteracoesSemMelhora do
    s_i = BuscaLocal(melhoresSolucoes)[0] ;
    if s_i ≤ solucaoAtual then
        solucaoAtual = s_i;
    else
        incrementa iteracoesSemMelhora ;
        s_i = solucaoAleatoriaNaoTabu;
        inverte status de um local aleatorio;
```

Com a realização dos testes, conseguimos gerar a tabela contida na Tabela 2, que contem uma visão comparativa geral dos métodos testados.

Tabela 2. Sumário comparativo dos resultados obtidos.

	Tabu Search	**GRASP+PR**
% de otimalidade	19,1%	32,3%
Instâncias ótimas	42	71
Desvio padrão (%)	2,77%	1,78%
Desvio padrão (quantidade)	1.075,4	638,2
Melhor que Tabu Search(%)	-	73,36%
Melhor que Tabu Search(quantidade)	-	168

Pela Tabela 2, é fácil notar que a implementação da metaheuristica *GRASP+PR* teve resultados significativamente melhores do que os apresentados pela implementação feita do *Tabu Search*. Essa melhora se mostra na proporção de soluções com melhor qualidade, no número de resultados ótimos obtidos e na robustez da metodologia, ou seja, no desvio padrão que o método apresentou da diferença entre a solução obtida e a solução ótima. Em outras palavras, em média, a implementação do *GRASP+PR* é mais acertiva e a sua acertividade se mantém para diferentes instâncias.

Pela Tabela 2 conseguimos observar que uma parte significativa do ganho no custo da solução *GRASP+PR* se deve ao algoritmo de *Path-Relinking*. Isso mostra a capacidade que esse método tem de evitar mínimos locais e encontrar soluções melhores que o *GRASP* "puro".

A Figura 2 ilustra claramente o dilema que existe entre tempo de execução e qualidade da solução obtida para heurísticas. Enquanto a solução construtiva encontrou soluções em um curto espaço de tempo, a qualidade das soluções encontradas foi baixa. Em contra partida, o método *GRASP+PR* encontrou soluções muito boas. Como vimos na figura 1, para as instancias testadas, nenhuma das soluções retornadas desse método

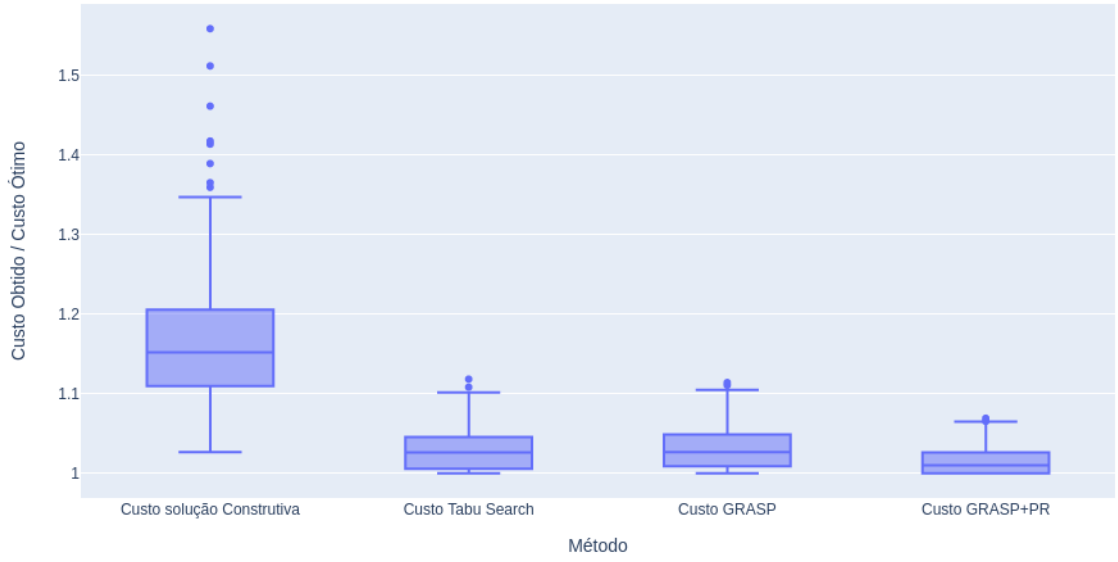

Figura 1. Boxplot do custo obtido/custo ótimo de cada método.

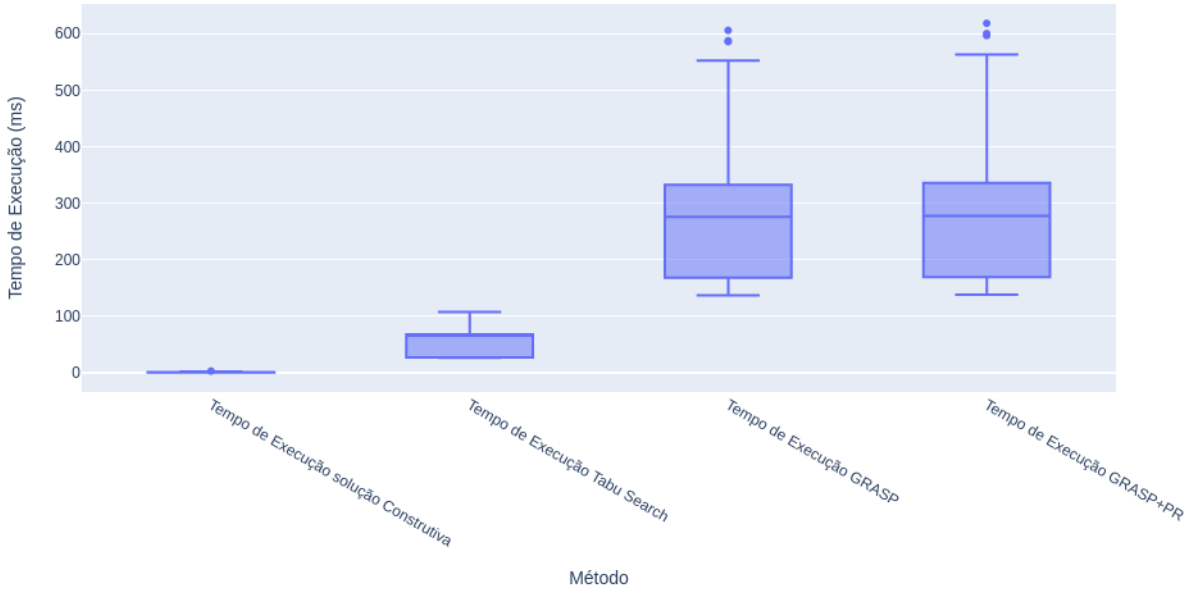

Figura 2. Boxplot do tempo de execução de cada método.

foi 10% pior do que a solução ótima. Porém esse ganho veio com o custo de um maior tempo de execução.

Foi também verificado que os métodos testados não apresentaram uma variação significativa nem na qualidade da solução, nem no tempo de execução entre diferentes classes de instâncias.

Com isso podemos concluir que os resultados obtidos pelos experimentos dão um suporte parcial as hipóteses levantada por esse estudo. As funções de custo retornadas pelo método *GRASP+PR* foram significativamente melhores do que os obtidos pelo *Tabu Search*, para as instancias fornecidas por Bilde-Krarup. Por outro lado, o tempo de

execução do *GRASP+PR* foi superior ao *Tabu Search* em até 6 vezes, nas instâncias que mostraram um maior tempo de processamento.

5. Conclusão

Com esse estudo, concluímos portanto que, apesar de apresentar um tempo de processamento significativamente maior, o método *GRASP+PR* despontou em relação ao *Tabu Search* em termos de qualidade. Notamos também que o método *GRASP* puro apresentou um tempo de execução maior do que o *Tabu Search* e, apesar disso, não apresentou uma melhora significativa na qualidade das soluções fornecidas.

Para trabalhos futuros entendemos que seria interessante analisar o comportamento de um método *Tabu Search* unido ao *Path-Relinking*, comparando esse método com os resultados que obtemos para o *GRASP+PR*, em termos de qualidade das soluções retornadas e em tempo de execução.

Referências

Ahmadi-Javid, A., Seyedi, P., and S. Syam, S. (2017). A survey of healthcare facility location, computers operations research. 79:223–263.

Bilde, O. and Krarup, J. (1977). Sharp lower bounds and efficient algorithms for the simple plant location problem annals of discrete mathematics. 1:79–97.

Erlenkotter, D. (1978). A dual-based procedure for uncapacitated facility location: General solution procedures and computational experience. *Operations Research*, 26:937–1094.

Feo, T. A. and Resende, M. G. C. (1995). Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6:109–133.

Festa, P. and Resende, M. (2011). *Effective Application of GRASP*.

Glover, F. (1997). *Tabu Search and Adaptive Memory Programming — Advances, Applications and hallenges*, pages 1–75. Springer US, Boston, MA.

Kratica, J., Tošić, D., Filipović, V., Ljubic, I., and Tolla, C. (2001). Solving the simple plant location problem by genetic algorithm. *RAIRO - Operations Research*, 35:127–142.

Laurent, M. and Pascal, V. (2004). A simple tabu search for warehouse location. *European Journal of Operational Research*, 157(3):576–591.