LII Simpósio Brasileiro de Pesquisa Operacional
João Pessoa - PB, 3 a 5 de novembro de 2020

Um algoritmo branch-and-cut-and-price para o Problema de Localização de Instalações Capacitado com Única Fonte

Verônica de Miranda Prottes
Universidade Federal Fluminense
Rua Passo da Pátria,156, Bloco E, 4o andar, sala 440, São Domingos, Niterói, RJ
veronicaprottes@id.uff.br

Artur Alves Pessoa
Universidade Federal Fluminense
Rua Passo da Pátria,156, Bloco E, 4o andar, sala 440, São Domingos, Niterói, RJ
artur@producao.uff.br

RESUMO

Este trabalho apresenta um algoritmo exato para o SSCFLP - Problema de Localização de Instalações Capacitado com Única Fonte. O algoritmo é desenvolvido em duas etapas. A primeira utiliza planos de corte para fortalecimento da formulação e fixação das variáveis, de modo a reduzir substancialmente o tamanho do problema, e a segunda etapa aplica BCP no VRPSolver, um solver genérico para problemas de roteamento e demais problemas com estrutura semelhante. No modelo criado no VRPSolver, as restrições de capacidade são modeladas como recursos em grafos, permitindo que os subproblemas de *pricing* sejam resolvidos como Problemas de Caminho mais Curto Restringido por Recursos-RCSPs. Os resultados computacionais mostram o grande potencial em desenvolvimento deste algoritmo em comparação com o estado-da-arte, apresentando baixos *gaps* entre o limite inferior e a solução ótima e tendo um desempenho muito superior em algumas instâncias consideradas difíceis até o momento.

PALAVRAS CHAVE. Localização de instalações capacitado. Programação inteira. Algoritmo.

Tópicos (Otimização Combinatória, Programação Matemática, Logística)

ABSTRACT

This paper presents an exact algorithm for the single-source capacitated facility location problem - SSCFLP. The algorithm consists in two phases. The first phase uses cutting planes and variable fixing to strengthens the formulation , in order to substantially reduce the problem size, and the second phase applies a BCP framework on VRPSolver, a generic solver for vehicle routing and other related problems. In VRPSolver model, capacity constraints are modeled like resources on graphs, allowing pricing subproblems be solved through Resource Constrained Shortest Path Problems - RCSPs. The computational results shows the great potential of this algorithm comparing to state-of-art specific-problem algorithm, with very small gaps between the lower bound and the optimal solution and exhibiting a better performance on hard instances that have been considered so far.

KEYWORDS. Capacitated Facility Location. Integer Programming. Algorithm.

Paper topics (Combinatorial Optimization, Mathematical Programming, Logistics))

1. Introdução

O problema de Localização de Instalações Capacitado com Única Fonte - ou SSCFLP (*Single-source Capacitated Facility Location Problem*)- é o problema cujo objetivo é abrir de um conjunto de instalações às quais são atribuídas clientes, respeitando as capacidades das instalações enquanto minimiza o custo total de atribuição e abertura. A diferença entre este problema e o Problema de Localização de Instalações Capacitado - ou CFLP(*Capacitated Facility Location Problem*) - é que a demanda de cada cliente deve ser suprida a partir de uma única instalação. O problema, então, torna-se mais complexo em sua resolução se comparado com o CFLP e, de fato, é conhecido que o SSCFLP é fortemente NP-difícil, sendo provado em Karp [1972] que ele é reduzido polinomialmente ao problema de cobertura por nós. Assim, a maior parte das pesquisas tem se restringido à abordagens heurísticas para a solução do problema.

Como é conhecido que o limite inferior da relaxação do problema é fraco, a maioria das abordagens sobre o problema utiliza relaxação lagrangeana sobre um conjunto de restrições, seja sobre as restrições de capacidade ou de demanda. Gadegaard et al. [2017] apresenta algumas considerações sobre trabalhos que utilizam heurísticas baseadas nesta abordagem. Devido ao número de variáveis inteiras existentes nos modelos deste problema, menor atenção é dada a abordagens exatas para a solução do problema.

Dentre as principais abordagens exatas, Holmberg et al. [1999] relaxou as restrições de demanda através de uma abordagem Langrangeana para a obtenção de limites inferiores para o problema, enquanto os limites superiores são encontrados através de uma heurística de *matching* repetida. Ambos os limites são utilizados em um algoritmo branch-and-bound para encontrar a solução ótima do problema. Ceselli e Righini [2005] consideram o problema p-mediana capacitado, o qual é proximamente relacionado ao SSCFLP; enquanto o último considera custos fixos para instalações, o primeiro considera que o número de abertura destas não pode exceder a p. O algoritmo branch-and-cut proposto pelos autores é baseado numa formulação do problema de cobertura por conjuntos.Yang et al. [2012] utilizou um algoritmo de plano de corte baseado na separação exata da mochila para fortalecer a formulação do SSCFLP e, em seguida, aplicar cut-and-solve para resolver o problema na otimalidade. Gadegaard et al. [2017] propôs um algoritmo baseado em três fases, sendo a primeira o fortalecimento da formulação para melhorar os limites inferiores da relaxação linear e, na segunda fase utiliza uma heurística de branching local que explora os dois níveis de decisão - localização e alocação - para chegar a uma solução inicial próxima da ótima. A terceira etapa utiliza uma estrutura cut-and-solve para resolver o problema na otimalidade. Os dois últimos trabalhos citados ilustram uma tendência no desenvolvimento dos algoritmos relacionados ao SSCFLP, isto é, o uso de métodos para o fortalecimento dos limites inferiores do problema. Além dos já citados, cortes baseados na separação da mochila utilizando desigualdades de cobertura elevada como em Kaparis e Letchford [2010] e separação exata Boyd [1993] são implementados.

Este trabalho propõe um algoritmo para o SSCFLP dividido em duas etapas. A primeira insere cortes propostos em Gadegaard et al. [2017] utilizando a implementação proposta em Avella et al. [2010] na relaxação linear com o objetivo de melhorar o limite inferior da solução para que as variáveis sejam fixadas a partir da avaliação de seus valores duais. Assim, aquelas que não fazem parte da solução final são, tanto quanto possível, retiradas do programa inteiro, reduzindo substancialmente o tamanho do problema. A segunda etapa resolve o problema de maneira exata através de um algoritmo branch-and-cut-and-price pelo VRPSolver, um solver exato genérico para problemas de roteamento e demais problemas de estrutura semelhante. O solver trata o conjunto das restrições de capacidade do SSCFLP como problemas de caminho mais curto restringido por recursos - RCSP - ou seja, as capacidades das instalações e demandas dos clientes são modeladas

como recursos em grafos, e as soluções definem a envoltória convexa das restrições de capacidade. O algoritmo é testado para o grupo de instâncias desenvolvido por Gadegaard et al. [2017] para avaliar o desempenho do algoritmo em um grupo de instâncias considerado difícil, uma vez que possui ainda instâncias em aberto.

O artigo é organizado da seguinte maneira: a seção 2 apresenta a formulação matemática para o problema SSCFLP. A seção 3 trata cada uma das etapas do algoritmo e a 4 apresenta os resultados fazendo uma comparação com o algoritmo estado-da-arte específico para o problema. A última seção apresenta as conclusões deste trabalho e as direções futuras de pesquisa.

2. Formulação do Problema

Seja F, $|F| = m$ o conjunto de potenciais locais de instalações para atendimento de clientes e C, $|C| = n$ o conjunto de clientes a serem atendidos. Cada instalação $i \in F$ possui capacidades $s_i > 0$ e cada cliente possui demanda $d_j > 0$ para $j \in C$. Para cada instalação, o custo fixo de abertura associado a ela é $f_i \geq 0$. Já o custo de associar o cliente j a instalação i é $c_{ij} \geq 0$. Seja y_i a variável binária cujo valor é igual a 1 se a instalação é aberta e 0, caso contrário. Seja x_{ij} a variável binária cujo valor é igual a 1 caso o cliente j seja associado a instalação i e 0, caso contrário. O problema pode ser então formulado como um problema de programação inteira, como a seguir:

$$\text{Min} \quad \sum_{i \in F} \sum_{j \in C} c_{ij} x_{ij} + \sum_{i \in F} f_i y_i \tag{1a}$$

$$\text{S.a.} \quad \sum_{i \in F} x_{ij} = 1, \qquad \forall j \in C \tag{1b}$$

$$\sum_{j \in C} d_j x_{ij} \leq s_i y_i \qquad \forall i \in F \tag{1c}$$

$$\sum_{i \in F} s_i y_i \geq D \tag{1d}$$

$$0 \leq x_{ij}, y_i \leq 1, \qquad \forall i \in F, \forall j \in C \tag{1e}$$

$$x_{ij}, yi \in 0, 1, \qquad \forall i \in F, \forall j \in C \tag{1f}$$

A função objetivo (1a) minimiza os custos totais associados a abertura das instalações e a atribuição dos clientes. O segundo conjunto de restrições (1b) - de *demanda* - garantem que todo cliente deve ser atribuído a uma instalação. As restrições de *capacidade* (1c) garantem que as capacidades das instalações sejam respeitadas, enquanto a restrição de *demanda total* (1d) aparece aqui, embora de maneira redundante, para fortalecer o limite inferior da relaxação linear, como será visto na metodologia deste trabalho. Os últimos dois conjuntos de restrições, (1e) e (1f), referem-se, respectivamente, às variáveis em sua forma relaxada e ao valor que assumem ao considerar a formulação do programa inteiro.

3. Metodologia

O algoritmo para o problema do SSCFLP desenvolvido neste trabalho é dividido em duas etapas. A primeira consiste basicamente em um algoritmo de plano de corte como aquele desenvolvido em Gadegaard et al. [2017]. Os cortes são implementados para fortalecer a formulação do problema de modo a obter limites inferiores melhores, reduzindo assim o tempo para a busca da solução ótima inteira. A partir do resultado da relaxação, as variáveis são fixadas a partir de um critério de avaliação de seus valores duais. As variáveis não fixadas são inseridas no programa inteiro na segunda etapa, a qual aplica um algoritmo branch-and-cut-and-price (BCP) no VRPSolver para alcançar a otimalidade. Cada uma das etapas é apresentada a seguir.

3.1. Fase 1: Planos de Corte no LP e Fixação de Variáveis

É conhecido que a relaxação do problema SSCFLP é normalmente fraca, fornecendo valores para os limites inferiores com *gaps* razoavelmente grandes para a busca de uma solução ótima. Uma forma de fortalecer a formulação é a utilização de cortes no modelo que busquem a aproximação do envoltório convexo de pelo menos um conjunto de restrições da formulação. Neste artigo busca-se aproximar o envoltório convexo das soluções inteiras para as restrições de capacidade (1c) e a restrição de demanda total (1d). Esses cortes caracterizam as facetas das estruturas de mochila que surgem das restrições de capacidade e sua demonstração encontra-se no trabalho de Gadegaard et al. [2017]. As definições seguintes são importantes para o entendimento do teorema que define as desigualdades definidoras de facetas para as restrições de capacidade. Seja

$$\mathcal{X} = x \in 0,1^m : a^T x \leq a_0$$

$$\mathcal{X}_y = (x,y) \in 0,1^m + 1 : a^T x \leq a_0 y,$$

onde $a_k > 0$ para $k = 0,1,...,m$. Adicionalmente, $a_j \leq a_0$ para $j = 1,2,...,m$. Se $\pi_x^T \leq \pi_0$ é uma desigualdade válida para $\mathcal{X}$, $\pi_x^T \leq \pi_0 y$ é desigualdade válida para $\mathcal{X}_y$. Além disso, como é facilmente visto que $\pi_x^T \leq \pi_0$ é definidora de faceta para conv($\mathcal{X}$), então $\pi_x^T \leq \pi_0 y$ é definidora de faceta para conv($\mathcal{X}_y$), implicando que os planos de cortes fortes para conv($\mathcal{X}$) serão também para conv($\mathcal{X}_y$). O teorema a seguir enuncia a relação entre os dois tipos de facetas.

Teorema 1. *As desigualdades definidoras de faceta de conv($\mathcal{X}_y$), diferente das facetas triviais $-x_j \leq 0$ e $y \leq 1$, são da forma $\pi_x^T \leq \pi_y y$, onde $\pi \geq 0$ e $\pi_y > 0$ e $\pi_x^T \leq \pi_y$ é faceta definidora para conv($\mathcal{X}$).*

Demonstração. A prova para o Teorema 1 pode ser encontrado em Gadegaard et al. [2017]. □

O que o Teorema 1 afirma é que podem ser encontradas desigualdades definidoras de facetas para conv($\mathcal{X}_y$) encontrando as desigualdades para conv($\mathcal{X}$) e fazendo a respectiva tradução dessas desigualdades. Assim, a variável de localização pode ser omitida ao gerar os cortes para as restrições de capacidade (1c) e simplesmente multiplicar o lado direito da equação obtido pela variável y correspondente. Com relação à restrição de demanda total (1d), a modificação na geração dos cortes ocorre modificando-se o sentido da desigualdade (de $\geq$ para $\leq$) e adequando o novo formato da restrição através da criação de uma nova variável auxiliar $z_i = 1 - y_i$ para se obter uma restrição do tipo mochila em seu formato normal.

Com relação à geração dos cortes, foi seguida a implementação como em Avella et al. [2010]. Vale aqui ressaltar que foi feita uma ligeira modificação na forma como são encontrados os coeficientes inteiros para as variáveis do corte. Ao invés de usar um MIP, é utilizado o algoritmo de Euclides para obtenção do máximo divisor comum para números fracionários. Além disso, outro tratamento feito durante a geração dos cortes é que se $a_j \geq a_0$ para qualquer $j = 1,2,...,m$, o valor da variável é fixada a zero e o respectivo corte $x_{ij} = 0$ é inserido no modelo.

A etapa 1 do algoritmo pode ser vista no algoritmo 1. Todas as variáveis do modelo cujo valor dual exceda a diferença entre entre o valor ótimo da solução e o valor do limite inferior fornecido pela relaxação são retiradas do modelo, pois não possuem o potencial de melhorar a solução, logo são fixadas a zero. A demais variáveis não fixadas são inseridas no modelo do VRPSolver. Esta segunda etapa é apresentada a seguir.

Algoritmo 1 Inserção de Cortes e Fixação de Variáveis

1: Resolve o LP e obtém os valores de entrada $(\underline{x}, \underline{y})$ e o valor da solução ótima Z^*;
2: Para i = 1, ..., m;

2.1: Recebe os parâmetros de entrada para o corte (os valores de $\underline{x}$, os coeficientes a_i das variáveis, o rhs e parâmetros numéricos de separação);

2.2: Separa x_i de conv$(\{\underline{x}_i \in \{0,1\}^n : \sum_{j \in C} d_j x_j \leq s_i\})$ através do processo de separação como em Avella et al. [2010]. Se a desigualdade resultante é violada por x_i, adiciona o corte $\pi x \leq \pi_0 y_i$;

2.3: Se o valor do coeficiente a_i da variável x_i na entrada for maior que o rhs da restrição, adiciona o corte na forma $x_{ij} = 0$;

3: Separa y de conv$(\{\underline{y} \in \{0,1\}^{|I|} : \sum_{i \in F} s_i y_i \geq D\})$ através do processo de separação como em Avella et al. [2010]. Se a desigualdade resultante é violada por y, adiciona o corte $\pi y \leq \pi_0$.
4: Se não houverem mais cortes a serem inseridos em 1.2 e em 2, retorna$(\underline{x}, \underline{y})$ e vai para o passo 5. Caso contrário, vai para o passo 0.
5: Obtém os valores duais das variáveis y_i da solução para $i = 1, \ldots, m$; se $y_{i_{dual}} > Z^* - \underline{Z}^*$ fixa a variável a 0;
6: Obtém os valores duais das variáveis x_{ij} da solução para $i = 1, \ldots, m$, $j = 1, \ldots, n$; se $x_{ij_{dual}} > Z^* - \underline{Z}^*$ fixa a variável a 0;
7: Retorna as variáveis x e y não fixadas e todos os cortes para inserção na etapa 2.

3.2. Fase 2: Resolução Exata Usando o VrpSolver

A segunda etapa do algoritmo consiste em resolver o modelo gerado na etapa 1, após a fixação das variáveis, através de Branch-and-Cut-and-Price (BCP) no VRPSolver. O VRPSolver é um solver MIP genérico para problemas de roteamento de veículos e outros problemas relacionados cuja modelagem do problema apresenta variáveis associadas a caminhos restringidos por recursos sobre grafos orientados (não necessariamente simples) definidos pelo usuário.Como o número de variáveis geralmente é enorme, o *pricing* geralmente é resolvido dinamicamente através de *resource constrained shortest path problems* - RCSP. O VRPSolver permite, através de parametrização, a utilização de vários elementos avançados de BCP, tais como: R1C (*rank-1 cuts*) de memória limitada, enumeração e *strong branching*, por exemplo. Extensivos resultados computacionais têm mostrado que os modelos construídos para o VRPSolver têm um desempenho muito robusto, sendo frequentemente superior aos algoritmos exatos existentes para as instâncias mais difíceis. Esta seção apresenta o modelo do SSCFLP para o VRPSolver. Os elementos utilizados são elencados aqui em uma breve explicação. Aos leitores interessados, o trabalho de Pessoa et al. [2019] apresenta todos os recursos disponíveis em detalhes.

3.2.1. Grafos para Geração de Caminhos

O VRPSolver gera variáveis mapeadas sobre caminhos em grafos que são previamente definidos pelo usuário. Podem ser definidos um ou mais grafos dependendo do modelo. Assim, $G^k = (V^k, A^k) \forall k \in k$ é o conjunto de todos os grafos do problema, sendo $k = 1$ a situação em que apenas um grafo é definido. Para cada grafo k são definidos 1) vértices v_{source} e v_{sink} que definem o início e final do caminho percorrido no grafo, 2) o conjunto R de recursos. Para cada $r \in R$ e $a \in A$, $q_{a,r}$ é o consumo do recurso r no arco a, e $[l_{a,r}, u_{a,r}]$ o seu consumo acumulado. Um caminho em G $p = (v^k_{\text{source}} = v_0, a_1, v_1, \ldots, a_{n-1}, v_{n-1}, a_n, v_n = v^k_{\text{sink}})$ é restringido por

recursos se, para todo $r \in R$ o consumo acumulado do recurso $S^p_{j,r}$ na visita j, $0 \leq j \leq n$, onde $S^p_{0,r} = 0$ e $S^p_{j,r} = max\{l^r_{a,j}, S^p_{j-1,r} + q_{a_j,r}\}$ não excede $u_{a_j,r}$. Assim, P^i é conjunto de todos possíveis caminhos no grafo i é $P = \cup_{k \in K} P^k$. Para todo $a \in A$ e $p \in P$, h^p_a é o número de vezes que o arco a aparece no caminho p.

3.2.2. Formulação e *Mapping*

O modelo pode ser definido pelo usuário como a seguir.Sejam as variáveis x_j, $1 \leq j \leq n_1$ e y_s, $1 \leq s \leq n_2$. As primeiras $\bar{n}_1$ x variáveis $\bar{n}_2$ y são inteiras. As equações (2a) e (2b) definem, respectivamente, a função objetivo e m restrições sobre as variáveis. Para cada variável x_j, $1 \leq j \leq n_1$, $M(x_j) \subseteq A$ define o seu *mapping* dentro de um subconjunto não vazio dos arcos. Ressalta-se aqui que o mesmo arco pode ser mapeado para mais que uma variável x_j. Define-se $M^{-1}(a)$ como $\{j | 1 \leq j \leq n_1; a \in M(x_j)\}$. Como nem todos os arcos necessitam pertencer a algum *mapping*, alguns conjuntos M^{-1} podem ser vazios. Para cada caminho $p \in P$, seja λ_p uma variável inteira não negativa; o coeficiente h^p_a indica quantas vezes a aparece em p. A relac cão entre as variáveis x e λ é dada por (2c). Para cada $k \in K$, L^k e U^k são dados limites inferior e superior em relação ao número de caminhos na solução.

$$\text{Min} \quad \sum_{j=1}^{n_1} c_j x_j + \sum_{s=1}^{n_2} f_s y_s \tag{2a}$$

$$\text{S.t.} \quad \sum_{j=1}^{n_1} \alpha_{ij} x_j + \sum_{s=1}^{n_2} \beta_{is} y_s \geq d_i, \qquad i = 1, \ldots, m, \tag{2b}$$

$$x_j = \sum_{k \in K} \sum_{p \in P^k} \left(\sum_{a \in M(x_j)} h^p_a \right) \lambda_p, \qquad j = 1 \ldots, n_1, \tag{2c}$$

$$L^k \leq \sum_{p \in P^k} \lambda_p \leq U^k, \qquad k \in K, \tag{2d}$$

$$\lambda_p \in \mathbb{Z}_+, \qquad p \in P, \tag{2e}$$

$$x_j \in \mathbb{N},\ y_s \in \mathbb{N} \qquad j = 1, \ldots, \bar{n}_1,\ s = 1, \ldots, \bar{n}_2. \tag{2f}$$

Se as variáveis x foram eliminadas e as restrições de integralidade forem relaxadas, o PL seguinte é obtido:

$$\text{Min} \quad \sum_{k \in K} \sum_{p \in P^k} \left(\sum_{j=1}^{n_1} c_j \sum_{a \in M(j)} h^p_a \right) \lambda_p + \sum_{s=1}^{n_2} f_s y_s \tag{3a}$$

$$\text{S.t.} \quad \sum_{k \in K} \sum_{p \in P^k} \left(\sum_{j=1}^{n_1} \alpha_{ij} \sum_{a \in M(x_j)} h^p_a \right) \lambda_p + \sum_{s=1}^{n_2} \beta_{is} y_s \geq d_i, \qquad i = 1, \ldots, m, \tag{3b}$$

$$L^k \leq \sum_{p \in P^k} \lambda_p \leq U^k, \qquad k \in K, \tag{3c}$$

$$\lambda_p \geq 0, \qquad p \in P. \tag{3d}$$

O problema mestre é (3) resolvido por geração de colunas. Seja π_i, $1 \leq i \leq m$, as variáveis duais das restrições (3b), ν^k_+ and ν^k_-, $k \in K$, as duais das restrições (3c). O custo reduzido de um arco $a \in A$ é:

$$\bar{c}_a = \sum_{j \in M^{-1}(a)} c_j - \sum_{i=1}^{m} \sum_{j \in M^{-1}(a)} \alpha_{ij} \pi_i.$$

O custo reduzido de um caminho $p = (v_0, a_1, v_1, \ldots, a_{n-1}, v_{n-1}, a_n, v_n) \in P^k$ é:

$$\bar{c}(p) = \sum_{j=1}^{n} \bar{c}_{a_j} - \nu_+^k - \nu_-^k.$$

Então, os subproblemas de *pricing* correspondem a encontrar para cada $k \in K$, um caminho $p \in P^k$ com mínimo custo reduzido.

3.2.3. *Packing Sets*

Seja $\mathcal{B} \subset 2^A$ uma coleção de subconjuntos mutuamente disjuntos de A tal que as restrições:

$$\sum_{a \in B} \sum_{p \in P} h_a^p \lambda_p \leq 1, \quad B \in \mathcal{B}, \tag{4}$$

são satisfeitas por pelo menos uma solução ótima (x^*, y^*, λ^*) da formulação (2). Assim, $\mathcal{B}$ define uma coleção de *packing sets*. Um *packing set* pode conter arcos de grafos diferentes e nem todos os arcos em A necessitam pertencer a algum *packing set*. A definição de um $\mathcal{B}$ é parte da modelagem, sendo definida pelo usuário. Para a modelagem do SSCFLP, os *packing sets* são definidos para cada cliente $j \in C$.

3.2.4. *Branching*

O *branching* sobre as variáveis y e x é simples e não muda a estrutura do subproblema de *pricing*. O usuário define sobre qual das variáveis o branching é considerado prioritário. Para o SSCFLP, que possui as variáveis y de alocação e x de localização, a experiência mostra que priorizar o *branching* sobre as variáveis y é mais efetivo. Assim, o *branching* é feito sobre as variáveis x somente se não houverem mais variáveis y a serem analisadas.

3.2.5. Enumeração

Consiste em tentar enumerar em uma tabela todos os caminhos em um certo conjunto P^k que têm potencial de serem parte de uma solução melhor. Depois que a enumeração é feita, o subproblema de *pricing* k pode ser solucionado por inspeção, economizando tempo. Se a enumeração ocorreu para todos os $k \in K$ e o número de caminhos é razoável, o problema inteiro pode ser finalizado por um solver MIP. Neste contexto, a enumeração ocorre dinamicamente enquanto o VRPSolver processa o modelo.

3.2.6. *Rank-1 Cuts*

O VRPSolver generaliza o conceito de *rank-1 cuts*, que são cortes potencialmente fortes, mas cuja inserção no modelo torna o *pricing* mais difícil. A técnica de memória limitada ajuda a mitigar os efeitos negativos da adição de tais cortes. Neste trabalho os R1C são utilizados dinamicamente e adicionalmente aos cortes previamente inseridos no modelo que foram gerados na etapa 1.

3.2.7. O Modelo VRPSolver para o SSCFLP

Dados os conceitos apresentados nas seções anteriores, o modelo VRPSolver para o problema do SSCFLP é apresentado a seguir.

Dados: Conjunto C de clientes; conjunto F de instalações; subconjunto $K \subseteq F$ das instalações não fixadas; subconjuntos $C'_k \subseteq C$ para $k = 1, \ldots, |K|$, dos clientes que são atribuídos a cada instalação do subconjunto K; capacidade da instalação s_k, $k \in K$; demanda do cliente d_j, $j \in C$; custos fixos de abertura das instalações f_k, $k \in K$; custo de atribuição cliente-instalação c_j^i,$i \in F, j \in C$.
Objetivo: Encontrar uma atribuição de clientes a instalações tal que a capacidade total da instalação não seja excedida, minimizando o custo total de atribuição e abertura de instalações.

Modelo: Grafo $G^k = (V^k, A^k)$ para cada instalação $k \in K$, $V^k = \{v_j^k : j = 0, \ldots, |C'_k|\}$, $A^k = \{a_{j+}^k = (v_{j-1}^k, v_j^k), a_{j-}^k = (v_{j-1}^k, v_j^k) : j = 1, \ldots, |C'_k|\}$, $v_{\text{source}}^k = v_0^k$, $v_{\text{sink}}^k = v_{|C'_k|}^k$. $R = R^M = \{1\}$; $q_{a_{j+}^k,1} = d_j^k, q_{a_{j-}^k,1} = 0, j \in C$; $[l_{v_j^k,1}, u_{v_j^k,1}] = [0, s_k]$, $t \in T \cup \{0\}$. Variáveis binárias x_j^k, $j \in C$, $k \in K$. A formulação fica:

$$\text{Min} \quad \sum_{j \in C'_k} \sum_{k \in K} c_j^k x_j^k + \sum_{k \in K} f_k y_k \tag{5a}$$

$$\text{S.t.} \quad \sum_{k \in K} x_j^k = 1, \qquad j \in C; \tag{5b}$$

$L^k = 0$, $U^k = 1$, $k \in K$; $M(x_j^k) = \{a_{j+}^k\}, j \in C, k \in K$. $\mathcal{B} = \cup_{j \in C}\{\{a_{j+}^k : k \in K\}\}$. O branching é realizado sobre as variáveis y e x, nesta ordem de prioridade. Os cortes gerados na etapa 1 do algoritmo, bem como as restrições de integralidade, são inseridas no modelo como restrições adicionais. A figura 3.2.7 apresenta o grafo gerado no modelo para o problema de *pricing*.

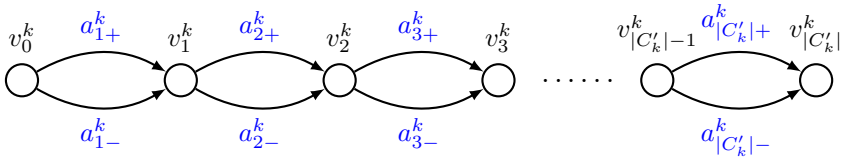

Figura 1: Grafo modelo para o SSCFLP. Os RCSPs correspondem a soluções da mochila binária.

4. Experimentos Computacionais

Nesta seção, a eficiência do algoritmo desenvolvido neste trabalho é testada através de uma análise dos resultados obtidos em um conjunto de instâncias criado no trabalho de Gadegaard et al. [2017] e da comparação com o algoritmo dele, o qual será aqui chamado de $Alg - G$. A implementação deste algoritmo foi realizada utilizando o solver de programação linear do ILOG CPLEX 12.9 (CPLEX) na etapa 1. Todo o código foi implementado em linguagem Julia e os experimentos foram realizados em um laptop Samsung 270E com 4GB RAM e um processador Intel Core i7-4510U 2.00GHz rodando uma versão 64 bit do Ubuntu 19.04.

O conjunto de testes utilizado é composto por 45 instâncias que variam de 50 a 60 instalações e 100 a 300 clientes. Embora numericamente possa ser um conjunto de instâncias com tamanho inferior àquelas encontradas em Yang et al. [2012], onde as instâncias chegam a 800 clientes, possuem estrutura diferente no que concerne aos custos da matriz de atribuição cliente-instalação: a razão entre a capacidade das instalações e os clientes varia de 2 a 5 vezes. O consenso obtido entre as pesquisas na área é que quanto menor a razão entre a capacidade total e a demanda total, maior é o número necessário de abertura de instalações e maior é, por este motivo, o esforço computacional para a resolução do problema. Isto significa que, do ponto de vista computacional, essa instancias são consideradas difíceis.

Além disso, o conjunto de instâncias criado por Gadegaard et al. [2017] é de interesse neste trabalho uma vez que há instâncias que seu algoritmo não foi capaz de resolver na otimalidade num limite de 50000 segundos. O conjunto - chamado de TB4 - está disponível em `https://github.com/SuneGadegaard/SSCFLPsolver`. Para melhor entendimento das tabelas que serão apresentadas aqui, a descrição das abreviaturas encontradas nos cabeçalhos são apresentadas na Tabela 1.

Descrição das Abreviações	
ID	Identificação do Problema
$\|I\|x\|J\|$	Número de Clientes e Instalações na instância
r	Taxa entre capacidade e demanda totais
% TE-1 TE-2	O percentual de tempo gasto em cada uma das etapas do algoritmo
$\underline{Z}$, Z^*	Denotam, respectivamente, o limite inferior encontrado no nó raiz na etapa 2 do algoritmo, e a solução ótima
LB_{gap}	$(Z^* - \underline{Z})/\underline{Z}$
LB_{gapG}	O valor do gap para o algoritmo estado-da-arte que está sendo comparado
CPU	Tempo de cpu usado para resolver cada instância
CPU_G	Tempo de cpu usado para resolver cada instância no algoritmo estado-da-arte

Tabela 1: Descrição das abreviações dos cabeçalhos das tabelas

ID	$\|I\|x\|J\|$	r
n1 - n5		2
n6 - n10	$\|50\|x\|100\|$	3
n11 - n15		5
n16 - n20		2
n21 - n25	$\|50\|x\|200\|$	3
n26 - n30		5
n31 - n35		2
n36 - n40	$\|60\|x\|300\|$	3
n41 - n45		5

Tabela 2: Características das instâncias

4.1. Resultados

A Tabela 1 apresenta os resultados obtidos pelo algoritmo desenvolvido neste trabalho. Infelizmente, devido ao atual contexto vivido da pandemia, não houveram condições para desempenhar os testes em laboratório específico, o que dificultou a testagem completa do corpo de instâncias. Assim, grande parte das instâncias não foi processada devido ao desempenho da memória da máquina utilizada para os testes.

No entanto, algumas análises já podem ser realizadas com os dados parciais obtidos. O conjunto TB4 pode ser dividido conforme a Tabela 2 apresentada. A cada 15 instâncias o número de clientes ou de potenciais instalações é alterado e, dentro deste subconjunto, a razão entre a capacidade total e a demanda total cresce de 2 a 5 vezes.

$Alg - G$ consome a maior parte do processamento das instâncias com maior tempo de processamento na fase 3 do algoritmo, que é a fase do *cut-and-solve*. Basicamente, o *cut-and-solve* é semelhante ao *branch-and-bound*, mas com apenas duas direções de *branching*, ou seja, o *cut-and-solve* ocorre sobre um conjunto de variáveis e não sobre uma variável por vez. Os nós a direita

da árvore de *branching* são associados ao problema denso - os quais geram o valor $\underline{Z}$ associado, e os à esquerda são associados ao problema esparso, gerando o $\bar{Z}$ da solução em cada iterac cão. O processo itera até que possa ser alcançada a otimalidade. De modo a acelerar o processo, os cortes gerados nas fases anteriores são inseridos no problema, mais os *piercing cuts*, que são cortes gerados sobre um conjunto de varivéis de localização em cada iteração dos problemas denso ou esparso que geram potencial de melhorar a solução.

Já no algoritmo desenvolvido neste trabalho, a etapa 2 utilizou *branch-and-cut-and-price* no VRPSolver e, adicionalmente, todos os cortes criados na etapa 1 são alocados ao modelo na etapa 2. Durante o processamento, o VRPSolver utiliza técnicas como adicionar os *rank-1 cuts* desde o nó raiz de maneira a aumentar o limite inferior da solução e reduzir o tempo de processamento. Além disso, tenta enumerar em cada grafo criado para as variáveis de localização não fixadas os possíveis caminhos e, dado o tamanho da solução, possibilita o cálculo do *pricing* por inspeção.

ID	Tamanho	r	$\underline{Z}$	LB_{gap}	LB_{gapG}	Z^*	$\%TE1$	$\%TE2$	CPU	CPU_G
n1			18294	**0,00**	0,01	18294	46,64	53,36	**43,8**	616,74
n2			19688	**0,00**	0,06	19688	40,15	59,85	**67,67**	3705,52
n3		2	19075	**0,00**	0,01	19075	48,56	51,44	**19,17**	192,32
n4			18619,4	**0,00**	0,01	18620	44,64	55,35	**31,74**	92,10
n5			18498,6	0,02	0,02	18502	28,14	71,86	**76,42**	788,01
n6			16937,3	0,06	0,03	16948	3,64	96,36	**569,09**	3788,94
n7			15062,1	0,01	0,01	15063	32,34	67,66	67,41	32,37
n8	$50X100$	3	15105,7	**0,01**	0,02	15107	10,28	89,72	**159,68**	170,56
n9			14343,1	0,03	0,02	14347	16,00	84,00	245,26	89,11
n10			14811,8	**0,01**	0,03	14813	10,33	89,67	**156,31**	476,28
n11			12070,7	0,01	0,01	12072	2,66	97,34	1060,24	10,36
n12			11890,4	0,06	0,03	11898	2,65	97,35	797,78	46,46
n13		5	11117,2	0,07	0,00	11125	7,82	92,18	532,68	8,58
n14			11816,33	0,01	0,01	11817	46,11	53,89	64,63	10,05
n15			11489	**0,00**	0,02	11489	57,7	42,3	28,09	8,68
n16			25986,7	0,09	0,01	25992	1,53	98,46	2362,38	2267,62
n17			25866,12	0.01	0,01	25868	-	-	>	122,87
n18		2	26929	0,04	0,01	26930	5,46	94,54	643,24	869,75
n19			25951.47	0,11	0,11	25980	-	-	>	14790,36
n20			25326,77	0,26	0,26	25390	-	-	>	21308,44
n21			20695.6	0,03	0,02	20701	-	-	>	1397,86
n22			22019,6	**0,006**	0,01	22021	3,61	96,39	**1370,84**	19588,60
n23	50 X 200	3	20037.5	**0,002**	0,01	20038	-	-	>	7,42
n24			-	-	0,00	20595	-	-	>	28,08
n25			-	-	0,00	21168	-	-	>	87,55
n26			-	-	0,00	16659	-	-	>	11,54
n27			-	-	0,01	16138	-	-	>	18,92
n28		5	-	-	0,00	17755	-	-	>	15,37
n29			-	-	0,01	15858	-	-	>	12,08
n30			-	-	0,00	16884	-	-	>	13,68

Tabela 3: Desempenho do Algoritmo nas instâncias TB4. Os valores em negrito são correspondentes aos tempos de rodada nas instâncias e os gaps em que o algoritmo deste trabalho superou $Alg-G$. As instâncias em que foi possível ver o LB_{gap} antes do estouro de memória estão com as colunas da tabela preenchidas

Na Tabela 3 estão disponíveis os principais resultados, em média, dos testes realizados.

Para as primeiras instâncias (n1 - n10) um *gap* extremamente pequeno foi observado. Em alguns casos ele foi igual a zero, dado que os cortes gerados na primeira etapa fortaleceram o problema e os *rank-1 cuts* foram muito efetivos neste caso, bem como a estratégia de fixação de variáveis na etapa 1. Assim, os valores de LB_{gap} foram 0,00 neste caso pois o valor final encontrado pelo mestre na raiz antes de tentar fazer o *branching* foi igual ao valor da solução otima.

Os três subconjuntos n1- n5, n6-n10 e n11-n15, em que todas foram resolvidas, apresentam um comportamento oposto ao desempenho das mesmas em [Gadegaard et al., 2017].A tendência para $Alg-G$ foi aumentar o tempo de rodada à medida em que a razão entre a capacidade total e a demanda total diminuiu, seguindo o que tem sido observado nos estudos do SSCFLP até então. Isso porque nestes casos a alocação de instalações aumentou, aumentando também o tamanho do problema esparso para o *cut-and-solve*. No caso do algoritmo deste trabalho, a média de tempo das instâncias n1-n5 foi muito melhor, inclusive, mais de 20 vezes menor em comparação a $Alg-G$: 47,76s contra 1078,94s. Isso pode ser atribuído à efetividade dos cortes do VRPSolver para as instâncias com estas particularidades, bem como ao alto desempenho do solver quando o *gap* alcançado no nó raiz é baixo. No entanto, à medida em que r aumenta, nota-se um aumento no LB_{gap} que aumenta substancialmente o tempo de processamento. Dessa forma, deve ser avaliada a efetividade dos *rank-1 cuts* em conjunção a outros elementos, como por exemplo a parametrização da enumeração. Outra questãoé que, em instâncias maiores por exemplo, os cortes dessa natureza podem não ser tão efetivos pois aumentam a dificuldade do *pricing*, sendo importante investigar outros tipos de cortes que podem ser introduzidos nesta situação. A grande vantagem evidenciada aqui é que, para as instâncias consideradas muito difíceis para o desempenho de $Alg-G$, aquelas com *gap* equivalente ou um pouco pior no nosso caso tiveram um desempenho muito superior em termos de tempo de processamento(ver n-5 e principalmente n-22, por exemplo, em que o tempo de processamento de n-22 foi 14 vezes menor).

5. Conclusões

Este trabalho implementou um algoritmo para a resolução exata do SSCFLP, sendo dividido em 2 etapas: a primeira insere cortes e fixa variáveis para o modelo que é criado no VRPSolver, e a segunda resolve esse mesmo modelo de maneira exata através de uma estrutura *branch-and-cut-and-price*. A principal contribuição deste trabalho é comparar o desempenho do estado-da-arte para algoritmos dedicados ao problema com um solver genérico, e verificar como o algoritmo se comportou no caso das instâncias mais difíceis. Observou-se que o desempenho do algoritmo é excelente justamente onde o algoritmo comparado possui uma de suas fraquezas. Os gaps foram relativamente muito baixos nas instâncias processadas, e para gaps comparativamente iguais nas instâncias mais dificeis o algoritmo desenvolvido aqui teve um desempenho muito superior.

No entanto, deve ser observado que devido ao estouro de memória não foi possível verificar o desempenho em termos de tempo de processamento, porém o indicativo é que para estes casos, o *pricing* pode ter pesado bastante, comprometendo a memória. Mais trabalho deve ser realizado neste sentido. Assim, direções futuras de pesquisa envolvem investigar a determinação de novos possíveis cortes na etapa 1 para melhorar ainda mais a fixação das variáveis e o limite inferior da solução, e avaliar o ganho obtido nos *rank-1 cuts* com a implementação dos grafos para geração dos caminhos mais curtos considerando-se uma matriz de distâncias entre os clientes, criada a partir da matriz de custos de atribuição.

Referências

Avella, P., Boccia, M., e Vasilyev, I. (2010). A computational study of exact knapsack separation for the generalized assignment problem. *Computational Optimization and Applications*, 45(3):

543–555. URL https://EconPapers.repec.org/RePEc:spr:coopap:v:45:y:2010:i:3:p:543-555.

Boyd, E. A. (1993). Generating fenchel cutting planes for knapsack polyhedra. *SIAM Journal on Optimization*, 3(4):734–750. URL https://doi.org/10.1137/0803038.

Ceselli, A. e Righini, G. (2005). A branch-and-price algorithm for the capacitated p-median problem. *Networks*, 45(3):125–142. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/net.20059.

Gadegaard, S., Klose, A., e Nielsen, L. (2017). An improved cut-and-solve algorithm for the single-source capacitated facility location problem. *EURO Journal on Computational Optimization*, 6.

Holmberg, K., RÃ¶nnqvist, M., e Yuan, D. (1999). An exact algorithm for the capacitated facility location problems with single sourcing. *European Journal of Operational Research*, 113(3):544 – 559. ISSN 0377-2217. URL http://www.sciencedirect.com/science/article/pii/S0377221798000083.

Kaparis, K. e Letchford, A. N. (2010). Separation algorithms for 0-1 knapsack polytopes. *Mathematical Programming*, 124(1):69 – 91. ISSN 1436-4646. URL https://link.springer.com/article/10.1007%2Fs10107-010-0359-5#article-info.

Karp, R. M. (1972). *Reducibility among Combinatorial Problems*, p. 85–103. Springer US, Boston, MA. ISBN 978-1-4684-2001-2. URL https://doi.org/10.1007/978-1-4684-2001-2_9.

Pessoa, A., Sadykov, R., Uchoa, E., e Vanderbeck, F. (2019). A generic exact solver for vehicle routing and related problems. In Lodi, A. e Nagarajan, V., editors, *Integer Programming and Combinatorial Optimization*, p. 354–369, Cham. Springer International Publishing. ISBN 978-3-030-17953-3.

Yang, Z., Chu, F., e Chen, H. (2012). A cut-and-solve based algorithm for the single-source capacitated facility location problem. *European Journal of Operational Research*, 221(3):521 – 532. ISSN 0377-2217. URL http://www.sciencedirect.com/science/article/pii/S0377221712002731.