Projeto LABTEC: Alocação de Custos de Serviços de Transmissão [1]

Relatório Técnico 97.2 - Dezembro 1997

Celso Carneiro Ribeiro, Marcus V. Poggi de Aragão,
Andréa Iabrudi Tavares, Renata Machado Aiex,
Eduardo Uchoa Barbosa

Departamento de Informática
Pontifícia Universidade Católica do Rio de Janeiro
Rua Marquês de São Vicente 225
Rio de Janeiro 22453-900, Brasil

1 Programas Lineares com Número Exponencial de Restrições

Problemas de programação linear (PLs) com um número super-polinomial de restrições ou variáveis ocorrem freqüentemente na literatura e na prática. Uma situação comum onde aparecem é na modelagem de uma propriedade que deve ser satisfeita por todos os subconjuntos de elementos de um determinado conjunto. Como há um número exponencial de tais subconjuntos, há um número exponencial de restrições. Qualquer PL com número exponencial de variáveis pode ser resolvido através do seu dual, que será um PL com número exponencial de restrições. Assim sendo, no restante deste texto serão tratados apenas os problemas com número exponencial de restrições, sem perda de generalidade.

Um aspecto muito importante, que permite que essas formulações sejam de interesse prático, é que um número de restrições exponencial não implica, necessariamente, em que a complexidade de resolução do PL resultante seja exponencial. Considera-se assim a classe de PLs que possuam uma forma sistemática para definir implicitamente todas suas restrições,

[1] Este relatório contém desenvolvimentos algorítmicos e resultados computacionais que farão parte da tese de doutorado de Andréa I. Tavares e do exame de qualificação de Renata M. Aiex, doutorandas do Departamento de Informática da PUC-Rio.

sem que seja necessário armazená-las na memória, propriedade esta que é verificada por um grande número de problemas. Grötschel, Lovasz e Schrijver [5] mostraram que se existir um algoritmo polinomial para identificar restrições violadas dentre aquelas implicitamente descritas, mas ainda não enumeradas, o PL também pode ser resolvido em tempo polinomial.

Qualquer PL pode ser formulado na forma canônica max $cx : Ax \leq b$, onde c, A e b são matrizes com as dimensões apropriadas. O chamado *problema de otimização* associado consiste em resolvê-lo encontrando uma solução x^* ótima ou provando que não há solução viável. Já o *problema de separação* associado a esse PL consiste em, dado um vetor $\bar{x}$ qualquer, verificar se $\bar{x}$ é uma solução viável, ou seja, se satisfaz $A\bar{x} \leq b$; caso contrário, deve ser encontrada a restrição mais violada, isto é, uma linha a_i da matriz A que maximize $a_i\bar{x} - b_i$. O teorema de Grötschel, Lovasz e Schrijver [5] garante então que:

Teorema 1 *Os problemas de otimização e de separação têm complexidade equivalente, i.e. se um deles for polinomial, o outro também o será, e se um deles for NP-difícil, o outro também o será.*

Caso o problema de separação seja polinomial, a prova do teorema 1 fornece um algoritmo garantidamente polinomial para resolver um PL com um número exponencial de restrições, que entretanto não tem utilidade prática por ser uma variante do algoritmo do elipsóide de Khachian [7]. Normalmente, usa-se um algoritmo muito mais simples, sem garantia de polinomialidade, mas que quase sempre é eficiente. Assim, resolve-se inicialmente o chamado *PL restrito* max $cx : A'x \leq b'$, onde A' e b' representam um subconjunto das restrições, obtendo-se uma solução $\bar{x}$. Resolve-se, então, o problema de separação. Se $\bar{x}$ for uma solução viável para o PL completo max $cx : Ax \leq b$, então $\bar{x}$ é também uma solução ótima desse PL, isto é, $x^* = \bar{x}$. Caso contrário, adiciona-se ao PL restrito uma restrição violada por $\bar{x}$ e resolve-se o novo PL restrito com a nova restrição adicional. O processo é repetido até que $\bar{x}$ seja uma solução ótima do PL completo. Freqüentemente, o número de iterações realizadas é relativamente baixo, crescendo de forma aproximadamente proporcional ao tamanho do problema. Ao se reotimizar os PLs restritos é recomendável usar o algoritmo dual Simplex, pois ele é capaz de reutilizar a base da solução do problema anterior para acelerar significativamente a busca da nova solução.

Infelizmente, na maioria dos modelos com número exponencial de restrições, não se conhecem algoritmos de separação polinomiais e, em muitos casos, a separação é um problema NP-difícil. Nestes casos, este enfoque de resolução pode ser proibitivo. Isto se deve à necessidade de se resolver de forma exata um problema de separação computacionalmente difícil a cada iteração, o que pode exigir um enorme esforço computacional. Apesar dessas dificuldades, existem alternativas que às vezes permitem a resolução de modelos com número exponencial de restrições, mesmo quando a separação é NP-difícil.

Uma destas alternativas é usar a *separação heurística*: ao invés de, a cada iteração, encontrar-se a restrição mais violada (*separação exata*), o que pode ser computacionalmente caro, usam-se heurísticas para tentar encontrar uma restrição bastante violada, sem garantir que seja a mais violada. Quando a heurística não for capaz de achar uma restrição violada, deve-se então usar a separação exata, pois apenas ela é capaz de garantir que realmente não há restrições violadas, o que permite identificar o término do algoritmo. O objetivo da separação heurística é tentar reduzir o número de separações exatas, que são o gargalo do algoritmo, e assim viabilizar a solução do PL.

Uma questão fundamental ao se usar esse esquema é a qualidade da separação heurística empregada, ou seja, o quanto esta separação é capaz de se aproximar da separação exata. Se a qualidade da heurística for baixa, não apenas muitas separações exatas podem ser necessárias, mas também o número de iterações do algoritmo pode crescer muito, tornando-o ineficiente. Por outro lado, heurísticas de alta qualidade são computacionalmente muito caras, às vezes quase tanto quanto a separação exata. Dessa forma, torna-se necessário encontrar um balanço adequado entre a qualidade e o custo computacional da separação usada, o que normalmente é feito de modo experimental.

Este relatório descreve os passos que foram seguidos para a resolução numérica exata do Problema de Alocação de Custos de Sistemas de Trasmissão (PACST), segundo um modelo descrito em Campodónico e Pereira [2] que é revisto na próxima seção. Esse modelo corresponde a um PL com um número exponencial de restrições, cujo problema de separação associado é NP-difícil. A seção 3 descreve os algoritmos para resolução do problema de separação, apresentando uma abordagem heurística e uma exata, assim como diferentes formas de aplicá-las. O estudo de um caso realista baseado no sistema brasileiro é objeto da seção 4, onde são apresentados resultados computacionais com diferentes variantes dos algoritmos de solução. A seção 5 é dedicada a uma crítica do modelo proposto, com base na interpretação dos resultados obtidos. Finalmente, apresenta-se na seção 6 uma análise das etapas de modelagem e resolução do PACST. Um apêndice sobre o estado da arte com respeito à resolução do problema de separação exato completa este documento.

2 Alocação de Custos de Sistemas de Transmissão

Seja $G = (V, E)$ o grafo não-orientado associado a um sistema de transmissão de energia elétrica. Os nós da rede representam pontos de geração, consumo ou conexão. Esses nós são ligados por arestas que representam linhas de transmissão. Cada aresta possui uma capacidade já instalada conhecida (não-nula para as arestas já existentes, nula para as arestas ainda não existentes mas que podem ser construídas), mas pode ser expandida de forma a aumentar a capacidade da conexão. Para cada aresta passível de ser expandida, são conhecidos o acréscimo de capacidade fornecido por cada expansão, o custo de cada expansão e o número máximo de expansões que podem ser realizadas.

O Problema de Expansão de Redes Capacitadas (PERC) consiste em determinar o conjunto de arestas que devem ter sua capacidade expandida, de maneira a tornar a rede viável ao menor custo possível. Por rede viável, entende-se uma rede que supra as necessidades de geração e consumo dos nós. Supondo-se que esta rede será utilizada por um consórcio de K agentes, onde cada agente gera energia em um certo nó da rede e fornece a mesma quantidade para outro nó, o Problema de Alocação de Custos de Serviços de Transmissão (PACST) consiste em obter uma distribuição do custo de expansão da rede entre os agentes do consórcio, satisfazendo as seguintes propriedades:

- A soma dos valores cobrados de cada agente deve ser igual ao custo do serviço.
- A soma dos valores cobrados de cada subgrupo de agentes não deve exceder o custo deste subgrupo formar um consórcio separado.

Campodónico e Pereira [2] formularam este problema como um PL com um número exponencial de restrições. Sejam Z^* o custo mínimo da expansão necessária para construir uma rede viável satisfazendo as ofertas e demandas do consórcio formado pelos K agentes, Z_S o custo de expansão necessário para atender um consórcio separado formado por um subconjunto qualquer de agentes $S \subset \{1, \ldots, K\}$ e θ_k o custo alocado ao agente $k \in \{1, \ldots, K\}$. Tem-se então o modelo:

$$
(\text{PACST}) \begin{cases}
\text{Min} & \Delta & & (PACST.0) \\
\text{Sujeito a} & \displaystyle\sum_{k=1}^{K} \theta_k + \Delta = Z^* & & (PACST.1) \\
& \displaystyle\sum_{k \in S} \theta_k \leq Z_S & \forall S \subset \{1, \ldots, K\} \quad \text{e} \quad S \neq \emptyset & (PACST.2) \\
& \theta_k \geq 0 & k = 1, \ldots, K & (PACST.3) \\
& \Delta \geq 0 & & (PACST.4)
\end{cases}
$$

A função objetivo Δ pode ser vista como uma medida da inviabilidade do problema e um valor estritamente positivo dessa variável na solução ótima indica que não foi possível satisfazer as duas primeiras propriedades simultaneamente. A restrição (PACST.1) força a verificação da primeira propriedade acima, já que o mínimo é atingido com $\Delta^* = 0$, caso isto seja viável. As restrições (PACST.2) impõem a segunda propriedade, i.e. qualquer subconjunto próprio de agentes tem custo total alocado inferior ao de sua expansão isolada. Naturalmente, os custos são todos positivos.

O valor de Z^* pode ser obtido resolvendo-se um Problema de Expansão de Redes Capacitadas (PERC). Seja $b_i^0 > 0$ (resp. $b_i^0 < 0$) a demanda (resp. oferta) básica da rede, associada a cada nó $i \in V$. Sejam ainda as demandas b_i^k adicionais associadas exclusivamente a cada um dos $k = 1, \ldots, K$ agentes em cada nó $i \in V$, com as mesmas convenções de sinal. Cada aresta (i, j) possui quatro características: $\overline{c}_{ij}$, sua capacidade instalada: c_{ij}, o acréscimo de capacidade devido a cada expansão; d_{ij}, o custo de uma expansão; e u_{ij}, o número máximo de expansões. A cada aresta (i, j) estão associadas duas variáveis: uma variável contínua f_{ij} representando o fluxo que a percorre (se $f_{ij} > 0$ o fluxo vai de i para j, caso contrário de j para i) e uma variável inteira x_{ij} indicando o número de expansões realizadas na aresta. Tem-se então o problema modelado da seguinte forma:

$$
(\text{PERC})\left\{\begin{array}{llll}
\text{Min} & \displaystyle\sum_{(i,j)\in E} d_{ij}x_{ij} & & (PERC.0)\\[2ex]
\text{Sujeito a} & \displaystyle\sum_{j:(j,i)\in E} f_{ji} - \sum_{j:(i,j)\in E} f_{ij} = b_i^0 + \sum_{k=1}^{K} b_i^k & \forall i \in V & (PERC.1)\\
 & |f_{ij}| - c_{ij}x_{ij} \le \bar{c}_{ij} & \forall (i,j)\in E & (PERC.2)\\
 & x_{ij} \le u_{ij} & \forall (i,j)\in E & (PERC.3)\\
 & x_{ij} \in \mathbb{N} & \forall (i,j)\in E & (PERC.4)
\end{array}\right.
$$

Os valores de $Z_S, \forall S \subset \{1,\ldots,K\}$ podem ser obtidos pela resolução de problemas similares, substituindo-se as demandas (ou ofertas) b_i pelo total associado a cada subconjunto S de agentes. O PERC é uma generalização do problema conhecido por *Fixed Charge Network Flow* (FCNF), descrito no apêndice A. Como o FCNF é NP-difícil, PERC também é NP-difícil. Assim sendo, para enumerar-se todas as restrições de PACST (uma para cada um dos $2^K - 1$ subconjuntos de agentes), é preciso resolver um número exponencial de problemas NP-difíceis, o que torna sua abordagem direta inviável. Ou seja, a resolução do PACST tem como dificuldades inerentes não somente o número exponencial de restrições, mas também a particularidade de que gerar qualquer de suas restrições corresponde a um problema NP-difícil.

A resolução computacional do PACST através do tipo de enfoque descrito na seção 1 deste texto exige a definição de um problema de separação. Este tem como solução ótima a restrição do tipo $PACST.2$ mais violada pela solução corrente $\bar{\theta}$, onde $\bar{\theta}_k$ é o custo alocado ao agente $k = 1,\ldots,K$ na solução corrente, obtido resolvendo-se um PL restrito do PACST. Este problema de separação é formulado em Campodónico e Pereira [2] como o seguinte programa linear misto:

$$
(\text{Sep-PACST})\left\{\begin{array}{llll}
\text{Min} & \displaystyle\sum_{(i,j)\in E} d_{ij}x_{ij} - \sum_{k=1}^{K} \bar{\theta}_k \lambda_k & & (Sep.0)\\[2ex]
\text{Sujeito a} & \displaystyle\sum_{j:(j,i)\in E} f_{ji} - \sum_{j:(i,j)\in E} f_{ij} = b_i^0 + \sum_{k=1}^{K} b_i^k \lambda_k & \forall i \in V & (Sep.1)\\
 & |f_{ij}| - c_{ij}x_{ij} \le \bar{c}_{ij} & \forall (i,j)\in E & (Sep.2)\\
 & x_{ij} \le u_{ij} & \forall (i,j)\in E & (Sep.3)\\
 & x_{ij} \in \mathbb{N} & \forall (i,j)\in E & (Sep.4)\\
 & \lambda_k \in \{0,1\} & k = 1,\ldots,K & (Sep.5)
\end{array}\right.
$$

As variáveis binárias λ_k estão associadas aos agentes $k = 1, \ldots, K$. Qualquer atribuição de valores 0 ou 1 a essas variáveis especifica um subconjunto $S = \{k : \lambda_k = 1\}$ de agentes. Para cada uma dessas possíveis atribuições, o problema de separação se reduz em verificar a diferença entre o custo ótimo de expansão do PERC definido pelo subconjunto de agentes S e o custo total cobrado dos agentes em S de acordo com a alocação corrente. Se alguma dessas diferenças for negativa, a restrição $(PACST.2)$ associada a S estará violada. O subconjunto de agentes para o qual diferença e' mais negativa está associado à restrição mais violada.

O problema de separação Sep-PACST é uma generalização do PERC e, portanto, também é NP-difícil. No apêndice A, mostra-se que tanto o PERC quanto o problema Sep-PACST podem ser reformulados como um FCNF. Assim sendo, pelo menos em princípio, sua resolução poderia beneficiar-se de técnicas já testadas nesse último problema. Observa-se que a resolução de uma instância de grande porte do problema PACST exige um grande esforço computacional, o que torna inviável a resolução exata de Sep-PACST a cada iteração onde o problema de separação deve ser resolvido. Portanto, a eficiência da resolução do PACST dependerá da qualidade da abordagem heurística aplicada ao problema de separação. Este é o assunto da próxima seção.

3 Resolução do Subproblema

Supondo-se conhecidas todas suas restrições, conforme discutido na seção 1, o problema PACST é resolvido por um algoritmo de programação linear. Neste caso, o programa linear considera apenas um subconjunto das restrições, determinando-se através da resolução do subproblema se este subconjunto é suficiente ou não para garantir a otimalidade da solução encontrada. Chama-se este programa linear de problema restrito. Chama-se de *corte* qualquer restrição violada identificada e adicionada ao problema restrito. Chega-se então ao algoritmo para a resolução do PACST:

Algoritmo `Resolve-PACST`:

1. Obter uma alocação de custos viável $\overline{\theta}$ para o programa linear restrito (definido apenas pelo subconjunto corrente de restrições já identificadas), utilizando, por exemplo, o algoritmo Simplex.

2. Resolver o subproblema Sep-PACST. Se a resolução deste problema de separação encontrou uma restrição violada, acrescenta-se este corte ao subconjunto corrente de restrições, obtendo-se um novo programa linear restrito com uma restrição a mais em relação ao anterior.

3. Senão, a alocação corrente de custos para o problema restrito também é viável para o problema linear completo e o algoritmo termina com uma solução para PACST.

Como discutido na seção 1, para que este procedimento funcione, basta ser possível encontrar um corte sempre que este existir. Assim sendo, é possível optar-se por uma abordagem aproximada para a geração de cortes (separação heurística), que permita tratar a dificuldade computacional do subproblema de separação. Esta relaxação elimina a necessidade de encontrar-se o corte com violação máxima. Dada uma alocação de custos $\bar{\theta}$, procura-se *qualquer* corte, ou seja, λ e x viáveis para o problema Sep-PACST satisfazendo

$$\sum_{(i,j)\in E} d_{ij}x_{ij} < \sum_{k=1}^{K} \bar{\theta}_k \lambda_k,$$

ou seja, uma solução cujo valor da função objetivo Sep-PACST seja negativo. Quando a separação é feita de forma exata através da solução de Sep-PACST por um algoritmo de *branch-and-bound*, essa relaxação equivale a suspender a busca assim que uma solução de custo negativo é encontrada. Economiza-se a exploração do restante da árvore, que seria necessária para obtenção da restrição mais violada. No restante deste texto, referencia-se por `Exato-Mínimo` a separação exata feita por um algoritmo de *branch-and-bound* para achar a restrição mais violada, e por `Exato-Primeiro` a separação feita por um *branch-and-bound* relaxado que termina ao encontrar a primeira restrição violada. O algoritmo `Exato-Primeiro` também pode gerar mais de um corte, bastando que seja suspenso assim que um certo número de soluções de custo negativo sejam encontradas ou que a árvore já tenha sido toda explorada.

Essa relaxação pode ter efeito na convergência do algoritmo (número de resoluções do subproblema Sep-PACST necessário para obtenção de uma solução viável), já que a qualidade do corte gerado pode não ser boa. Por outro lado, pode diminuir significativamente o tempo necessário para se obter um novo corte.

Como segunda forma de acelerar a solução do problema de separação, foi elaborada uma heurística para procurar um corte (separação heurística), chegando-se então a uma nova versão para o algoritmo de resolução do PACST:

Algoritmo `Resolve-PACST-Acelerado`:

1. Obter uma alocação de custos viável $\overline{\theta}$ para o programa linear restrito (i.e. que contém apenas o subconjunto corrente de restriçõesjá identificadas), utilizando por exemplo o algoritmo Simplex.

2. Resolver o subproblema Sep-PACST através de uma separação heurística. Se a resolução deste problema de separação encontrou uma (ou mais) restrição violada, acrescenta-se este corte ao subconjunto corrente de restrições, obtendo-se um novo programa linear restrito com uma restrição a mais em relação ao anterior.

3. Senão, aplicar a separação exata, através dos algoritmos `Exato-Primeiro` ou `Exato-Mínimo`. Da mesma forma que na situação anterior, se a resolução deste problema de separação encontrou uma restrição violada, acrescenta-se este corte ao subconjunto corrente de restrições, obtendo-se um novo programa linear restrito com uma restrição a mais em relação ao anterior.

4. Senão, a alocação de custos corrente para o problema restrito também é viável para o problema linear completo e o algoritmo termina com uma solução para PACST.

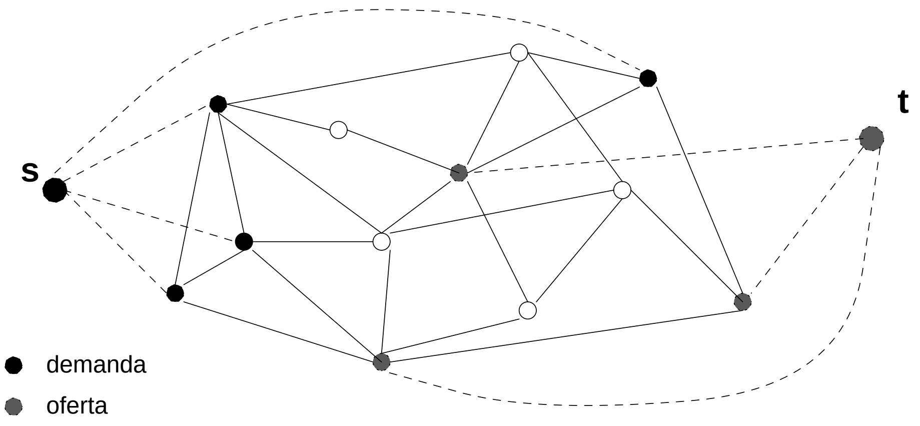


Figura 1: Transformações na rede original: super-fonte s e super-sumidouro t

O algoritmo que realiza a separação heurística se compõe de duas partes distintas, que são combinadas até que se encontre um corte:

- Uma *busca local* no espaço dos subconjuntos de agentes, para determinar um subconjunto de agentes representado por um vetor característico λ ao qual poderá estar associado um corte.
- Para cada subconjunto de agentes investigado pela parte acima, uma heurística para *construção de uma rede viável* (problema PERC) capaz de atender às demandas dos agentes selecionados, que determina uma solução viável x de Sep-PACST associada ao vetor λ. Essa heurística consiste de uma fase construtiva gulosa, que obtém uma solução viável, seguida de uma busca local, que tenta encontrar uma solução de menor custo.

3.1 Construção de uma Rede Viável

Seja a rede cujas demandas são definidas pelo conjunto de agentes $\{k : k = 1, \ldots, K \text{e} \lambda_k = 1\}$, onde λ é o vetor que indica os agentes presentes no sistema. Define-se o vetor c' como o de capacidades instaladas em cada arco da rede ($c'_{ij} = \overline{c}_{ij} + c_{ij}x_{ij}$). A viabilidade de uma solução x é testada através de um problema de fluxo máximo na rede com capacidades c' nos arcos e com a introdução de nós "super-fonte" s e "super-sumidouro" t. Para todo $i \in V$, se $b_i < 0$, então insere-se um arco de s para i com $c'_{si} = -b_i$. Se $b_i > 0$, é inserido um arco de i para t com $c'_{it} = b_i$. Na rede modificada ilustrada na Figura 1, o maior fluxo possível de s para t é $F_{viavel} = \sum_{i:b_i>0} b_i$, dada a capacidade das arcos que saem de s. Uma solução x será viável se o fluxo máximo F_{max} na rede for igual a F_{viavel}. O modelo para o problema de fluxo máximo é:

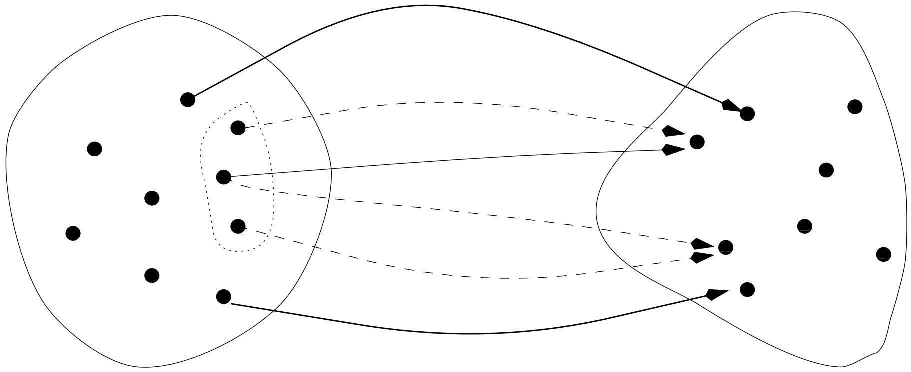


Figura 2: Corte mínimo, arcos candidatos e nós de fronteira

$$
(\text{PFM}) \begin{cases}
F_{max} = \text{Max} \quad F & & (PFM.0) \\
\text{Sujeito a} \quad \sum_{j:(j,i)\in E} f_{ji} - \sum_{j:(i,j)\in E} f_{ij} = 0 & \forall i \in V,\ i \neq s,t & (PFM.1) \\
\sum_{i:(s,i)\in E} f_{si} = F & & (PFM.2) \\
\mid f_{ij} \mid \leq c'_{ij} & \forall (i,j) \in E & (PFM.3) \\
F \geq 0 & & (PFM.4)
\end{cases}
$$

Heurística Construtiva Gulosa: A heurística construtiva começa com uma solução não necessariamente viável $x = 0$ e vai replicando arcos (incrementando o valor de cada variável x_{ij}) até obter uma solução viável. Para a rede inicial, considerando-se apenas as capacidades inicialmente instaladas sem nenhuma replicação, obtém-se o valor de F_{max}, resolvendo-se o problema PFM. Se $F_{max} = F_{viavel}$, então $x = 0$ é uma solução viável ótima para PERC. Caso contrário ($F_{max} < F_{viavel}$), sabe-se pela teoria de fluxos em redes que existe um corte de capacidade F_{max} na rede (c.f. por exemplo [1]). Ou seja, existe uma partição de V em (S,T), com $s \in S$, $t \in T$, tal que o conjunto de arcos $E_{ST} = \{(i,j) \in E : i \in S, j \in T\}$ satisfaz

$$\sum_{(i,j)\in E_{ST}} c'_{ij} = F_{max}.$$

Na Figura 2, um corte E_{ST} é representado pelos seus arcos. Para que o fluxo máximo da rede aumente, é essencial que a *capacidade do corte* (isto é, a capacidade total instalada nas linhas que definem o corte) E_{ST} seja aumentada. Isso só é possível através da replicação de alguns arcos. Define-se o conjunto E_F de arcos de fronteira como aquele formado pelos arcos do corte que podem ser replicados, isto é:

$$E_F = \{(i,j) \in E_{ST} : x_{ij} < u_{ij}\}.$$

O conjunto E_F está representado na Figura 2 pelos arcos pontilhados. Resta então determinar quais arcos de E_F serão replicados e quantas vezes. De modo a tornar a rede viável, é necessário aumentar a capacidade do corte E_{ST} pelo menos da quantidade $F_{faltante} = F_{viavel} - F_{max}$. Adota-se como objetivo local achar uma atribuição das capacidades x que implique no menor custo possível e possa levar a uma rede viável. Essa é uma abordagem gulosa, no sentido de que tenta-se gastar o menos possível localmente, sem considerar-se diretamente o aspecto global da minimização dos custos.

Define-se l_{ij} como o número máximo possível de replicações para um arco, a fim de aumentar a capacidade do corte mínimo, ou seja:

$$l_{ij} = \begin{cases} u_{ij} - x_{ij} & (i,j) \in E_F \\ 0 & (i,j) \notin E_F. \end{cases}$$

No modelo para replicação de arcos num corte mínimo, a variável y_{ij} associada a cada arco da rede representa o número de replicações a serem realizadas. Tem-se então a formulação do problema resolvido localmente, chamado de Problema Guloso para Replicação:

$$(\text{PGR}) \begin{cases} \text{Min} & \sum_{(i,j)\in E} d_{ij} y_{ij} & & (PGR.0) \\ \text{Sujeito a} & \sum_{(i,j)\in E} c_{ij} y_{ij} \geq F_{faltante} & & (PGR.1) \\ & y_{ij} \leq l_{ij} & \forall (i,j) \in E & (PGR.2) \\ & y_{ij} \in \mathbb{N} & \forall (i,j) \in E & (PGR.3) \end{cases}$$

Substituindo-se $z_{ij} = l_{ij} - y_{ij}$ e definindo-se

$$C_{max} = \sum_{(i,j)\in E} l_{ij} c_{ij},$$

transforma-se PGR em uma versão multi-item do clássico problema da mochila:

$$(\text{PMM}) \begin{cases} \text{Max} & \sum_{(i,j)\in E} d_{ij} z_{ij} & & (PMM.0) \\ \text{Sujeito a} & \sum_{(i,j)\in E} c_{ij} z_{ij} \leq C_{max} - F_{faltante} & & (PMM.1) \\ & z_{ij} \leq l_{ij} & \forall (i,j) \in E & (PMM.2) \\ & z_{ij} \in \mathbb{N} & \forall (i,j) \in E & (PMM.3) \end{cases}$$

Caso $C_{max} < F_{faltante}$, então o problema é inviável, pois a capacidade total de expansão do corte é menor do que a expansão mínima necessária para satisfazer as demandas. Senão, obtém-se o valor de y, que será somado a x para obter-se o vetor c' atualizado de capacidades instaladas.

A modelagem através do problema da mochila é uma aproximação para o problema de escolher os arcos que devem ser replicados num corte mínimo. A solução ótima para PMM pode não ser a melhor solução para o problema geral (PERC), não havendo nem mesmo garantia da viabilidade da rede com as novas capacidades c'. Além disso, o problema da mochila é NP-completo, o que sugere sua resolução aproximada. A solução aproximada de PMM é feita através de uma conhecida heurística onde os arcos são ordenados pela razão crescente entre custo e capacidade. Arcos são escolhidos para replicação enquanto a capacidade replicada total não exceder $F_{faltante}$. O vetor x e a capacidade instalada da rede c' são atualizados e o procedimento de cálculo do fluxo máximo é repetido. Se o fluxo máximo da rede for inferior a F_{viavel}, o procedimento de aumento de capacidade do corte mínimo é repetido até que se obtenha uma rede viável.

Apresenta-se a seguir, em pseudo-código, o algoritmo GeraRedeViavel, com o procedimento básico de construção de rede viável x' para o fluxo F_{viavel}, a partir da rede x. Ele utiliza o algoritmo *push-relabel* descrito em [1] para encontrar o fluxo máximo (procedimento FluxoMaximo) e obter o corte mínimo de menor número de vértices no conjunto T (procedimento CorteMinimo, que retorna o conjunto expandível E_F), e retorna se foi possível encontrar rede viável:

Algoritmo 1. GeraRedeViavel(x, F_{viavel}, x')

$F_{max} \leftarrow$ FluxoMaximo
CorteMinimo(E_F)
$x' \leftarrow x$
enquanto $F_{max} < F_{viavel}$ **e** $E_F \neq \emptyset$ **fazer**
    ReplicaArcosCorte(E_F,$F_{viavel} - F_{max}$,x')
    Atualizar capacidades c'
    $F_{max} \leftarrow$ FluxoMaximo
    CorteMinimo(E_F)
fim-enquanto
se $F_{viavel} = F_{max}$
então retornar `Viavel`
senão retornar `Inviavel`

O algoritmo ReplicaArcosCorte descreve a heurística para resolução do problema da mochila multi-item descrita anteriormente, a fim de obter as replicações x do conjunto E_F, para tentar suprir $F_{faltante}$:

Algoritmo 2. **ReplicaArcosCorte**$(E_F, F_{faltante}, x)$

Calcular o valor de l_{ij} para todo arco $(i,j) \in E$
$L \leftarrow$ lista de arcos de E_F ordenados pela razão d_{ij}/c_{ij}
$F \leftarrow 0$; $y \leftarrow 0$
enquanto $F < F_{faltante}$ **e** $L \neq \emptyset$ **fazer**
  $(i,j) \leftarrow$ primeiro arco de L
  $L \leftarrow L - \{(i,j)\}$
  $y_{ij} \leftarrow \min(l_{ij}, \lceil (F_{faltante} - F)/c_{ij} \rceil)$
  $F \leftarrow F + y_{ij}c_{ij}$
fim-enquanto
$x \leftarrow x + y$

Ao final do procedimento GeraRedeViavel é obtida uma solução viável para o problema PERC. Testes iniciais mostraram que a heurística algumas vezes falhava, por não levar em conta a estrutura da rede ao escolher arcos para replicação (a escolha era baseada somente no custo). Foi então elaborada uma variação da heurística gulosa básica, que visa orientar a escolha dos arcos replicados através de informações de fluxo residual disponíveis após a aplicação do algoritmo de fluxo máximo.

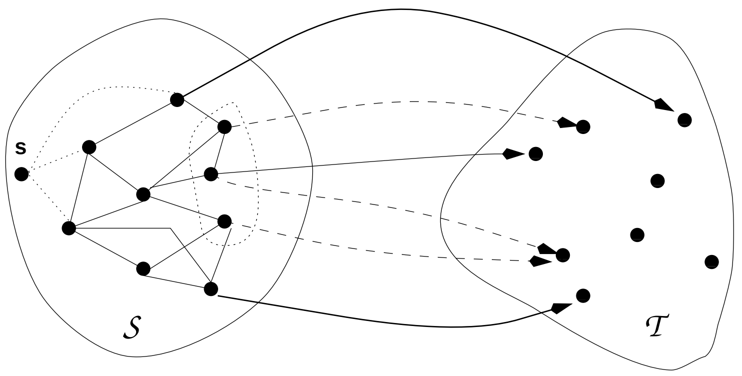
Variação da Heurística Básica por Controle de Excessos: É inútil replicar um arco se não é possível transmitir fluxo para seu nó origem. Pode-se utilizar o excesso de fluxo que é possível transmitir para esse nó como estimativa para o valor máximo que deve ser atingido pela capacidade do arco.

Seja o corte mínimo E_{ST} da Figura 3(a), onde E_F está representado pelos arcos pontilhados. O conjunto de nós de fronteira V_F (subconjunto destacado de S na figura) é formado pelas origens de arcos do corte E_F, ou seja:

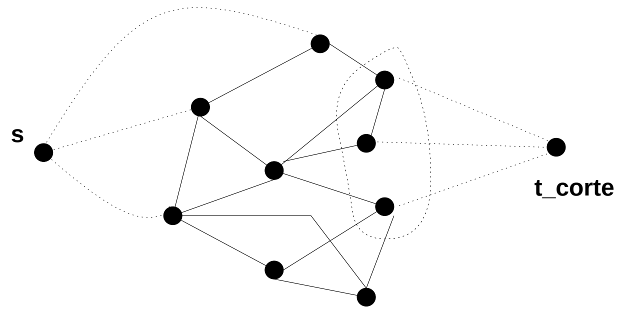
$$V_F = \{i \mid (i,j) \in E_F \text{ e } i \in S\}.$$

Há interesse em enviar todo o fluxo que sofreu contenção devido ao corte para os nós de V_F, utilizando os excessos para guiar a escolha de arcos a replicar. Para isso, o algoritmo de fluxo máximo é aplicado na rede composta apenas pelos nós de S e com super-sumidouro t_{corte} conectado aos nós de V_F (Figura 3(b)). A capacidade do arco entre um nó $i \in V_F$ e o super-sumidouro é dada pela soma das capacidades de replicação de todos os arcos de E_F que tenham origem naquele nó, ou seja, $c'_{i,t\_corte} = \sum_{j|(i,j) \in E_F} l_{ij}c_{ij}$. O valor do fluxo a ser enviado é $F_{faltante}$, ou seja, todo o fluxo que sofreu contenção.

Para cada nó i em V_F, o algoritmo de fluxo máximo fornece o valor do fluxo f_{i,t_corte} que passa pelo arco entre ele e o sumidouro. O excesso e_i é definido como sendo esse fluxo. O algoritmo MandaExcessoFronteira descreve o procedimento para obtenção dos excessos e, a partir dos conjuntos S e E_F:



(a) Corte mínimo



(b) Pseudo-sumidouro de corte

Figura 3: Excessos para nós de fronteira

Algoritmo 3. MandaExcessoFronteira(S, E_F, e)

Definir V_F
Construir nova rede a partir de S e V_F (Figura 3(b))
$F_{temp} \leftarrow$ FluxoMaximo
$e \leftarrow 0$
para $\forall i \in V_F$ **fazer**
    $e_i \leftarrow f_{i,t_{corte}}$
fim-para

O algoritmo ReplicaArcosCorte é modificado, de modo que um arco só seja replicado se o seu nó origem ainda tem excesso. Obtém-se então o novo algoritmo ReplicaArcosCorteExcesso:

Algoritmo 4. **ReplicaArcosCorteExcesso**($E_F, F_{faltante}, x$)

Calcular o valor de l_{ij} para todo arco $(i, j) \in E$
$L \leftarrow$ lista de arcos ordenados de E_F pela razão d_{ij}/c_{ij}
$F \leftarrow 0$; $y \leftarrow 0$
MandaExcessoFronteira(S, E_F, e)
enquanto $F < F_{faltante}$ **e** $L \neq \emptyset$ **fazer**
  $(i, j) \leftarrow$ primeiro arco de L
  $L \leftarrow L - \{(i, j)\}$
  se $e_i > 0$ **então**
    $y_{ij} \leftarrow \min(l_{ij}, \lceil (F_{faltante} - F)/c_{ij} \rceil)$
    $F \leftarrow F + y_{ij}c_{ij}$
    $e_i \leftarrow e_i - y_{ij}c_{ij}$
  fim-se
fim-enquanto
$x \leftarrow x + y$

Com essa nova limitação, espera-se replicar menos arcos desnecessários. Por outro lado, o espalhamento do excesso de fluxo para os nós de fronteira é apenas um limite inferior do fluxo que pode chegar ao nó, podendo ser pessimista e restringir replicações proveitosas.

Se existem dois ou mais cortes mínimos (de capacidade total F_{max}), o excesso em todos os nós de fronteira V_F poderá ser igual a 0. Isso ocorre porque CorteMinimo retorna sempre o corte mínimo que possui o menor número de nós em T. Logo, existe outro corte mínimo mais próximo à super-fonte pelo qual só passa o fluxo F_{max}, o que pode implicar na ausência de excesso na fronteira de E_F, já que o fluxo sofreu contenção nesse outro corte. Fica impossível replicar qualquer arco de E_F, já que $e = 0$. Para contornar esta dificuldade, são propostos dois procedimentos alternativos:

- aplicar o algoritmo ReplicaArcosCorte, que não leva em conta o excesso dos nós de fronteira; ou
- encontrar o corte mínimo mais próximo à fonte, para que se possa enviar fluxo para fronteira e, então, aplicar o algoritmo ReplicaArcosCorteExcesso.

No segundo caso, um novo corte mais próximo à fonte é encontrado através da rede residual resultante do algoritmo MandaExcessoFronteira utilizado para obter os excessos. Encontra-se um novo corte mínimo e redefinem-se os conjuntos S e V_F. Aplica-se novamente o algoritmo MandaExcessoFronteira e essa seqüência é repetida até que $e \neq 0$. Nesse momento, existe excesso na fronteira e pode-se aplicar o algoritmo ReplicaArcosCorteExcesso.

Retirada de Replicações Excedentes: Durante a construção da solução viável, replicações desnecessárias podem ter sido realizadas. Uma replicação excedente é aquela cuja remoção não implica na inviabilidade da solução. Ou seja, ao fazer $x_{ij} = x_{ij} - 1$ para o arco (i, j) (com $x_{ij} > 0$), a quantidade de fluxo que passa pela rede não diminui. O algoritmo apresentado a seguir pode ser aplicado para retirar todas as replicações excedentes de x, retornando o valor da redução de custo e mantendo a solução viável (suportando fluxo F):

Algoritmo 5. RetiraReplicacoesExcedentes(x, F)

$Melhoria \leftarrow 0$
para todo arco $(i, j) \mid x_{ij} > 0$ **fazer**
  repetir
    $x_{ij} \leftarrow x_{ij} - 1$
    $c'_{ij} \leftarrow c'_{ij} - c_{ij}$
    $Melhoria \leftarrow Melhoria + d_{ij}$
    $F_{max} \leftarrow$ FluxoMaximo
  enquanto $F_{max} = F$ e $x_{ij} > 0$
  se $F_{max} < F$ **então**
    $x_{ij} \leftarrow x_{ij} + 1$
    $c'_{ij} \leftarrow c'_{ij} + c_{ij}$
    $Melhoria \leftarrow Melhoria - d_{ij}$
  fim-se
fim-para
retornar $Melhoria$

A ordem na qual os arcos são verificados pode ter influência na qualidade da solução final gerada. Utilizou-se uma ordenação decrescente pelo custo unitário do fluxo no arco.

Busca Local na Rede: Com objetivo de melhorar a solução construída e encontrar-se um mínimo local, utiliza-se um algoritmo de busca local. A busca local parte de uma solução viável obtida por GeraRedeViavel e melhorada por RetiraReplicacoesExcedentes. Depois de apresentar o algoritmo básico de busca utilizado, define-se o conceito de vizinhança e apresenta-se o algoritmo para avaliar a variação na função objetivo devida a um movimento.

Em algoritmos de busca local do tipo descida-mais-rápida, avaliam-se todas as soluções na vizinhança da solução corrente, na procura do movimento que mais diminua seu custo. Se nenhum movimento aprimorante é encontrado, a busca termina com a solução corrente sendo um ótimo local. Caso contrário, o melhor movimento é executado, a solução é atualizada e a busca continua, explorando a nova vizinhança. O valor retornado pelo procedimento AvaliaMovimento(x, a, F, x^a) corresponde à redução no custo atingida pelo movimento de x para x^a. O algoritmo básico de busca local descrito a seguir encontra um mínimo local a partir de x, mantendo viabilidade para fluxo F.

Algoritmo 6. BuscaLocal(x, F)

---

repetir
  $Valor \leftarrow 0$
  $\overline{x} \leftarrow x$
  para toda solução vizinha x^a de x **fazer**
    $VariacaoMovimento \leftarrow \mathsf{AvaliaMovimento}(x, a, F, x^a)$
    se $VariacaoMovimento < Valor$ **então**
      $Valor \leftarrow VariacaoMovimento$
      $\overline{x} \leftarrow x^a$
    fim-se
  fim-para
  $x \leftarrow \overline{x}$
enquanto $Valor < 0$

Seja uma solução viável dada pelo vetor de replicações x. Sua vizinhança consiste em todas as soluções que podem ser obtidas pela eliminação de uma replicação de um arco da rede, com posterior replicação de outros arcos para recuperação de viabilidade. Assim, x^a é a solução vizinha de x obtida eliminando uma replicação do arco $a = (u, v)$ (com $x_a > 0$), isto é:

$$x_{ij}^a = \begin{cases} x_{ij} & \text{se } (i,j) \neq (u,v) \\ x_{ij} - 1 & \text{se } (i,j) = (u,v) \end{cases}$$

A rede resultante associada a uma solução vizinha não é necessariamente viável, já que a capacidade do arco a foi diminuída. Para recompor uma solução viável, um dos algoritmos construtivos apresentados deve ser utilizado, com a restrição de que o arco a não poderá ser replicado. Podem ocorrer dois casos:

- a rede associada à solução vizinha x^a não é viável, caso em que esta última é descartada; ou
- a rede é viável e, nesse caso, novas replicações foram necessariamente realizadas.

No segundo caso, o algoritmo de retirada de replicações excedentes deve ser novamente aplicado, para tentar melhorar a qualidade da solução vizinha. Depois da aplicação desse algoritmo, obtém-se finalmente a solução vizinha x^a, referente à remoção de uma replicação da aresta a. Note-se que pode haver substituição de um conjunto de replicações por outro na geração de um vizinho, quando várias replicações são inseridas pelo algoritmo construtivo e outras são retiradas na eliminação de excedentes. Percorrendo-se todas as arestas $a = (i, j)$ tais que $x_{ij} > 0$ na solução corrente, obtém-se aquela cuja remoção de uma replicação provoca a maior redução de custo, definindo-se então a melhor solução $\overline{x}$ vizinha da solução corrente x.

Baseado nesse conceito de vizinhança, o algoritmo AvaliaMovimento calcula a variação da função objetivo para o movimento de retirada de uma replicação do arco a da solução x, criando a solução vizinha x^a, que também é viável para o fluxo F. A alteração do estado da aresta visa evitar que a replicação removida seja reinserida pelo algoritmo GeraRedeViavel. A função CustoSolucao retorna o custo de x, ou seja, $\sum_{(i,j)\in E} x_{ij}d_{ij}$.

Algoritmo 7. AvaliaMovimento(x, a, F, x^a)

a.estado $\leftarrow$ PROIBIDO
$x_t^a \leftarrow x$
Retirar uma replicação da aresta $a = (i, j)$ da solução x_t^a
$ValorMovimento \leftarrow -d_{ij}$
$F_{max} \leftarrow$ FluxoMaximo
se $F_{max} < F$ **então**
  se GeraRedeViavel(x_t^a, F, x^a)
    $ValorMovimento \leftarrow ValorMovimento +$ (CustoSolucao(x^a) $-$ CustoSolucao(x_t^a))
    $ValorMovimento \leftarrow ValorMovimento -$ RetiraReplicacoesExcedentes(x^a, F)
  senão
    $ValorMovimento \leftarrow \infty$
  fim-se
senão
  $x^a \leftarrow x_t^a$
fim-se
a.estado $\leftarrow$ LIVRE
retornar $ValorMovimento$

3.2 Busca no Conjunto de Agentes para Identificar Cortes

Uma estratégia de busca local é utilizada para a geração de restrições que estejam sendo violadas pela solução corrente do problema restrito. Uma solução para este processo de busca é definida pela escolha de um subconjunto de agentes (representado por seu vetor característico λ). Estes agentes estão associados a uma certa demanda total, que implicará na necessidade de construção de uma rede viável para atendê-los, utilizando-se para isso a heurística descrita anteriormente.

Em um procedimento de busca local, como já mencionado, o espaço de soluções é explorado através da aplicação de pequenas modificações à solução corrente para transformá-la em outra que dará continuidade à trajetória. Para isso, é necessário que uma solução inicial seja fornecida. Deve-se também definir o tipo de vizinhança que será considerado na obtenção de novas soluções e estabelecer um critério de parada para a estratégia.

Solução Inicial: Foram testadas duas estratégias para a obtenção da solução inicial para a busca no conjunto de agentes:

- Inicialização aleatória: determina-se aleatoriamente os agentes que farão parte da solução inicial (probabilidade 0.5 para cada agente).
- Solução anterior: toma-se como solução inicial o conjunto de agentes que gerou a última restrição encontrada, por ocasião da separac,a o anterior (inicialização aleatória é utilizada na primeira separação).

Vizinhança e Movimentos: A vizinhança explorada pela heurística consiste em todas as soluções que diferem da corrente por apenas um agente. Dessa forma, os movimentos a partir da solução corrente consistem na inserção de um agente que não esteja na solução ou na retirada de um agente que faça parte dela. Para que os movimentos sejam gerados e avaliados mais rapidamente, os agentes são implantados na rede e são instaladas as arestas necessárias para o transporte do fluxo da oferta até a demanda utilizando-se apenas as estratégias de construção de soluções propostas anteriormente. Em seguida, são eliminadas as possi'veis replicac,o es excedentes. Ou seja, no cálculo do custo de uma solução vizinha não é empregado o Algoritmo 6 para busca local na rede descrito na subseção anterior. Portanto, os custos das soluções vizinhas são limites superiores mais grosseiros para os custos de implantação dos respectivos agentes na rede. Esta aproximação é necessária para acelerar o processo de busca no conjunto de agentes.

Lista de Candidatos: A cada iteração, o movimento que será realizado é escolhido de forma aleatória. Para isso, monta-se um vetor com os custos de todos movimentos, ordenados de forma crescente. Somente serão para seleção aleatória os movimentos cujo custo esteja no intervalo

$$[c_{menor}, c_{menor} + (c_{maior} - c_{menor})\alpha],$$

onde c_{menor} é o custo do melhor movimento e c_{maior}, o do pior movimento. O valor do parâmetro α é usado para controlar o tamanho da lista de movimentos candidatos, ou seja, para $\alpha = 0$, somente o(s) movimento(s) que produze(m) a(s) solução(ões) de menor custo poderá(ão) ser escolhido(s) e para $\alpha = 1$, qualquer um dos movimentos poderá ser escolhido. O valor do parâmetro α utilizado nos testes foi igual a 50%. Após a escolha aleatória do movimento, ele é efetivado e aplica-se o procedimento de busca local na rede, como descrito anteriormente, a fim de procurar melhorar ainda mais o custo da solução vizinha escolhida, aumentando a chance de se encontrar um corte. Esse custo ainda é um limite superior para o custo mínimo da implantação destes agentes na rede, visto que o PERC é resolvido através de heurísticas.

Critério de Parada: A busca no conjunto de agentes termina quando:

- para os valores correntes θ dos custos alocados aos agentes, uma nova restrição violada (corte) é encontrada e incorporada ao novo problema restrito; ou

- após um certo número MaximoTentativas iterações sem que uma nova restrição violada tenha sido encontrada, caso em que a separação heurística não foi capaz de obter cortes (sendo então necessário que o problema de separação seja resolvido de forma exata).

O pseudo-código abaixo descreve o algoritmo SeparacaoHeuristica que, a partir de uma alocação de custos θ, implementa busca no espaço de conjunto de agentes, retornando se achou um corte associado ao conjunto de agentes S, com custo Z_S, ou se não foi possível encontrá-lo. Ele se utiliza do procedimento ExpansaoHeuristica(F_S, x), que retorna uma replicação viável x que suporte o fluxo F_S.

Algoritmo 8. SeparacaoHeuristica(θ, S, Z_S)

$NumeroTentativas \leftarrow 0$
Obter solução inicial S
$C_S \leftarrow \sum_{k \in S} \theta_k$; $F_S \leftarrow \sum_{k \in S} b_k$
ExpansaoHeuristica(F_S, x)
$Z_S \leftarrow$ CustoSolucao(x)
$AchouCorte \leftarrow (C_S > Z_S)$? VERDADE : FALSO
enquanto $AchouCorte$ = FALSO **e** $NumeroTentativas <$ MaximoTentativas
  para k variando de 1 a K **fazer**
    se $k \in S$
      Retirar agente k; $C_S^k \leftarrow C_S - \theta_k$; $F_S^k \leftarrow F_S - b_k$
    senão
      Inserir agente k; $C_S^k \leftarrow C_S + \theta_k$; $F_S^k \leftarrow F_S + b_k$
    fim-se
    GeraRedeViavel(x, F_S^k, x_t)
    $Z_S^k \leftarrow$ CustoSolucao(x_t) - RetiraReplicacoesExcedentes(x_t, F_S^k)
    se $C_S^k > Z_S^k$
      $AchouCorte \leftarrow$ VERDADE
      $m \leftarrow k$ (agente associado ao movimento que gera o corte)
    fim-se
  fim-para
  se $AchouCorte$ = FALSO
    Montar lista de candidatos com base nos valores $Z_S^k - C_S^k, k = 1, \ldots, K$
    Escolher aleatoriamente um agente m que define um movimento
  fim-se
  Modificar x e S para realizar movimento alterando o estado do agente m
  BuscaLocal(x, F_S^m)
  $Z_S \leftarrow$ CustoSolucao(x); $C_S \leftarrow C_S^m$; $F_S \leftarrow F_S^m$
  se $C_S > Z_S$
    $AchouCorte \leftarrow$ VERDADE
  fim-se
  $NumeroTentativas \leftarrow NumeroTentativas + 1$
fim-enquanto
retornar $AchouCorte$

Algoritmo 9. **ExpansaoHeuristica**(F_S, x)

$x' \leftarrow 0$
GeraRedeViavel(x', F_S, x)
RetiraReplicacoesExcedentes(x, F_S)
BuscaLocal(x, F_S)

3.3 Algoritmo Completo para Solução de PACST

O pseudo-código para o algoritmo `Resolve-PACST-Acelerado` que, a partir do valor ótimo de expansão da rede Z^*, procura uma alocação de custo viável θ, resolvendo o problema PACST:

Algoritmo 10. **Resolve-PACST-Acelerado**(Z^*, θ)

Inicializar o problema restrito pela restrição $\sum_{k=1}^{K} \theta_k = Z^*$
$AlocacaoViavel \leftarrow$ FALSO
enquanto ($AlocacaoViavel$ = FALSO) **fazer**
  Resolver o problema linear restrito, obtendo a alocação θ
  se SeparacaoHeuristica(θ, S, Z_S)
    Acrescentar ao problema restrito o corte definido por S e Z_S
  senão
    se SeparacaoExata(θ, S, Z_S)
      Acrescentar ao problema restrito o corte definido por S e Z_S
    senão
      $AlocacaoViavel \leftarrow$ VERDADE
    fim-se
  fim-se
fim-enquanto

Vale ressaltar que tanto a separação heurística quanto a separação exata podem gerar mais de um corte por vez. Na descrição acima apresenta-se a geração de uma única restrição para simplificar a apresentação. Além disso, SeparacaoExata pode implementar tanto a versão `Exato-Primeiro` quanto `Exato-Mínimo`.

4 Resultados Computacionais

Os resultados computacionais abaixo referem-se ao caso fornecido pelo CEPEL para efeito de experiências computacionais e validação do modelo e das técnicas propostos. Este caso representa uma parte da rede brasileira, composta de 79 barras e 283 circuitos, das quais 134 podem ser duplicados. O número de agentes é 19. Há restrições quanto ao número máximo de duplicações de cada circuito (no máximo 3) e o custo unitário de fluxo varia de circuito para circuito (no intervalo de 2.8 a 111.94). A descrição da demanda de cada agente é fornecida na primeira coluna da Tabela 2.

Na Tabela 1 são apresentados resultados computacionais para quatro variantes de solução para o problema de separação. As colunas Aleatorio e Anterior correspondem a duas variantes do algoritmo `Resolve-PACST-Acelerado`, referindo-se às duas estratégias de inicialização utilizadas na busca local feita por SeparacaoHeuristica. As colunas `Exato-Primeiro` e `Exato-Minimo` referem-se aos resultados obtidos sem a utilização da separação heurística. Observe-se que o procedimento `Exato-Minimo` corresponde ao algoritmo originalmente proposto em [2]. Nas duas variantes de `Resolve-PACST-Acelerado` que utilizam a separação heurística, a SeparacaoExata implementa a variante `Exato-Primeiro`, parando assim que obtém os 5 primeiros cortes ou que tiver explorado toda a árvore de busca. Caso não haja solução de custo negativo, o critério de parada foi atingido.

Todos os tempos são dados em minutos e se referem à utilização efetiva de CPU. A máquina utilizada foi uma Sun UltraSparc, com 256M de memória. Na primeira linha apresenta-se o tempo total de execução até a obtenção de uma alocação de custos viável. Na segunda linha, apresenta-se o tempo gasto dentro do procedimento SeparacaoHeuristica. Na terceira, apresenta-se o tempo gasto no passo de validação, que corresponde à última chamada dos procedimentos SeparacaoHeuristica e SeparacaoExata, quando é detectada a viabilidade da alocação corrente. A quarta linha apresenta o total de cortes gerados e, na quinta, indica-se quantos desses cortes foram gerados pela separação heurística. A seguir, apresenta-se o número de total de passos (número de separações realizadas) e, finalmente, a última linha indica em quantos desses passos a heurística não foi capaz de identificar um corte, exigindo então o emprego da separação exata.

	`Resolve-PACST-Acelerado`			
	Aleatorio	Anterior	`Exato-Primeiro`	`Exato-Minimo`
Tempo total	2943.34	2515.83	3881.47	5539.67
Tempo na heurística	67.18	41.34	-	-
Tempo de validação	283.13	311.37	420.06	305.37
Total de cortes	246	235	247	75
Cortes pela heurística	109	140	-	-
Total de passos	142	166	59	76
Separações exatas	33	26	59	76

Tabela 1: Resultados computacionais para o caso fornecido pelo CEPEL (19 agentes)

Comparando-se as duas últimas colunas, onde a separação exata é sempre empregada, observa-se que a relaxação da busca da restrição mais violada diminui em cerca de 30% o tempo de processamento. O procedimento `Exato-Primeiro` é sensivelmente mais eficiente do que `Exato-Minimo`. Embora o número de cortes gerados por `Exato-Minimo` seja menor, na média gasta-se muito mais tempo para gerar cada corte. O compromisso entre a qualidade e a quantidade dos cortes gerados deve ser investigado mais profundamente.

Considera-se agora a comparação entre o enfoque originalmente proposto para a solução de PACST e aquele proposto neste trabalho, baseado no emprego de separações heurísticas. Observa-se assim que o tempo de processamento utilizando-se a separação heurística com o procedimento de inicialização Anterior é cerca de 65% do tempo gasto por `Exato-Primeiro`. Ou seja, a geração aproximada de cortes é bastante importante na diminuição do tempo de processamento. Nota-se ainda que apenas 3% do tempo total de processamento é gasto na solução da separação heurística e, que, nessa fração de tempo, cerca de 60% dos cortes são gerados. Um investimento no melhoramento da qualidade e/ou quantidade dos cortes gerados pela heurística poderá reduzir ainda mais o tempo de processamento, já que poderá evitar chamadas ao algoritmo SeparacaoExata.

Outra possibilidade para melhoria do tempo de processamento utilizando o modelo proposto consiste na calibragem do algoritmo de *branch-and-bound* utilizado na separação exata, através da utilização de critérios mais adaptados ao problema. Geralmente, modificações na ordem de fixação de variáveis e o cálculo de limites, entre outras, permitem melhorar significativamente o desempenho desse algoritmo. Também deve-se considerar a utilização de outros algoritmos exatos na resolução do modelo, utilizando-se a teoria desenvolvida para problemas do tipo *fixed charge*.

Na Tabela 2 são apresentadas as alocações obtidas por cada uma das variantes, bem como o valor ótimo de Δ. Note-se que o valor de Δ^* não é nulo, contrariamente ao que era esperado pela formulação do modelo. Modificações no modelo devem ser realizadas para que esse problema seja superado e algumas sugestões são apresentadas na próxima seção. Além disso, deve-se caracterizar melhor o conceito de similaridade de agentes para a atribuição de custos, a fim de também incorporá-lo ao modelo.

5 Avaliação do Modelo

Foram encontrados dois motivos para que Δ^* seja positivo:

- A utilização da rede do sistema existente a custo nulo, na resolução dos problemas para subconjuntos isolados de agentes.
- A existência de limites para o número de replicações de um circuito, atingidos em Z_S e Z^*.

Para exemplificar, seja a seguinte instância: duas barras a e b, com dois agentes 1 e 2, ambos com demanda na barra a e oferta na barra b de 10 MW. Existem dois circuitos x e y conectando a e b, ambos com capacidade máxima de 10 MW.

	Demanda	Aleatorio	Anterior	Exato-Primeiro	Exato-Minimo
Δ^*	-	277765.32	277765.32	277765.32	277765.32
θ_1	317	207.88	208.67	208.67	208.67
θ_2	154	98.78	15.05	98.78	15.05
θ_3	345	9.70	47.08	9.70	47.08
θ_4	46	0.00	0.00	0.00	0.00
θ_5	531	35.54	35.10	35.10	35.25
θ_6	1259	103.54	103.54	103.54	103.48
θ_7	1631	54.04	53.90	53.90	53.75
θ_8	49	22.53	22.53	56.94	22.53
θ_9	405	29.89	30.46	30.46	30.26
θ_{10}	207	25.99	26.03	26.03	25.63
θ_{11}	212	166.02	165.92	165.92	167.09
θ_{12}	647	33.85	33.85	33.85	33.85
θ_{13}	100	23.51	23.44	23.44	24.27
θ_{14}	60	46.50	46.50	18.40	46.50
θ_{15}	100	0.00	0.00	0.00	0.00
θ_{16}	84	27.99	27.91	27.91	28.89
θ_{17}	153	18.15	18.15	18.15	18.15
θ_{18}	155	30.34	30.25	30.25	31.32
θ_{19}	132	22.95	23.05	23.05	21.80

Tabela 2: Soluções obtidas em custo unitário por MW ($Z^* = 693273.18$)

Para análise da rede existente, suponha-se que x já esteja instalado, não podendo ser replicado, e que o custo de replicação de y seja igual a 10. O valor de Z^* é 10, mas o valor de Z^S é nulo tanto para o agente 1, quanto para o agente 2, separadamente, já que todo o fluxo pode ser passado pelo circuito já instalado. O modelo 1 compreende então as seguintes restrições:

$$\theta_1 + \theta_2 + \Delta = 10$$

$$\theta_1 \leq 0$$

$$\theta_2 \leq 0$$

onde $\Delta^* = 10$. A solução mais simples para esse problema seria a proibição da utilização da rede já existente no cálculo dos Z_S para os subsistemas formados por agentes isolados. A justificativa econômica para isto é que, como o subconjunto de agentes não fará parte do consórcio, não poderá utilizar os recursos existentes do mesmo.

Para o problema com a limitação no número de replicações de circuito, suponha-se agora que x e y não estão instalados, podendo ser replicados uma única vez, com custos 5 e 10, respectivamente. O valor de Z^* é 15, já que não é possível enviar todo o fluxo pelo menor caminho. Mas Z_S é igual a 5, tanto para o agente 1, quanto para o agente 2, separadamente. Assim, tem-se:

$$\theta_1 + \theta_2 + \Delta = 15$$
$$\theta_1 \leq 5$$
$$\theta_2 \leq 5$$

com $\Delta^* = 5$. Para suplantar esse problema, o modelo deve refletir, de alguma forma, a limitação de recursos existentes ao representar as restrições de estabilidade.

6 Análise de Resultados e Extensões

Encontrar uma solução viável para PACST é uma tarefa árdua em termos coputacionais. Entretanto, pode-se verificar que as dificuldades não são intransponíveis e que é possível desenvolver algoritmos eficientes. Algumas observações indicam os aspectos mais relevantes da metodologia empregada:

- A obtenção do custo ótimo Z^* de expansão do sistema global formado por todos agentes consiste na resolução de um PERC que consome um enorme esforço computacional (aproximadamente 5 horas no caso estudado). Este é o ponto de partida para a resolução do PACST e um de seus principais limitantes. Acredita-se que o estudo dedicado desse problema, seguindo as linhas apresentadas no apêndice (e em conjunto com heurísticas e uma análise das características específicas das instâncias reais) possa gerar algoritmos mais satisfatórios.
- A instância do caso realista proposto pelo CEPEL foi resolvida. Os resultados evidenciam a importância da abordagem heurística para o problema Sep-PACST. Entretanto, ficou claro também que o grande esforço computacional exigido limitou muito as experiências para se encontrar a melhor parametrização da heurística e sua intervenção ao longo do algoritmo de resolução do PACST. É possível se prever ganhos significativos em eficiência apenas com a busca de uma melhor parametrização. Algumas possibilidades são:
  - Melhorar a qualidade dos cortes gerados pelas heurísticas.
  - Gerar vários cortes a cada chamada da heurística.
  - Analisar o compromisso entre qualidade do corte gerado e o tempo gasto, principalmente através do algoritmo `Exato-Primeiro`.
  - Aumentar a eficiência da separação exata através da otimização das estratégias do *branch-and-bound* utilizado na sua resolução.
  - Avaliação de novas heurísticas para a separação.

- Verificou-se que a solução ótima da instância testada é *inviável*, ou seja, o valor da variável Δ no ótimo é estritamente superior a zero. Isto se deve ao fato de que a resolução independente dos problemas de separação permite o uso de um mesmo recurso (por exemplo, uma linha de transmissão existente) por mais de um agente ao mesmo tempo. Os fatores que levam a este compartilhamento são o custo baixo (zero) das linha existentes e os limites considerados para a expansão de uma linha de transmissão (por exemplo, permite-se no máximo duas replicações de uma dada linha), além de vantagens de coalisão. São necessárias alterações no modelo, a fim de evitar a sua inviabilidade, que se deve às simplificações feitas. Além disso, uma caracterização mais elaborada da função objetivo do problema de encontrar uma alocação viável pode permitir encontrar alocações economicamente mais justas. Algumas possibilidades seriam:
  - Incorporação da rede de transmissão existente ao núcleo do jogo cooperativo, transformando as linhas já instaladas em agentes que poderiam se consorciar, oferecendo sua capacidade de transmissão. O sistema existente passa a ser remunerado por sua utilização. Uma das vantagens é a capacidade de se inferir o valor de mercado das linhas instaladas. Uma das grandes desvantagens é o crescimento do tamanho do problema, já que o número de restrições é exponencial no número de agentes. Soluções intermediárias, com utilização de agentes que são compostos por todo um conjunto de linhas, podem representar uma alternativa viável.
  - Representação dos problemas de compartilhamento de recursos escassos, que geram a inviabilidade do modelo. Consórcios diferentes não podem utilizar, ao mesmo tempo, mais do que a totalidade do recurso existente. É necessário expressar, de alguma forma, esse acoplamento existente entre as diversas restrições do núcleo.
  - Modificação da função objetivo, a fim de melhor expressar questões de uniformidade na alocação dos custos, evitando incentivos muito grandes a alguns consórcios em relação a outros. Uma possibilidade é a minimização do maior incentivo (folga existente em cada restrição).

Referências

[1] R. K. Ahuja, T. L. Magnanti e J. B. Orlin, *Network Flows: Theory, Algorithms and Applications*, Prentice-Hall, 1993.

[2] N. Campodónico e M.V. Pereira, "A Methodology for Transmission Service Cost Allocation in the Core of Cooperative Games", Relatório Técnico, Power Systems Research Inc., Rio de Janeiro, 1996.

[3] B.V. Cherkassky e A.V. Goldberg, "On Implementing Push-relabel Method for the Maximum Flow Problem", Technical Report STAN-CS-TR-94-1523, Stanford University, 1994.

[4] A.V.Goldberg, E. Tardos e R.E. Tarjan , "Network Flow Algorithms", in *Flows, Paths and VLSI Layout* (B. Korte, L. Lovasz, H.J. Pronel e A. Schrijver, eds.), 101-164, Springer-Verlag, 1990.

[5] M. GRÖTSCHEL, L. LOVASZ E A. SCHRIJVER, "The Ellipsoid Method and Its Consequences in Combinatorial Optimization", *Combinatorica* 1 (1981), 169-197.

[6] Z. GU, G. NEMHAUSER e M. SAVELSBERGH, "Lifted Flow Cover Inequalities for Mixed 0-1 Integer Programs", Relatório Técnico, 1996.

[7] L.G. KHACHIAN, "A Polynomial Time Algorithm in Linear Programming", *Soviet Mathematics Doklady* 20 (1979), 191-194.

[8] G.L. NEMHAUSER e L.A. WOLSEY, *Integer and Combinatorial Optimization*, Wiley, 1988.

[9] M. PADBERG, T. VAN ROY e L. WOLSEY, "Valid Linear Inequalities for Fixed Charge Problems", *Operations Research* 33 (1985), 842-861.

[10] S. THIENEL, "ABACUS-A Branch-And-CUt System", Doctoral Thesis, Universität zu Köln, 1995.

[11] E. ZEMEL, "Polynomial Algorithms for Estimating Network Reliability", *Networks* 12 (1982), 439-452.

A O Problema "Fixed Charge Network Flow"

O *Fixed Charge Network Flow* (FCNF) é um tipo de problema de fluxo em redes no qual os custos de se usar os arcos são fixos, no sentido de que se algum fluxo passar por um arco paga-se um custo pré-determinado. Se nenhum fluxo passar por esse arco, o custo é nulo. Nos problemas de fluxo em rede tradicionais, o custo de um arco é proporcional ao fluxo que passa nesse arco. Ao contrário do fluxo em rede tradicional, que pode ser resolvido de forma eficiente, o FCNF é NP-difícil.

O FCNF pode ser formulado da seguinte maneira. Sejam $G = (V, A)$ um grafo orientado representando a rede, b_i a demanda (ou oferta, se negativo) de fluxo no nó i, d_{ij} o custo fixo de se ter fluxo no arco (i, j), h_{ij} uma eventual parcela do custo proporcional ao fluxo e c_{ij} a capacidade máxima deste arco. Seja f_{ij} uma variável contínua representando a quantidade de fluxo que passa no arco (i, j) e x_{ij} uma variável binária indicando se este arco pode ou não ser atravessado por fluxo.

$$
(FCNF)\left\{\begin{array}{lllr}
\text{Min} & \displaystyle\sum_{(i,j)\in A} d_{ij}x_{ij} + h_{ij}f_{ij} & & (FNCF.0) \\
\text{Sujeito a} & \displaystyle\sum_{j:(j,i)\in A} f_{ji} - \sum_{j:(i,j)\in A} f_{ij} = b_i & \forall i \in V & (FNCF.1) \\
& f_{ij} \leq c_{ij}x_{ij} & \forall (i,j) \in A & (FNCF.2) \\
& f_{ij} \geq 0,\ x_{ij} \in \{0,1\} & \forall (i,j) \in A & (FCNF.3)
\end{array}\right.
$$

O FCNF pode ser resolvido por um algoritmo de *branch-and-bound* com base na formulação acima, utilizando-se sua relaxação linear para obtenção de limitantes. Entretanto, essa abordagem só costuma funcionar bem em instâncias pequenas, pois o salto de integralidade pode ser muito grande. A dificuldade é que ao se relaxar a integralidade das variáveis x, as restrições (FCNF.2) se tornam muito fracas, pois para passar um certo fluxo positivo f_{ij} só é necessário fazer $x_{ij} = f_{ij}/c_{ij}$. Isso faz com que só se obtenha soluções inteiras depois de se fixar um grande número de variáveis, o que pode levar a árvores de enumeração de tamanho impraticável. Para resolver instâncias maiores pode-se usar um método de *branch-and-cut*, ou seja, ao longo do *branch-and-bound* introduzir novas desigualdades para reduzir o salto de integralidade e o tamanho das árvore.

Dentre as famílias de desigualdades válidas conhecidas para o FCNF, as mais efetivas são as chamadas *flow cover inequalities*, descritas por Padberg, Van Roy e Wolsey [9]. Mais recentemente, elas foram superadas pelas chamadas *lifted flow cover inequalities* desenvolvidas por Gu, Nemhauser e Savelsbergh [6]), que como o próprio nome indica, são uma generalização e fortalecimento das primeiras. Um algoritmo de *branch-and-cut* com as *lifted flow cover inequalities* resolveu instâncias do FCNF em redes com mais de 300 arcos em tempo de processamento bastante razoável [6]. Na verdade, a complexidade de um FCNF pode variar enormemente de acordo com a topologia da rede e os valores dos custos e capacidades dos arcos. É possível que algumas instâncias bem menores não possam ser bem resolvidas. Mas, de qualquer forma, esse *branch-and-cut* sempre se mostrou muito mais eficiente do que o enfoque mais tradicional.

Ilustra-se um caso particular de uma dessas desigualdades, para que se tenha uma noção de sua natureza. Sejam $\delta^+(i) = \{(j,i) \in A\}$ e $\delta^-(i) = \{(i,j) \in A\}$ os conjuntos de arcos que, respectivamente, entram e saem do nó i. Supõe-se que um certo nó i da rede só tenha arcos de entrada, i.e. $\delta^-(i) = \emptyset$. Um conjunto $C^+ \subseteq \delta^+(i)$ é chamado de *flow cover* se $\sum_{(j,i)\in C^+} c_{ji} > b_i$. A desigualdade

$$\sum_{(j,i)\in C^+} f_{ji} \leq b_i - \sum_{(j,i)\in C^{++}} (c_{ji} - \beta)(1 - x_{ji}),$$

onde $\beta = \sum_{(j,i)\in C^+} c_{ji} - b_i$ e $C^{++} = \{(j,i) \in C^+ : c_{ji} > \beta\}$, é um exemplo de uma *flow cover inequality*. Essa desigualdade é valida, ou seja, não corta nenhuma solução inteira, o que pode ser verificado pelo seguinte argumento. Se todos os x_{ji} correspondentes aos arcos em C^{++} forem iguais a 1, a desigualdade diz que o máximo fluxo que pode entrar em i através do conjunto C^+ é menor do que a demanda b_i do nó, o que é claramente válido. Mas, para cada x_{ji} de um arco em C^{++} que valer 0, esse máximo fluxo é reduzido em pelo menos $c_{ji} - \beta$ unidades.

Segue-se um exemplo de como uma solução fracionária pode ser cortada por essa desigualdade. Sup oe-se que o nó i tenha dois arcos de entrada, (a,i) e (b,i), com $c_{ai} = 3$, $c_{bi} = 2$ e $b_i = 4$. Supõe-se ainda que a solução da relaxação linear de FCNF seja $f_{ai} = 2$, $f_{bi} = 2$, $x_{ai} = 2/3$, $x_{bi} = 1$. Tomando-se $C^+ = \{(a,i),(b,i)\}$, β tem valor 1 e $C^{++} = C^+$, levando à desigualdade

$$f_{ai} + f_{bi} \leq 1 + 2x_{ai} + x_{bi}.$$

Essa desigualdade faz com que x_{ai} passe a ser 1, pois seu lado direito deve valer 4 para satisfazer a demanda. Caso um nó tenha arcos de entrada e de saída, as correspondentes *flow cover inequalities* são mais complexas. As *lifted flow cover inequalities*, por sua vez, usam essas desigualdades como ponto de partida para gerar desigualdades ainda mais fortes.

A.1 Solução de PERC como um "Fixed Charge Network Flow"

O problema PERC apresenta semelhanças ao problema do FCNF, pois seu principal fator complicante é o fato de que, para se construir ou replicar uma linha de transmissão com uma certa capacidade, incorre-se em um custo fixo, independente da quantidade de energia que irá passar por essa linha. Dessa forma, há potencial para se melhorar a resolução de problemas formulados como PERÇ transformando-os em FCNFs e aplicando-se um *branch-and-cut* que incorpore as *lifted flow cover inequalities*.

Há que se considerar, entretanto, duas diferenças importantes entre esses dois tipos de problemas, que impedem a aplicação direta das *flow cover inequalities*:

- O FCNF é definido como um problema de fluxo em arcos. Em particular, as *flow cover inequalities* são definidas a partir desse modelo de arcos. Já o PERC é um problema de fluxo em arestas, representando o fato de que uma linha de transmissão pode ser usada em um sentido ou em outro.

- No FCNF todas as variáveis inteiras são binárias 0-1, indicando se um arco pode ser usado ou não. A validade das *flow cover inequalities* depende dessa propriedade. No PERC as variáveis inteiras podem assumir qualquer valor, condicionadas apenas pelos limites u.

A transformação de uma instância do PERC em um FCNF pode ser feita da seguinte maneira. Recorde-se que uma aresta (i, j) possui quatro características: sua capacidade instalada $\overline{c}_{ij}$, sua capacidade aumentada c_{ij} em cada replicação, o d_{ij} custo de uma replicação, e o número máximo u_{ij} de replicações. A capacidade instalada desta aresta pode ser representada por um par de arcos em sentidos opostos (entre (i, j) e (j, i)), ambos com capacidade $\overline{c}_{ij}$ e custos nulos. Arcos já "instalados" são representados no FCNF como arcos de custo nulo e as variáveis 0-1 associadas a esses arcos podem ser fixadas em 1 e eliminadas da formulação. Suponha-se agora que $u_{ij} = 1$. Nesse caso, o potencial de replicação desta aresta pode ser representada por outro par de arcos opostos, ambos com capacidade c_{ij} e custo fixo d_{ij}. Existiriam duas variáveis de fluxo, por exemplo f_a e f_b, associadas a esses arcos, mas apenas uma única variável 0-1 associada ao par, por exemplo x_c. Duas restrições de (FCNF.2) são usadas: $f_a \leq c_{ij}x_c$ e $f_b \leq c_{ij}x_c$.

Entretanto, se $u_{ij} > 1$ é necessário introduzir novos arcos para que todas as variáveis inteiras sejam 0-1. Por exemplo, seja $u_{ij} = 7$. Nesse caso, o potencial de replicação da aresta pode ser representado por 3 pares de arcos opostos: o primeiro par com capacidades c_{ij} e custos fixos d_{ij}, o segundo com capacidades $2c_{ij}$ e custos $2d_{ij}$ e o terceiro com capacidades $4c_{ij}$ e custos $4d_{ij}$. Há 6 variáveis de fluxo, 3 variáveis 0-1 e 6 restrições de (FCNF.2) associadas a esses arcos. Pode-se verificar que qualquer número inteiro de replicações entre 0 e 7 pode ser representado por alguma maneira de fixar essas 3 variáveis em 0 ou 1. Esse esquema pode ser usado para qualquer valor de u_{ij}, usando-se $\log u_{ij} + 1$ pares de arcos.[2]

Após essa transformação obtém-se um problema de FCNF, que pode ser resolvido por um *branch-and-cut* usando as *flow cover inequalities*.

A.2 Solução de Sep-PACST como um "Fixed Charge Network Flow"

O problema Sep-PACST pode ser transformado em um FCNF de modo similar ao PERC. A única questão adicional é como modelar o fato de que a demanda/oferta de um nó depende das variáveis λ_k.

A idéia consiste em representar a porção b_i^k da demanda do nó i associada ao agente k como um arco adicional na rede. Se b_i^k for positivo, esse arco sai do nó, e vice-versa. A variável de fluxo f_k^i representa o fluxo que passa por esse arco, $c_k^i = |b_k^i|$ e $d_k^i = 0$. Esse fluxo está associado à variável inteira λ_k. Há, entretanto, um detalhe importante: ao invés de se usar a restrição (F1.2) na forma usual $f_k^i \leq c_k^i \lambda_k$, deve-se usar $f_k^i = c_k^i \lambda_k$. Esse detalhe faz com que a transformação leve a um problema que foge um pouco do modelo do FCNF apresentado nesse apêndice.

O aspecto importante é que esse problema transformado pode ser resolvido por um algoritmo de *branch-and-cut* usando as *flow cover inequalities*, de modo idêntico a um FCNF. A questão é que qualquer desigualdade que seja válida para $f_k^i \leq c_k^i \lambda_k$ também o será para a igualdade $f_k^i \leq c_k^i \lambda_k$.

[2] Em programação inteira, de modo geral, sempre existem transformações capazes substituir uma variável inteira entre 0 e u por $\log u{+}1$ variáveis 0-1, eventualmente introduzindo-se novas restrições ou novas variáveis contínuas.