6
Generalized Assignment Problem (GAP)

O Problema de Alocação Generalizado (*Generalized Assignment Problem*, GAP) é definido como o problema de associar um conjunto de tarefas a um conjunto de máquinas. Cada possível associação de uma tarefa a uma máquina consome uma determinada quantidade de recursos da máquina e tem um custo, enquanto cada máquina possui uma quantidade máxima de recursos que podem ser consumidos pelas tarefas. Deseja-se encontrar a associação de custo total mínimo.

O GAP é um dos mais famosos problemas NP-Difíceis. Somente determinar se existe ou não solução viável para uma instância do GAP é um problema NP-Completo [34]. É um problema de formulação simples e de aplicações importantes, em problemas do "mundo-real" [35] e em problemas de *scheduling*, *supply chain* e *vehicle routing*, fazendo com que a formulação de heurísticas e algoritmos exatos tenha importância tanto teórica quanto prática [39].

O GAP foi o problema escolhido para ser abordado neste trabalho por diferentes motivos:

- é um problema intuitivo, com uma modelagem simples e de grande aplicação prática;
- possui uma modelagem alternativa (exponencial) que gera bons resultados, descrita na seção 6.2;
- possui um conjunto de instâncias de tamanho médio e com variados graus de dificuldade;
- existem metaeurísticas altamente especializadas que possuem ótimos resultados, como as descritas nas seções 6.3.3, 6.3.4 e 6.3.5.

Todas estas características fazem do GAP um excelente problema para se experimentar e comparar diferentes metaeurísticas e diferentes métodos exatos.

6.1 Formulação clássica

A mais comum formulação MIP do GAP é dada por:

$$\min \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij},$$

sujeito a:

$$\begin{aligned} &\sum_{j=1}^{n} a_{ij} x_{ij} \leq b_i, \quad i \in I, \\ &\sum_{i=1}^{m} x_{ij} = 1, \quad j \in J, \\ &x \in \{0,1\}, \quad i \in I, j \in J. \end{aligned} \tag{6-1}$$

Onde:

- $I = (i_1, i_2, i_3, \ldots, i_m)$ é o conjunto de máquinas, $|I| = m$;
- $J = (j_1, j_2, j_3, \ldots, j_n)$ é o conjunto de tarefas, $|J| = n$;
- x_{ij} é a variável binária que indica se a tarefa j foi associada à máquina i;
- c_{ij} indica o custo de associar a tarefa j à máquina i;
- a_{ij} é o quanto dos recursos da máquina i são consumidos quando esta é associada à tarefa j;
- b_i é a quantidade total de recursos da máquina i;
- a primeira restrição exige que a capacidade de cada máquina não seja ultrapassada;
- a segunda restrição exige que cada tarefa esteja associada somente a uma máquina;
- a restrição de integralidade em x garante que uma tarefa não possa estar parcialmente associada a uma máquina.

A formulação clássica retratada no modelo (6-1), apesar de sua simplicidade, apresenta um problema: a solução de sua relaxação linear (onde se remove a restrição de integralidade das variáveis x) não gera bons *bounds* [30], dificultando assim a tarefa de algoritmos *branch and bound*. Uma formulação mais forte, com melhores *bounds*, foi proposta inicialmente por Savelsbergh [35] e pode ser vista como uma versão desassociada da formulação clássica.

6.2 Formulação com número exponencial de colunas (geração de colunas)

Para uma certa máquina i, seja $v_i^k = \{v_{i1}^k, v_{i2}^k, \dots v_{in}^k\}$ um vetor binário que representa uma das K_i possíveis associações válidas das n tarefas à máquina i, ou seja, uma solução viável para $\sum_{j=1}^{n} a_{ij} v_{ij}^k \leq b_{ij}$. Em outras palavras, $v_{ij}^k = 1$ se na k-ésima possível alocação da máquina i a tarefa j está alocada a i. O GAP pode então ser formulado como:

$$\min \sum_{i=1}^{m} \sum_{k=1}^{K_i} \left(\sum_{j=1}^{n} c_{ij} v_{ij}^k \right) y_i^k,$$

sujeito a:

$$\begin{aligned} &\sum_{i=1}^{m} \sum_{k=1}^{K_i} v_{ij}^k y_i^k = 1, \quad j \in J, \\ &\sum_{k=1}^{K_i} y_i^k \leq 1, \quad i \in I, \\ &y_i^k \in \{0,1\}, \quad k \in K_i, i \in I. \end{aligned} \tag{6-2}$$

Onde:

- I, J e c_{ij} têm o mesmo significado em relação ao modelo (6-1);
- v_{ij}^k indica, como descrito acima, se a tarefa j está associada à maquina i na k-ésima alocação possível;
- y_i^k é a variável binária que indica se a k-ésima alocação foi escolhida para a máquina i;
- a primeira restrição exige que cada tarefa esteja associada a exatamente uma máquina (mesmo que fracionalmente dividida entre várias alocações k diferentes);
- a segunda restrição exige que cada alocação k seja escolhida no máximo uma vez para cada máquina;
- a restrição de integralidade em y garante que as alocações não possam ser parcialmente escolhidas.

À primeira vista, resolver o modelo acima parece complicado: mesmo fazendo *branch and bound* nas variáveis y, não é possível resolver a relaxação linear diretamente, já que o número de possíveis alocações das n tarefas à máquina i (K_i) aumenta exponencialmente com n. Nesse caso, é necessário utilizar uma técnica chamada *geração de colunas*.

A técnica de geração de colunas, proposta inicialmente por Gilmore e Gomory [13] (baseada no algoritmo de decomposição de Dantzig-Wolfe), sugere que se trabalhe com somente algumas colunas de cada vez e que, quando necessário, se produzam mais colunas. Nesta formulação do GAP, uma coluna é um vetor v, uma associação de um conjunto de tarefas a uma máquina. Partindo de um conjunto de colunas inicial, resolve-se um sub-problema para encontrar novas colunas a serem adicionadas ao modelo. Esse problema é chamado de sub-problema de geração de colunas e, no caso do GAP, é o problema da mochila 0-1 (para o qual existem algoritmos de programação dinâmica com resultados excelentes). Quando não se consegue mais gerar colunas, ou seja, as colunas geradas tem um custo reduzido não-negativo, a solução ótima foi encontrada.

A grande vantagem da formulação apresentada nesta seção é que ela consegue excelentes *bounds* ao resolver a relaxação linear do GAP [30]. É importante notar que geração de colunas é somente uma técnica para resolver problemas lineares, e não problemas inteiros. Dessa maneira, ainda é preciso acoplar a geração de colunas a um algoritmo de *branch and bound*, resultando em um algoritmo chamado de *branch and price*. Como os *bounds* são melhores, espera-se que o B&B consiga terminar mais rapidamente.

6.3 Revisão da literatura do problema

Por ser um problema clássico, abordado pela primeira vez em 1976 [34], o GAP possui extensa literatura. Neste trabalho será revisada principalmente a literatura mais moderna do problema: inicialmente serão examinados dois métodos exatos, seguidos de três metaeurísticas. Além dos trabalhos examinados em detalhe nas seções seguintes, também pode-se mencionar como literatura relevante do problema o sistema de colônia de formigas de Lourenço e Serra [25], a busca tabu de Diaz e Fernández [8], o algoritmo genético de Chu e Beasley [4] e o *branch and bound* de Haddadi e Ouzia [18].

6.3.1 Savelsbergh (1997)

Savelsbergh [35] propõe um algoritmo *branch and price* para o GAP. Como mencionado, ele é o primeiro a propôr a formulação de particionamento de conjuntos descrita na seção 6.2, que é utilizada em um algoritmo *branch and price*. O trabalho experimentou diferentes possibilidades de *branching* e heurísticas de inicialização. O algoritmo foi testado em instâncias pequenas e apresentou bons resultados para a época.

6.3.2 Pigatti, Poggi e Uchoa (2005)

Pigatti, Poggi e Uchoa [30] partem da mesma formulação de particionamento de conjuntos e de um algoritmo *branch and price*. Notando que existe uma variação muito grande no número de iterações em algoritmos desta classe (mesmo para instâncias semelhantes), o trabalho propõe o uso de uma técnica para estabilizar a geração de colunas, simplificação de uma técnica apresentada em [9].

O trabalho apresenta também bons resultados para instâncias pequenas e resolveu à optimalidade algumas instâncias que ainda estavam em aberto na época. Além do algoritmo *stabilized branch-and-cut-and-price* (já que o trabalho propõe o uso de cortes, mas admite que eles não ajudaram a melhorar o algoritmo), o trabalho também é o primeiro a propôr uma versão das vizinhanças elipsoidais, como descrito no capítulo 5. O uso dos cortes permite encontrar uma excelente solução heurística para uma das mais difíceis instâncias do problema.

6.3.3 Feltl e Raidl (2004)

Um algoritmo genético fortemente baseado no de Chu e Beasley [4] é proposto no trabalho de Feltl e Raidl [11]. A base do algoritmo genético é a mesma entre os dois trabalhos, mas com algumas diferenças que fazem o segundo ter resultados consideravelmente melhores:

- uma inicialização das soluções através de dois métodos heurísticos: um que leva em conta quão atrativa é a alocação de uma tarefa a uma máquina (analisando a capacidade utilizada e o custo), outro que utiliza um arredondamento randômico da solução da relaxação linear do GAP;
- um novo algoritmo para selecionar as soluções para recombinação, baseado em quanto a solução está excedendo a capacidade das máquinas;
- uma nova forma de realizar mutação, onde um conjunto aleatório de tarefas é desalocado e posteriormente realocado utilizando heurísticas.

O algoritmo proposto é comparado somente em instâncias pequenas e com o algoritmo de Chu e Beasley [4], apresentando resultados superiores a este.

6.3.4
Yagiura, Ibaraki e Glover (2004)

Yagiura, Ibaraki e Glover [38] propõem um algoritmo de busca tabu para o GAP que apresenta algumas características inovadoras. Estas características, em conjunto, o fizeram na época o melhor algoritmo para o GAP. A primeira é que o algoritmo trabalha não somente com soluções válidas, mas também com soluções onde a restrição de capacidade das máquinas não é respeitada. As soluções inválidas (inviáveis) têm seu custo penalizado e esta penalidade é constantemente ajustada durante o algoritmo.

A segunda inovação introduzida por este trabalho é uma nova vizinhança, chamada de *Ejection Chain* (EC). Essa vizinhança consiste em trocas de alocações entre consecutivas máquinas, ou seja, é removida uma tarefa t_1 de uma máquina m_1, t_1 é então alocada a uma máquina qualquer m_2 e uma tarefa t_2 é retirada de m_2, a tarefa t_2 é então alocada a m_3 e assim sucessivamente. Essa vizinhança é uma generalização das vizinhanças mais comuns do problema: a vizinhança *swap*, onde a alocação de duas tarefas é trocada (chamada de *ejection chain* cíclica), e a vizinhança *shift*, onde a alocação de uma única tarefa é modificada.

A busca local utilizada neste trabalho consiste em consecutivas buscas em três vizinhanças: a vizinhança *shift*, a vizinhança *double shift* (uma EC de tamanho dois) e a vizinhança *long chain* (uma EC de tamanho qualquer). São utilizadas regras baseadas na relaxação Lagrangeana do GAP para controlar o tamanho destas vizinhanças. A lista tabu consiste em soluções que foram geradas nas últimas iterações.

A busca tabu proposta (chamada de *Tabu Search with Ejection Chains*, TSEC) consegue excelentes resultados em todas as instâncias testadas, em grande parte encontrando as melhores soluções para as instâncias mais difíceis. É fácil de perceber que este trabalho é, sem dúvida, o mais complexo trabalho na literatura moderna do GAP. Ele se utiliza de várias características únicas do problema, tornando interessante sua comparação com algoritmos gerais como o CPLEX e o VNSBra (seção 4.4).

6.3.5
Yagiura, Ibaraki e Glover (2006)

Utilizando idéias bastante semelhantes às usadas na busca tabu descrita na seção 6.3.4, Yagiura, Ibaraki e Glover [39] propõem um algorimo *Path Relink* para o GAP. Este algoritmo utiliza a mesma vizinhança de *Ejection Chains* e também trabalha com soluções inviáveis (novamente com um parâmetro controlado durante a busca).

O *Path Relink* proposto mantém um conjunto de soluções durante toda a busca. O propósito deste conjunto é manter soluções de qualidade mas, ao mesmo tempo, manter a diversidade da busca, guardando somente soluções diferentes umas das outras. A cada iteração, duas soluções deste conjunto são escolhidas aleatoriamente e é feito *relink* entre estas soluções. O PR gera várias soluções, que são então otimizadas por uma busca local EC e, dependendo da qualidade da solução e da diversidade do conjunto, talvez adicionadas ao conjunto de soluções.

Este algoritmo (chamado de *Path Relink with Ejection Chains*, PREC) consegue excelentes resultados, em geral iguais ou melhores que os do algoritmo TSEC, tornando-o melhor algoritmo da literatura para o GAP.

6.4 Instâncias

Existem cinco classes de instâncias clássicas para o GAP na literatura, denominadas instâncias A, B, C, D e E, originalmente usadas em [4, 24] e [27]. As instâncias das classes A e B são atualmente consideradas muito simples e, portanto, não serão abordadas neste trabalho. As classes de instância C, D e E são definidas da seguinte forma:

Classe C a_{ij} são inteiros escolhidos uniformemente no intervalo $[5, 25]$, c_{ij} são inteiros escolhidos uniformemente no intervalo $[10, 50]$ e $b_i = 0.8 \sum_{j \in J} a_{ij}/m$.

Classe D a_{ij} são inteiros escolhidos uniformemente no intervalo $[1, 100]$, $c_{ij} = 111 - a_{ij} + e_1$ onde e_1 é um inteiro escolhido uniformemente do intervalo $[-10, 10]$ e $b_i = 0.8 \sum_{j \in J} a_{ij}/m$.

Classe E $a_{ij} = 1 - 10 \ln e_2$ onde e_2 é um número escolhido uniformemente do intervalo $(0, 1]$, $c_{ij} = 1000/a_{ij} - 10e_3$ onde e_3 é um número escolhido uniformemente do intervalo $[0, 1]$ e $b_i = 0.8 \sum_{j \in J} a_{ij}/m$.

É importante notar que as classes D e E geralmente são as mais difíceis, já que nelas as variáveis a_{ij} e c_{ij} são inversamente correlacionadas. Em [38] são definidos três conjuntos de instâncias:

SMALL Total de 60 instâncias da classe C de tamanho n até 60. Essas instâncias e seus valores ótimos são retiradas da OR-Library [1].

[1] http://people.brunel.ac.uk/ mastjjb/jeb/orlib/gapinfo.html

MEDIUM Total de 18 instâncias das classes C, D e E de tamanho n até 200. As instâncias da classe C e D foram retiradas da OR-Lib e as instâncias da classe E estão disponíveis no site dos autores [2].

LARGE Total de 27 instâncias das classes C, D e E de tamanho n até 1600. Também disponíveis no endereço acima.

O conjunto SMALL é considerado muito simples e, portanto, também não será abordado neste trabalho. Algumas das instâncias dos conjuntos MEDIUM e LARGE também são triviais para os resolvedores MIP atuais e não serão utilizadas - mais detalhes no capítulo 8.

[2] http://www-or.amp.i.kyoto-u.ac.jp/ yagiura/gap/