Uma heurística híbrida em CPU-GPU baseada no método Scatter Search para resolver o problema de alocação generalizada

Danilo Santos Souza
Departamento de Computação, UFOP
danilo.gdc@gmail.com

Haroldo Gambini Santos
Departamento de Computação, UFOP
haroldo@iceb.ufop.br

Igor Machado Coelho
Instituto de Matemática e Estatística, UERJ
igor.machado@ime.uerj.br

RESUMO

No Problema de Alocação Generalizada, tarefas devem ser atribuídas a agentes com recursos limitados, a fim de minimizar o custo de alocação. Este problema tem várias aplicações na indústria e frequentemente aparece como subestrutura para outros problemas de otimização combinatória. Para aproveitar o poder computacional das unidades de processamento gráfico em uma metaheurística *Scatter Search*, propõe-se um método que gere eficientemente um pool de soluções usando busca tabu e um mecanismo de cadeia de ejeção. As características comuns são extraídas do pool e as soluções são combinadas para explorar um espaço restrito das soluções, com um modelo de programação binária. As instâncias clássicas variam de 100-1600 tarefas e 5-80 agentes, mas devido à grande quantidade de soluções ótimas encontradas pelo método, propomos novas instâncias de grande porte com até 9000 tarefas e 600 agentes. Os resultados indicam que o método é competitivo com algoritmos presentes na literatura.

PALAVRAS CHAVE. Scatter Search, Problema de Alocação Generalizada, Unidade de Processamento Gráfico.

Tópicos: SS3, MH, OC, OA

ABSTRACT

In the Generalized Assignment Problem, tasks must be allocated to agents with limited resources, in order to minimize allocation costs. This problem has several industrial applications and often appears as substructure of other combinatorial optimization problems. By harnessing the massive computational power of Graphics Processing Units - GPU in a Scatter Search metaheuristic framework, we propose a method that efficiently generates a solution pool using a Tabu list criteria and an Ejection Chain mechanism. Common characteristics are extracted from the pool and solutions are combined by exploring a restricted search space, as a Binary Programming model. Classic instances vary from 100-1600 jobs and 5-80 agents, but due to the big amount of optimal and near-optimal solutions found by our method, we propose novel large-sized instances up to 9000 jobs and 600 agents. Results indicate that the method is competitive with state-of-the-art algorithms in literature.

KEYWORDS. Scatter Search, Generalized Assignment Problem, Graphics Processing Unit

Paper topics: SS3, MH, OC, OA

1. Introdução

O Problema de Alocação Generalizada (PAG), originalmente proposto por Ross e Soland [1975], tem como objetivo encontrar uma alocação de custo mínimo dada uma quantidade de tarefas e agentes, de modo que uma tarefa pode ser alocada a um único agente satisfazendo as restrições de capacidade. O PAG é um problema NP-difícil [Sahni e Gonzalez, 1976], com aplicação em vários problemas da indústria.

Formalmente, o PAG pode ser definido da seguinte forma: dado n o número de tarefas e m o número de agentes, considerar o conjunto de tarefas $J = \{1, ..., n\}$ e o conjuntos de agentes $I = \{1, ..., m\}$. Ao alocar uma tarefa j ao agente i, é atribuído um custo d_{ji} e um consumo de recurso r_{ji}, de modo que a quantidade total de recursos disponíveis do agente i é denotado por b_i. A variável binária de decisão x_{ji} recebe 1 se a tarefa j é alocada para o agente i, e 0 caso contrário. Então, um modelo matemático pode ser formulado com a seguir:

$$Minimizar \sum_{j \in J} \sum_{i \in I} d_{ji} x_{ji} \tag{1}$$

Sujeito a:

$$\sum_{j \in J} r_{ji} x_{ji} \leq b_i \quad \forall i \in I \tag{2}$$

$$\sum_{i \in I} x_{ji} = 1 \quad \forall j \in J \tag{3}$$

$$x_{ji} \in \{0, 1\} \quad \forall i \in I, j \in J \tag{4}$$

Várias abordagens têm sido propostas para resolver este problema, incluindo métodos exatos e heurísticos. Os trabalhos propostos na literatura utilizam de algoritmos heurísticos como Algoritmos Genéticos [Chu e Beasley, 1997; Wilson, 1997; Feltl e Raidl, 2004], *Simulated Annealing* [Osman, 1995] e Busca Tabu [Laguna et al., 1995; Osman, 1995; Dıaz e Fernández, 2001; Higgins, 2001]. Além disso, alguns algoritmos exatos foram propostos: o algoritmo *branch-and-price* por Savelsbergh [1997], o algoritmo *Cutting Plane* por Avella et al. [2010], e *Branch-and-Cut* por Nauss [2003], onde soluções ótimas foram encontradas para muitas instâncias com até 200 tarefas e 20 agentes.

Chu e Beasley [1997] apresenta uma heurística para o problema de alocação generalizada baseada no algoritmo genético (AG). Em adição aos procedimentos padrões do AG, a heurística proposta apresenta uma representação não binária que satisfaz automaticamente as restrições de alocação das tarefas, uma avaliação de aptidão-inviabilidade e um operador heurístico que ajuda melhorar o custo e viabilidade da solução. Os resultados computacionais mostram que o AG proposto é capaz de encontrar solução ótima para vários problemas de tamanho médio (de 100 a 200 tarefas).

No trabalho proposto por Posta et al. [2012] um algoritmo exato para resolver o PAG é implementado. O problema tratado com otimização é reformulado em uma sequência de problemas de decisão, e regras de fixação de variáveis são usadas para resolver o problema eficientemente. Os problemas de decisão são resolvidos por um método *Depth-First Lagrangian Branch-and-Bound*, e as regras de fixação de variáveis de modo a podar a árvore de busca melhorando o desempenho do método. Estas regras baseiam-se em custos reduzidos de *Lagrange* que são calculados utilizando um algoritmo de programação dinâmica. A abordagem de resolver sucessivamente um problema de decisão para cada subproblema gerada é muito rápida na prática, uma vez que começam a partir de um bom limite inicial. Os resultados mostram que o método proposto é capaz de encontrar soluções ótimas para instâncias grandes (instâncias com quantidade de tarefas maiores de 200 e menores 1600).

Para mais detalhes sobre métodos de solução para o PAG, Cattrysse e Van Wassenhove [1992] e Öncan [2008] apresentam revisões extensivas de aplicações e algoritmos heurísticos e exatos existentes.

O uso de técnicas de otimização para obter soluções próximas às ótimas para os problemas NP-difícil muitas vezes demandam tempo elevado de processamento, especialmente ao tratar de problemas grandes. No entanto, técnicas computacionais de alto desempenho podem ser aplicadas para reduzir o tempo computacional e melhorar o desempenho do algoritmo na resolução do problema. Recentemente, o uso das Unidades de Processamento Gráfico (*Graphics Processing Unit GPU*) aumentou com o objetivo de acelerar os cálculos em muitas áreas de aplicação com equipamentos computacionais de baixo custo. Implementações bem sucedidas em GPU pode obter um *speed up* de até 100 vezes mais do que uma implementação sequencial em CPU [Kirk e Wenmei, 2012]. No entanto, devido a detalhes específicos da GPU, como o paradigma "Única Instrução e Múltiplas Threads" (*Single Instruction Multiple Threads*, SIMT), os algoritmos clássicos geralmente precisam ser totalmente reescritos para explorar a enorme quantidade de núcleos de processamento e a largura de banda abundante da memória do dispositivo.

Este artigo descreve uma metaheurística híbrida inspirada na metaheurística *Scatter Search* implementada para executar em plataforma heterogênea CPU-GPU com objetivo de produzir soluções de alta qualidade para o PAG. O método híbrido proposto inclui uma implementação da metaheurística Busca Tabu (BT) combinada com um método exato denominado *Binary Programming Pool Search (BPPS)*.

O uso da BT combinada com *scatter search* é proposto por se apresentar promissor para resolver outros problemas de otimização difíceis e por sua estrutura muito simples, de modo a ser muito adequado para a implementação baseada em GPU. O método inclui a geração de cadeias de ejeções factíveis com implementação baseada em GPU, de modo que uma vizinhança grande é explorada em paralelo na GPU a cada iteração da BT. A execução da heurística BT em GPU permite a geração de várias soluções em tempo computacional reduzido, facilitando a composição de um conjunto de soluções elite (pool de soluções) utilizadas como conjunto de referência para o método BPPS. O BPPS permite uma processo de busca rápido por usar regras para fixar variáveis de decisão usando as características mais comuns do pool de soluções e assim reduzindo o espaço de busca do problema.

O artigo está organizado da seguinte forma. A seção 2 discute a concepção e implementação do método híbrido proposto. A seção 3 detalha as experiências computacionais projetadas para avaliar e analisar os resultados dos experimentos. A seção 4 relata as conclusões, descobertas e os trabalhos futuros para melhorar o método.

2. Método Proposto

Nessa seção apresentamos a abordagem proposta para o Problema de Alocação Generalizada. O método proposto é uma abordagem híbrida composta por uma adaptação do *scatter search* usando técnicas da busca tabu e *binary programming pool search*.

A heurística *Scatter Search (SS)* é uma metaheurística evolucionária que foi inicialmente introduzida por Glover [1977] com o objetivo de explorar um espaço de soluções a partir da evolução de um conjunto de boas soluções chamado de *conjunto de referência* [Resende et al., 2010]. Para uma introdução geral ao SS, propõe-se os seguintes trabalhos: Glover [1977]; Martí et al. [2006, 2009]. O Algoritmo 1 apresenta a estrutura principal do método.

Na primeira fase da abordagem um algoritmo guloso aleatório é usado para gerar um conjunto de soluções iniciais distintas. Para melhorar a solução, a abordagem da busca tabu é implementada usando o paralelismo na GPU, onde o método é executado diversas vezes em paralelo (a geração atual de GPUs normalmente tem milhares de núcleos de processamento) gerando várias soluções, onde as melhores soluções são armazenadas em um *pool* (conjunto de referência).

Dado o pool de soluções, a solução com melhor função objetivo é usada com entrada da próxima fase. Para cada variável de decisão, um valor h_{ji} é calculado dado o número de vezes que

Algoritmo 1 Método Proposto

1: Construir uma solução inicial P com tamanho de $|P|$ usando o Método Guloso Aleatório.
2: Construir o conjunto de referência usando BT em GPU para melhorar as soluções geradas etapa 1.
3: Selecionar a melhor solução do conjunto de referência.
4: Utilizar o BPPS na solução selecionada na etapa 3 para executar a melhoria, atualização do conjunto de referência, gerar os subconjuntos e combinar as soluções.

a variável x_{ji} aparece com o valor 1 em soluções do conjunto de referência.

O método *Binary Program Pool Search* propõe regras de fixação das variáveis com os valores de h_{ji}, com o objetivo de reduzir o espaço de buscas para o modelo de programação binária e propagar as características comuns para as novas soluções, onde este processo é repetido aumentando gradualmente o espaço de busca sempre que não é possível obter uma melhoria na solução.

Os métodos usados são apresentados em detalhes nas subseções seguintes.

2.1. Método Guloso Aleatório

Um método guloso aleatório é usado para gerar as soluções iniciais. O método proposto considera um peso para priorizar as alocações que permitirá a redução do custo final. Esse peso é calculado em relação ao custo c_{ji} e o consumo de recursos r_{ji}: $W_{ij} = w_1 c_{ji} + w_2 r_{ji}$. A alocação é realizada pela tarefa, isto é, uma tarefa j deverá ser alocada ao agente i com o menor peso W_{ji} que satisfaz a restrição de capacidade, as tarefas são escolhidas sequencialmente.

Um pseudocódigo do método é apresentado no Algoritmo 2.

Algoritmo 2 Método Guloso

Entrada: w_1, w_2 ▷ Pesos atribuídos ao custo e ao recurso.
c_{ji}, r_{ji} ▷ Custo e recursos de alocação da tarefa j para o agente i.
b_i ▷ Capacidade disponível do agente i
Saída: x ▷ Solução Inicial

$W_{ji} \leftarrow w_1 c_{ji} + w_2 r_{ji} \ \forall j \in N, i \in M$
Para $k := 1$ **até** n **faça** ▷ $n = |N|$
  Para $z := 1$ **até** m **faça** ▷ $m = |M|$
    $a \leftarrow \underset{(i) \in M}{\textbf{Argmin}} \{W_{ki}\}$
    se $\sum_{i \in I} x_{ki} = 0 \wedge r_{ka} + \sum_{j \in J} x_{ja} r_{ja} \leq b_a$ **então**
      $x_{ka} \leftarrow 1$
    Se não
      $W_{ka} \leftarrow \infty$
    Fim se
  Fim para
Fim para
Retorne x

2.2. Busca Tabu

A Busca Tabu (BT) é uma metaheurística baseada em busca local ou busca em vizinhança que admite piora nas soluções, proposto por Glover [1989, 1990] para resolver problemas complexos de otimização combinatória. A busca tabu é um procedimento iterativo que objetiva explorar o espaço de solução do problema a fim de realizar movimentos sucessivos a partir de uma solução x para outra solução x' na vizinhança $N(X)$. Contudo, apenas este procedimento de busca não é suficiente para escapar de ótimos locais, uma vez que pode haver um retorno a uma solução gerada anteriormente. Para evitar isso, o algoritmo usa o conceito de lista tabu.

A lista tabu (memória adaptativa) pode ser utilizada como memória de curto prazo ou memória de longo prazo. A memória de curto prazo restringe a busca proibindo certos movimentos que permitem ultrapassar grandes locais, prevenindo ciclos e direcionando a busca para regiões não exploradas. O uso da memória de longo prazo objetiva proporcionar o retorno da busca por regiões consideradas promissoras. Para isso é usada uma medida da frequência de atributos das melhores soluções encontradas durante a busca, chamadas de soluções elite [Glover e Laguna, 2013].

Cada iteração do algoritmo busca tabu consiste das principais tarefas: gerar vizinhança, avaliação da vizinhança, escolher a melhor solução contida na vizinhança e atualizar a lista tabu, e finalmente mover a nova solução [Czapiński e Barnes, 2011]. No trabalho, propõe-se uma implementação paralela que usa a GPU para as fases de geração e avaliação da vizinhança, de modo que as permutações que formam a vizinhança são armazenadas na memória. Onde requer a transferência de dados entre GPU e CPU em cada iteração.

O uso da GPU é realizado por meio de implementações de *kernels* que são executados em um conjunto definido de *threads*. Teoricamente, todas as threads executam o mesmo código *simultaneamente* e tem um identificador único. Na prática, *threads* que pertencem a um mesmo *warp* de 32 *threads* irá executar o mesmo código (este valor pode variar em algumas microarquiteturas GPU). As *threads* são divididas em grupos chamados *blocos*, que são executados por diferentes *Stream Multiprocessors* (SMs). A dimensão máxima para o número de *threads* e *blocos* é limitada pela GPU, mas pode ser configurada pelo usuário a fim de otimizar o tempo de execução de cada *kernel* [Sulewski et al., 2011]. Em outras palavras, o mesmo método pode ser executado por várias *threads* e estes, por sua vez, agrupados em vários blocos. Vale ressaltar que entre as *threads* de um mesmo bloco é possível compartilhar dados através da ”memória compartilhada”.

A vizinhança é gerada a partir do método de cadeia de ejeção apresentado na subseção 2.3, onde cada *thread* gera um certo valor de cadeias de ejeção e a melhor cadeia será armazenada na memória compartilhada e assim propagados para a fase seguinte. Cada *bloco* da arquitetura GPU consiste de várias *threads*, uma única *thread* de cada bloco será responsável por avaliar as melhores cadeias geradas armazenadas na memória compartilhada, atualizando a solução com a melhor cadeia para esse bloco. Cada bloco irá executar um procedimento de busca tabu, e no final irá gerar um conjunto de referência (pool de soluções) com as melhores soluções geradas por cada bloco. A memória de curto prazo é utilizada no método da Cadeia de Ejeção, por outro lado, a memória de longo prazo é aplicada como um critério de desempate na avaliação das soluções. A memória de longo prazo é compartilhada por todos os blocos.

2.3. Cadeia de Ejeção

Os métodos de cadeia de ejeção são métodos de profundidade variável que geram sequências de movimentos simples, a fim de conseguir movimentos compostos mais complexos. Em outras palavras, uma cadeia de ejeção de nível L consiste de sucessivas operações realizadas em um conjunto de elemento, onde a l^n operação muda o estado de um ou mais elementos da solução (serão assim ejetadas na operação $l+1$). Os estados mudam, assim como os avanços na cadeia de ejeção, e esses dependem do efeito cumulativo das etapas anteriores [Glover e Rego, 2006].

Nesse artigo, o número de operações realizadas na solução s é limitada pelo tamanho de k. Os movimentos são baseados no tamanho de k, de tal modo que:

- para $k = 1$, uma tarefa j é selecionada aleatoriamente e será alocada a um novo agente w, essa nova alocação deve respeitar a capacidade do agente w;
- para $k = 2$, duas tarefas j_1 e j_2 são escolhidas aleatoriamente, onde i_1 e i_2 são os agentes alocados para realizar as tarefas j_1 e j_2, respectivamente. Onde, a tarefa j_1 irá ser alocada ao agente i_2 se possível, e a job j_2 ao agente i_1;
- para $k \geq 3$, a cadeia de ejeção corresponde a uma sequência de trocas para diferentes agentes e tarefas, onde tarefas são selecionadas aleatoriamente $j_1, j_2, ..., j_k$ e seus respectivos agentes

denominados $i_1, i_2, ..., i_k$. O movimento deve sempre obedecer as restrições de capacidade do problema, cada tarefa j_k, com exceção da última, irá ser alocada no agente i_{k+1}, ou seja, a tarefa j_1 será alocada ao agente i_2, a tarefa j_2 será alocada ao agente i_3, sucessivamente até que a tarefa $j_{(k-1)}$ seja atribuída ao agente i_k, e finalmente a tarefa j_k irá ser alocada ao agente i_1. O processo é construído iterativamente para verificar cadeias de tamanho 2 a k, retornando a cadeia que irá obter a maior melhora da solução (que irá contribuir para uma diminuição na função objetivo). Portanto, a cadeia retornada pode ter tamanho contido entre 2 e k.

Uma das principais diferenças na abordagem proposta por Yagiura et al. [2006] é que todos os movimentos usados na proposta desse trabalho devem gerar soluções factíveis. Outra diferença é que no trabalho de Yagiura et al. [2006] a probabilidade de escolha das tarefas não é a mesma, isso favorece um grupo específico de tarefas, e nosso algoritmo as probabilidades são iguais.

2.4. Conjunto de Referência (Pool de soluções)

Conforme afirmado anteriormente, o conjunto de referência é gerado a partir da execução da busca tabu na GPU. O conjunto de referência é usado em fases posteriores para calcular uma matriz de referência contendo informação da similaridade entre as soluções assim como determinar quais as características serão propagadas para as próximas soluções. O número de soluções armazenadas no pool de soluções foi determinado pelo número máximo de blocos permitidos pela arquitetura da GPU.

2.5. Binary Programming Pool Search

A proposta do *Binary Programming Pool Search* (BPPS) é baseada no *Relaxation Induced Neighborhood Search (RINS)* [Danna et al., 2005] e o uso de cortes elipsoidal [Pigatti et al., 2005], de modo a utilizar as características do pool de soluções.

Inicialmente, uma solução do pool de soluções é escolhida como entrada para a formulação de Programação Binária (PB), a solução com melhor função objetivo é escolhida. Para determinar as características comuns no pool de soluções uma matriz de residência h_{ji} é calculada, onde para cada valor de x_{ji} o valor de h_{ji} contêm o número de vezes que $x_{ji} = 1$ nas soluções contidas no pool de soluções, isto é, dado x_{ji} igual a 1 em w soluções do pool, o valor de h_{ji} irá ser definido com w.

Com a solução inicial e a matriz de residência h_{ji} das variáveis de decisão, o BPPS usa esses parâmetros para realização dos cortes elipsoidais (de modo a reduzir o espaço de soluções). A cada iteração a restrição 5 é adicionada ao modelo, onde o valor de α_{ji} é calculado a partir da normalização dos dados contidos na matriz h_{ji}. O valor β é a quantidade de folga que define a vizinhança elipsoidal e é definido com um parâmetro e incrementado a cada iteração que não houver melhora na solução.

$$\sum_{j \in N, i \in M | x_{ji=1}} \alpha_{ji}(1 - x_{ji}) \leq \beta \tag{5}$$

A restrição apresentada na Equação 6 limita o valor da função objetivo pelo valor da solução incumbente, e é atualizada a cada iteração.

$$\sum_{j \in N, i \in M} x_{ji} c_{ji} \leq F_o^{'} \tag{6}$$

Cada execução do solver é limitada a 2 situações, se houver melhora na solução incumbente o número de 500 nós explorados sem melhora é usado como critério de parada, caso contrário um tempo limite é usado. Um pseudocódigo da formulação é apresentado no Algoritmo 3.

Algoritmo 3 BPPS

Entrada: x ▷ Solução, $x_{ji} = 1$ se a tarefa j é alocada ao agente i
h_{ji} ▷ Matriz residência
N ▷ Conjunto com n tarefas
M ▷ Conjunto com m agentes
Parêmetros: β ▷ Folga para definir a vizinhança elipsoidal
$tempoLimite$ ▷ Limite de tempo para o BPPS
Saída: $x^{'}$ ▷ Melhor solução encontrada

1: $t \leftarrow \beta$
2: **Enquanto** $tempo \leq tempoLimite \wedge k \leq |N|$ **faça**
3: $m_1 =$ **MAX**$(h_{ji})\ \forall i \in N, j \in M |\ x_{ji} = 1$;
4: $m_2 =$ **MIN**$(h_{ji})\ \forall i \in N, j \in M |\ x_{ji} = 1$;
5: $\alpha_{ji} \leftarrow (h_{ji} - m_2)/(m_2 - m_1)\ \forall i \in N, j \in M$;
6: $F_o^{'} \leftarrow f(x)$;
7: $x^{'} \leftarrow ExecutaSolver(x, \alpha_{ji}, F_o^{'}, \beta, tempoLimite - tempo)$;
8: **se** $f(x^{'}) < f(x)$ **então**
9: $h_{ji} \leftarrow h_{ji} + 1, \forall x_{ji} = 1$;
10: $\beta \leftarrow t$;
11: **Se não**
12: $\beta \leftarrow \beta + 1$;
13: **Fim se**
14: $x \leftarrow x^{'}$
15: **Fim Enquanto**
16: **Retorne** $x^{'}$

3. Resultados Computacionais

Nessa seção, é avaliada a performance da heurística híbrida em GPU baseada no *Scatter Search*, nomeada de GSS.

A plataforma utilizada para os experimentos principais é composta por uma CPU Intel i7 3.40GHz, com 24GB de memória, e uma GPU GeForce GTX 780, com 2304 núcleos e 2GB de memória global. Vale ressaltar que 12 núcleos de CPU foram utilizados para a fase do BPPS. A metaheurística BT foi implementada em C com CUDA SDK 7.5. O BPPS foi codificado em C++ e linguagem AMPL, resolvido pelo solver Gurobi versão 7.0.2.

As instâncias usadas para os experimentos são divididas em 4 tipos chamadas de tipo C, D e E propostas por Chu e Beasley [1997] e Laguna et al. [1995]. As instâncias do tipo G foram propostas nesse trabalho para lidar com problemas maiores, com 10 vezes mais agentes e tarefas (todas as instâncias estão disponíveis na página dos autores [1]). Instâncias desses tipos são geradas da seguinte maneira:

- **Tipo C:** r_{ji} são números inteiros aleatórios no intervalo uniforme $[5, 25]$, d_{ji} são números inteiros aleatórios em $[10, 50]$, e $b_i = 0.8 * (\sum_{j \in J} r_{ji})/m$.
- **Tipo D:** r_{ji} são números inteiros aleatórios em $[1, 100]$, $d_{ji} = 111 - r_{ji} + e_1$, onde e_1 são inteiros aleatórios em $[-10, 10]$, e $b_i = 0.8 * (\sum_{j \in J} r_{ji})/m$.
- **Tipo E:** $r_{ji} = 1 - 10 * ln\ e_2$, onde e_2 são números reais aleatórios contidos em $(0, 1]$, $d_{ji} = 1000/r_{ji} - 10 * e_3$, onde e_3 são números reais entre $[0, 1]$, and $b_i = 0.8 * (\sum_{j \in J} r_{ji})/m$.
- **Tipo G:** r_{ji} são inteiros aleatórios em $[1, 100]$, $d_{ji} = 111 - r_{ji} + e_4$, onde e_4 são números inteiros aleatórios em $[-10, 10]$, e $b_i = 1.75 * (\sum_{j \in J} r_{ji})/m$.

[1] https://drive.google.com/open?id=0B20uFG9WVmWMM2t1MlIxNG5aNW8

Os testes utilizam diversos tamanhos de instâncias para o problema, essas foram classificadas em três conjuntos:

- **Médio:** Total de 18 instâncias dos tipos C, D e E com número máximo de 200 tarefas. Esse conjunto é composto por combinações de 100 e 200 tarefas com 5, 10 e 20 agentes.
- **Grande:** Total de 27 instâncias dos tipos C, D, e E com número máximos de 1600 tarefas. Esse conjunto é composto por combinações de 400, 900 e 1600 tarefas com 10, 15, 20, 30, 40, 60 e 80 agentes.
- **Muito Grande:** Total de 12 instâncias do tipo G com número máximos de 9000 tarefas, todas geradas e propostas nesse trabalho. Esse conjunto é composto por combinações de 1000, 2000, 4000 e 9000 tarefas com 50, 100, 200, 300, 400 e 600 agentes.

Como um parâmetro da metaheurística GBT(Busca tabu em GPU), o tamanho da lista tabu foi definido a partir de uma calibração offline, onde vários tamanhos de lista foram testados (5, 10, 15, 20, $(0.5, 0.75, 1, 1.25, 1.4, 1.5, 1.6) * (|N|/(max(k) + 1)\ )\ )$, porém o que apresentou melhor resultado foi $1.25 * (|N|/(max(k) + 1)\ )$, e como critério de parada foi utilizado o tempo de 15 segundos sem melhora na solução incumbente. O número de cadeias de ejeção geradas por *thread* foi definido para instâncias médias e grandes a partir de testes com valores de 1, 5, 10, 15 e 20, no entanto a melhor performance em relação a *custo-tempo* foi obtida com valor 5. Para as instâncias muito grandes, o valor do número de cadeias foi definido em 100. O tamanho k da cadeia de ejeção foi calibrada com os valores 1, 5, 10, 15 e 20 obtendo melhor desempenho para a cadeia de tamanhos 1 e 10. O parâmetro β do BPPS foi definido com 2 para as instâncias médias e grandes e $0.02 * |N|$ para as instâncias muito grandes, dado os testes realizados com valores 1, 2, 3, 4, 5, 6, 8, 10, 11, $0.01 * |N|$, $0.02 * |N|$ e $0.002 * |N|$. Para o método BPPS foi utilizado como critério de parada o valor máximo de 10 minutos ou provar a otimalidade da solução.

A Tabela 1 apresenta uma comparação entre o algoritmo promissor de Busca Tabu presente na literatura para o PAG, nomeado de BT Yagiura et al. [2006], e os algoritmos propostos GSS e GBT, usando as instâncias médias e grandes. Os métodos foram executados 10 vezes e os valores das melhores soluções são apresentados para comparação com o *Limite Inferior* de cada instância do problema. A GBT consiste na primeira fase da GSS, onde uma busca tabu é realizada com o uso da GPU, com objetivo de construir um conjunto de referência incluindo as soluções de boa qualidade. A segunda fase do GSS consiste no método BPPS executando sobre um *solver* exato, que tenta obter e provar a otimalidade da melhor solução atual, de modo a incluir as novas soluções encontradas no processo ao conjunto de referência.

Quando os gaps médios são comparados, o GSS é capaz de obter valores mais baixos para os grupos C e E (C: 0.03%, D: 0.23% e E: 0.02%) quando comparado com o BT (C: 0.04%, D: 0.20% e E: 0.04%). O GSS é capaz de encontrar 21 melhores soluções e 14 soluções iguais (em negrito) das 45 instâncias, em 15 soluções foi possível provar a otimalidade (soluções marcadas com asterisco).

De fato, uma vez que os gaps são bastante baixas quando calculadas sobre um *Limite Inferior* para o PAG, é muito provável que a maioria das soluções atuais mais conhecidas já sejam ótimas (embora ainda não comprovadas). O grupo de instâncias D ainda é bastante desafiador para o GSS, consumindo uma maior parcela dos tempos computacionais, ao mesmo tempo em que fornece soluções com maiores lacunas. Logo se fez necessário a proposta de novas instâncias para verificar melhor o desempenho do método.

Na Tabela 2 os resultados dos experimentos computacionais para as instâncias do tipo G são apresentados. O algoritmo GSS obtém soluções com gap entre 0.6% e 9.4% e gap médio de 3.4%. O método GBT foi capaz de gerar soluções de qualidade para o conjunto de referência, obtendo um gap médio de 6.8%. Devido ao tamanho das instâncias o tempo computacional foi

Tabela 1: Experimentos computacionais para a base de dados clássica

Inst.	**Limite**	**GSS (GBT + BPPS)**			**GBT**			**BT**		
	Inferior	**Best**	**gap**	**Tempo**	**Best**	**gap**	**Tempo**	**Best**	**gap**	**Tempo**
C.100.5	1930	**1931***	0.05	26.3	**1931**	0.05	23.9	**1931**	0.05	0.6
C.100.10	1400	**1402***	0.14	42.5	1404	0.29	32.8	**1402**	0.14	3.0
C.100.20	1242	**1243***	0.08	48.7	1252	0.81	37.9	**1243**	0.08	21.6
C.200.5	3455	**3456***	0.03	31.1	3475	0.58	21.3	**3456**	0.03	3.7
C.200.10	2804	**2806***	0.07	68.7	2818	0.50	30.7	**2806**	0.07	100.5
C.200.20	2391	**2391***	0.00	123.6	2408	0.71	54.4	2392	0.04	137.4
C.400.10	5596	**5597***	0.02	83.7	5629	0.59	63.8	**5597**	0.02	105.8
C.400.20	4781	**4782***	0.02	133.5	4827	0.96	49.0	**4782**	0.02	130.4
C.400.40	4244	**4244***	0.00	316.9	4279	0.82	62.0	**4244**	0.00	157.6
C.900.15	11339	**11341**	0.02	744.2	11431	0.81	71.1	**11341**	0.02	759.6
C.900.30	9982	**9983**	0.01	332.7	10087	1.05	13.3	9985	0.03	720.0
C.900.60	9325	**9327**	0.02	917.6	9425	1.07	150.6	9328	0.03	704.7
C.1600.20	18802	**18803**	0.01	831.4	18948	0.78	138.1	**18803**	0.01	691.6
C.1600.40	17144	**17146**	0.01	873.6	17340	1.14	143.1	17147	0.02	2103.7
C.1600.80	16284	**16288**	0.02	894.8	16456	1.06	203.2	16291	0.04	4317.2
C Média			**0.03**	**364.61**		**0.75**	**73.0**		**0.04**	**663.8**
D.100.5	6350	**6353***	0.05	151.3	6449	1.56	33.2	6357	0.11	62.9
D.100.10	6342	**6355**	0.20	510.0	6541	3.14	24.1	6358	0.25	107.2
D.100.20	6177	6229	0.84	209.8	6345	2.72	76.3	**6221**	0.71	111.0
D.200.5	12741	**12745**	0.03	691.3	12889	1.16	93.7	12746	0.04	95.5
D.200.10	12426	**12437**	0.09	1203.3	12688	2.11	116.2	12446	0.16	129.2
D.200.20	12230	**12267**	0.30	1155.2	12601	3.03	57.9	12284	0.44	120.7
D.400.10	24959	**24969**	0.04	616.0	25457	2.00	220.3	24974	0.06	16.1
D.400.20	24561	24621	0.24	500.6	25354	3.23	201.7	**24614**	0.22	81.3
D.400.40	24350	24506	0.53	416.6	25081	3.00	114.9	**24463**	0.46	165.2
D.900.15	55403	**55430**	0.05	1585.6	56888	2.68	218.2	55435	0.06	112.9
D.900.30	54833	54970	0.25	873.5	56713	3.43	271.0	**54910**	0.14	234.4
D.900.60	54551	54782	0.42	1438.9	56089	2.82	237.8	**54666**	0.21	833.8
D.1600.20	97823	97887	0.07	2527.7	100067	2.29	419.9	**97870**	0.05	143.0
D.1600.40	97105	97260	0.16	1938.7	99965	2.95	322.2	**97177**	0.07	1294.0
D.1600.80	97034	97316	0.29	587.1	99836	2.89	281.2	**97109**	0.08	4795.4
D Média			**0.23**	**960.3**		**2.60**	**179.2**		**0.20**	**553.5**
E.100.5	12673	**12681***	0.06	67.6	12868	1.54	58.4	12682	0.07	39.9
E.100.10	11568	**11577***	0.08	132.4	11879	2.69	45.8	**11577**	0.08	31.6
E.100.20	8431	**8436**	0.06	140.1	8938	6.01	59.1	8443	0.14	90.4
E.200.5	24927	**24930***	0.01	85.1	25228	1.21	78.4	**24930**	0.01	20.0
E.200.10	23302	**23307***	0.02	157.8	23518	0.93	95.9	**23307**	0.02	34.1
E.200.20	22377	**22379***	0.01	115.4	22814	1.95	24.0	22391	0.06	209.3
E.400.10	45745	**45746**	0.00	72.9	46135	0.85	15.8	**45746**	0.00	260.7
E.400.20	44876	**44877**	0.00	143.0	45436	1.25	20.9	44882	0.01	212.9
E.400.40	44557	**44570**	0.03	202.9	45570	2.27	95.4	44589	0.07	217.7
E.900.15	102420	102426	0.01	103.9	103643	1.19	18.0	**102423**	0.00	157.4
E.900.30	100426	**100431**	0.00	501.9	101354	0.92	16.8	100442	0.02	758.9
E.900.60	100144	**100171**	0.03	644.0	104007	3.86	42.6	100185	0.04	483.7
E.1600.20	180642	180654	0.01	616.6	182784	1.19	16.0	**180647**	0.00	1683.2
E.1600.40	178293	**178307**	0.01	620.0	179571	0.72	16.5	178311	0.01	2214.7
E.1600.80	176816	**176846**	0.02	624.6	179740	1.65	19.5	176866	0.03	2473.1
E Média			**0.02**	**281.9**		**1.88**	**41.5**		**0.04**	**592.5**
Média			**0.09**	**535.60**		**1.74**	**97.9**		**0.09**	**603.2**

*Indica que uma solução ótima foi encontrada (e provada) pelo modulo BPPS.

aumentado. Quatro das doze instâncias do tipo G o método BPPS não obteve melhoria nas soluções encontradas pelo GBT, geralmente ocorrendo nas instâncias com mais de 4 mil tarefas.

Finalmente, a configuração da GPU foi definida para 12 *blocos de threads* (objetivando maximizar a performance da GPU), cada uma realiza uma busca tabu independente e contribui com uma única solução para o pool (logo o conjunto de referência é composto por 12 soluções). Essa configuração obteve uma aceleração de 3 a 8 vezes, quando comparado a implementação pura em CPU do modulo GBT.

Tabela 2: Experimentos computacionais para a base de dados proposta

Inst.	**Limite**	**GSS (GBT+ BPPS)**			**GBT**		
	Inferior	**Best**	**gap**	**Tempo**	**Best**	**gap**	**Tempo**
G - 1000 - 50	13484	**13587**	0.8	678.6	14569	8.0	78.6
G - 1000 - 100	12802	**13258**	3.6	702.2	14400	12.5	102.2
G - 1000 - 200	12636	**13504**	6.9	696.0	14968	18.5	96.0
G - 2000 - 50	26794	**26942**	0.6	670.7	28456	6.2	70.7
G - 2000 - 100	26068	**26784**	2.7	675.6	27674	6.2	75.6
G - 2000 - 200	25094	**27457**	9.4	741.5	27554	9.8	141.5
G - 4000 - 100	50873	**51691**	1.6	714.1	52783	3.8	114.1
G - 4000 - 200	50345	**52519**	4.3	866.5	**52519**	4.3	266.5
G - 4000 - 400	50219	**53508**	6.5	1173.2	**53508**	6.5	573.2
G - 9000 - 150	113399	**115424**	1.8	851.8	116173	2.4	251.8
G - 9000 - 300	113176	**115248**	1.8	1124.1	**115248**	1.8	524.1
G - 9000 - 600	113238	**115406**	1.9	1219.6	**115406**	1.9	619.6
G Média			**3.5**	**842.8**		**6.8**	**242.8**

4. Conclusão

Esse trabalho apresenta uma Heurística Híbrida em GPU inspirada na metaheurística *Scatter Search* a fim de lidar com o Problema de Alocação Generalizada, ou PAG. O algoritmo proposto (chamado de GSS), consiste de 2 fases integradas, onde na primeira fase nomeada GBT, 12 buscas tabu independentes são realizadas em uma GPU para formar um pool de 12 soluções de alta qualidade. Este pool é posteriormente passado para o módulo BPPS, que consiste de um *solver* exato para o modelo de programação binária considerando uma versão limitada do problema original. A técnica proposta conseguiu encontrar as 21 melhores soluções, quando comparadas com o algoritmo de Busca Tabu da literatura para o PAG, provando também a otimalidade de 15 soluções. A média dos gaps são menores que as comparadas com a literatura, quando comparado com um *Limite Inferior* do problema. 12 novas instâncias foram propostas para esse problema clássico e desafiador, incorporando características dos problemas mais difíceis do grupo D, nomeado de tipo G. Com essas instâncias, foi possível verificar a eficiência do método proposto quando se trata de casos muito grandes.

Embora as lacunas obtidas sejam bastante baixas para a maioria dos casos, propomos as seguintes melhorias futuras para o algoritmo GSS. Muitas otimizações de baixo nível podem ser aplicadas no GBT em relação ao desempenho da GPU, como: redução do número de registradores, aumento do uso da memória compartilhada, armazenamento de dados de somente leitura em memórias constantes específicas, parâmetros de inicialização do kernel que aumentam a memória e calculam o *throughput*, reduzindo a ocupação geral.

Agradecimentos

Esta Pesquisa foi apoiada pelo Conselho Nacional de Desenvolvimento Científico e Tecnologico - CNPq, pela Fundação de Apoio a Pesquisa do Estado de Minas Gerais - FAPEMIG, pela Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - CAPES, pela Fundação de Amparo à Pesquisa do Estado do Rio de Janeiro - FAPERJ e pela Universidade Federal de Ouro Preto - UFOP.

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.

Cattrysse, D. G. e Van Wassenhove, L. N. (1992). A survey of algorithms for the generalized assignment problem. *European Journal of Operational Research*, 60(3):260–272. ISSN 03772217.

Chu, P. C. e Beasley, J. E. (1997). A genetic algorithm for the generalised assignment problem. *Computers & Operations Research*, 24(1):17–23.

Czapiński, M. e Barnes, S. (2011). Tabu search with two approaches to parallel flowshop evaluation on cuda platform. *Journal of Parallel and Distributed Computing*, 71(6):802–811.

Danna, E., Rothberg, E., e Pape, C. L. (2005). Exploring relaxation induced neighborhoods to improve mip solutions. *Mathematical Programming*, 102(1):71–90.

Dıaz, J. A. e Fernández, E. (2001). A tabu search heuristic for the generalized assignment problem. *European Journal of Operational Research*, 132(1):22–38.

Feltl, H. e Raidl, G. R. (2004). An improved hybrid genetic algorithm for the generalized assignment problem. In *Proceedings of the 2004 ACM symposium on Applied computing*, p. 990–995. ACM.

Glover, F. (1977). Heuristics for integer programming using surrogate constraints. *Decision Sciences*, 8(1):156–166.

Glover, F. (1989). Tabu search—part i. *ORSA Journal on computing*, 1(3):190–206.

Glover, F. (1990). Tabu search—part ii. *ORSA Journal on computing*, 2(1):4–32.

Glover, F. e Laguna, M. (2013). *Tabu Search*. Springer.

Glover, F. e Rego, C. (2006). Ejection chain and filter-and-fan methods in combinatorial optimization. *4OR*, 4(4):263–296.

Higgins, A. J. (2001). A dynamic tabu search for large-scale generalised assignment problems. *Computers & operations research*, 28(10):1039–1048.

Kirk, D. B. e Wen-mei, W. H. (2012). *Programming massively parallel processors: a hands-on approach*, volume 2. Elsevier.

Laguna, M., Kelly, J. P., González-Velarde, J., e Glover, F. (1995). Tabu search for the multilevel generalized assignment problem. *European Journal of Operational Research*, 82(1):176–189.

Martí, R., Duarte, A., e Laguna, M. (2009). Advanced scatter search for the max-cut problem. *INFORMS Journal on Computing*, 21(1):26–38.

Martí, R., Laguna, M., e Glover, F. (2006). Principles of scatter search. *European Journal of Operational Research*, 169(2):359–372.

Nauss, R. M. (2003). Solving the generalized assignment problem: An optimizing and heuristic approach. *INFORMS Journal on Computing*, 15(3):249–266.

Öncan, T. (2008). A Survey of the Generalized Assignment Problem and Its Applications. *INFOR: Information Systems and Operational Research*, 45(3):123–141. ISSN 0315-5986.

Osman, I. H. (1995). Heuristics for the generalised assignment problem: simulated annealing and tabu search approaches. *Operations-Research-Spektrum*, 17(4):211–225.

Pigatti, A., De Aragao, M. P., e Uchoa, E. (2005). Stabilized branch-and-cut-and-price for the generalized assignment problem. *Electronic Notes in Discrete Mathematics*, 19:389–395.

Posta, M., Ferland, J. A., e Michelon, P. (2012). An exact method with variable fixing for solving the generalized assignment problem. *Computational Optimization and Applications*, 52(3):629–644.

Resende, M. G., Ribeiro, C. C., Glover, F., e Martí, R. (2010). Scatter search and path-relinking: Fundamentals, advances, and applications. In *Handbook of metaheuristics*, p. 87–107. Springer.

Ross, G. T. e Soland, R. M. (1975). A branch and bound algorithm for the generalized assignment problem. *Mathematical programming*, 8(1):91–103.

Sahni, S. e Gonzalez, T. (1976). P-Complete Approximation Problems. *Journal of the ACM*, 23(3): 555–565. ISSN 00045411.

Savelsbergh, M. (1997). A branch-and-price algorithm for the generalized assignment problem. *Operations research*, 45(6):831–841.

Sulewski, D., Edelkamp, S., e Kissmann, P. (2011). Exploiting the Computational Power of the Graphics Card: Optimal State Space Planning on the GPU. *Icaps*, p. 242–249.

Wilson, J. (1997). A genetic algorithm for the generalised assignment problem. *Journal of the Operational Research Society*, 48(8):804–809.

Yagiura, M., Ibaraki, T., e Glover, F. (2006). A path relinking approach with ejection chains for the generalized assignment problem. *European journal of operational research*, 169(2):548–569.