Estratégia bi-critério para um problema de escalabilidade em computação nas nuvens

Marco Túlio Reis Rodrigues
Institut Supérieur d'Informatique, de Modélisation et de leurs Applications
Clermont-Ferrand, França
marcotuliorr@gmail.com

Rui Sá Shibasaki
Institut Supérieur d'Informatique, de Modélisation et de leurs Applications
Clermont-Ferrand, França
ruishiba@gmail.com

Bruno Bachelet
Institut Supérieur d'Informatique, de Modélisation et de leurs Applications
Clermont-Ferrand, França
bruno.bachelet@isima.fr

Christophe Duhamel
Institut Supérieur d'Informatique, de Modélisation et de leurs Applications
Clermont-Ferrand, França
christophe.duhamel@isima.fr

RESUMO

O número de serviços ofertados *online* cresce a cada dia. Em determinados casos alguns serviços podem sofrer um pico de demanda, por exemplo na venda de ingressos para grandes espetáculos ou durante a sexta-feira preta (do Inglês *Black Friday*). Nestes casos, servidores mais potentes são necessários para suportar o pico de demanda afim de manter serviços de qualidade. Uma vez que o preço dessas máquinas são elevados e suas utilizações temporárias, a melhor alternativa encontrada é alugar servidores em uma nuvem informática. Esse artigo trata do problema de dupla atribuição entre serviços e servidores, considerando diferentes tipos de serviços e máquinas, tendo em vista a minimização simultânea do tempo de tratamento dos serviços e dos custos envolvidos na locação dos servidores. Para a resolução do problema propõe-se uma modelagem matemática e uma abordagem heurística, objetivando determinar as soluções para o problema que considera o melhor compromisso entre os dois critérios de otimização.

PALAVRAS CHAVE. nuvens, atribuição, otimização bi-objetivo, escalabilidade.

Área Principal: Apoio à Decisão Multicritério, Otimização Combinatória

ABSTRACT

The number of online services grows continuously. In some cases, services may suffer a peak demand, for instance on tickets sales for a big concert or even during the Black Friday. In such cases, more powerful servers are needed during peak demand periods in order to keep a high-quality service. Since the price of these servers is high and their utilization is only temporary, the best deal could be renting servers in the cloud. This work addresses the services-servers assignment problem, with different types of machines and services, in order to simultaneously minimize time of

services and total costs. To solve the problem, a mathematical model is proposed and a heuristic approach is developed to compute solutions to the problem considering the best trade-off between both optimization criteria.

KEYWORDS. cloud, assignment, bi-objective optimization, scalability.

Main Area: Multicriteria Decision Support, Combinatorial Optimization

1. Introdução

Com o aumento da acessibilidade à internet ocorreu também o crescimento do número de serviços ofertados através desse meio de comunicação. Alguns desses serviços, como os de vendas *online* de ingressos para espetáculos e eventos esportivos, possuem demandas que variam conforme o período. Em alguns casos, devido a grandes quantidades de consumidores já conectados, novos compradores não conseguem acesso ao sistema, gerando insatisfação por parte do cliente e prejuízo para a empresa que comercializa o produto.

Sexta-feira negra (do Inglês *Black Friday*) é um bom exemplo do aumento repentino de demanda. O evento ocorre uma vez por ano e num curto período de tempo (um dia), onde os produtos sofrem uma importante redução nos preços, levando uma grande quantidade de consumidores às compras. A maioria dessas compras são realizadas via internet e as empresas devem estar preparadas para o aumento no acesso de seus serviços *online*.

Para que empresas ofereçam um serviço de qualidade em períodos de grandes demandas é necessário que uma determinada quantidade de servidores esteja disponível para suprir a necessidade dos clientes. No entanto, considerando o preço e o curto período em que essas máquinas seriam utilizadas, a compra dos servidores pode não ser a melhor opção. O preço de compra desses servidores é, em geral, elevado e utilizá-los apenas durante curtos períodos não justificaria o investimento realizado. Com isso, o aluguel dessas máquinas torna-se uma opção interessante para lidar com o problema. A maior dificuldade dessa abordagem encontra-se em determinar a quantidade e os tipos de servidores a serem alugados e em como distribuir a demanda entre eles de forma a minimizar o tempo de tratamento das demandas e os custos de locação e utilização dos servidores.

Atualmente, a solução para a falta de recursos computacionais próprios é encontrada através do *cloud computing* (em português, computação em nuvem). O orgão americano NIST (Instituto Nacional de Padrões e Tecnologia) fornece em [Mell and Grance(2011)] a definição para o conceito de computação em nuvens: *cloud computing é um modelo para permitir um acesso expandível, conveniente e sob demanda a recursos computacionais compartilhados e configuráveis, que podem ser rapidamente providos e lançados com um esforço de gerenciamento mínimo e uma mínima interação com fornecedores do serviço*. Para o problema em questão, o contratante do serviço de *cloud computing* alugaria servidores instalados em outro endereço, sem que ele precise ter as máquinas fisicamente presentes em sua empresa. Esse novo serviço computacional é atrativo para empresários, uma vez que elimina a planificação antecipada da necessidade de recursos e permite que pequenas empresas mantenham um nível mínimo de recursos, aumentando-os se necessário [Zhang et al.(2010)].

Uma vez que decide-se alugar servidores online torna-se necessário definir suas quantidades e capacidades de processamento e desempenho de forma a tratar os serviços e minimizar o custo e tempo de acesso. Assim, o Problema Bi-critério de Atribuição de Serviços em Nuvens (PBASN) investigado neste trabalho consiste em minimizar o custo e tempo de acesso, considerando as demandas de serviço e os servidores alugados. Dessa forma, estamos em um contexto estático e determinístico em relação às demandas.

Em [Visée et al.(1998)], é proposto um método em duas fases para um problema de otimização combinatória multi-objetivo, como o problema da mochila. Na primeira fase, um grupo de soluções é determinado através da resolução do problema com um único critério; em seguida, através de métodos de *Branch-and-Bound*, soluções correspondentes ao segundo critério são estabelecidas nas proximidades daquelas já encontradas. Em [Martí et al.(2015)], diferentes adaptações da metaheurística GRASP (*Greedy Randomized Adaptive Search Procedure*) são propostas para a resolução do problema com multi-critérios, de maneira aproximada. Uma heurística de busca local de vizinhança variável também adaptada a multi-critérios é proposta em [Arroyo et al.(2011)]. Neste caso, o problema envolve ordenação de tarefas e usa o conceito de dominância para selecionar soluções a cada iteração da busca local. Finalmente, em [Deb et al.(2002)] e [Zitzler and Thiele(1999)], trabalha-se com algoritmos genéticos evolutivos para a obtenção de soluções distribuídas na fronteira

de Pareto, otimizando vários critérios simultaneamente.

Neste trabalho nós propomos um modelo bi-critério de programação linear mista. Uma heurística construtiva e uma busca local mono-critéria são apresentadas, também com um busca bi-critéria para o PBASN. O resto deste artigo está organizado da seguinte forma. Na Sessão 2, o PBASN é definido. A heurística construtiva e a busca local são apresentadas na Sessão 3 e a busca bi-critério na Sessão 4. Em sequência, resultados computacionais são fornecidos na Sessão 5. Finalmente, as considerações finais são realizadas na Sessão 6.

2. Definição do problema

O problema PBASN é formalmente descrito a seguir: sejam I e J os conjuntos de tipos de serviços e de tipos de máquinas respetivamente. Para todo tipo de serviço $i \in I$ existe uma demanda D_i a ser tratada. Cada uma das Q_j máquinas do tipo $j \in J$ pode tratar apenas um tipo de serviço e possui capacidade de tratamento $C_{ij}, \forall i \in I$. Além disso, cada máquina possui um custo de locação r_j e um custo de tratamento c_j, por unidade de tempo. A função $t_i^j(z)$ define o tempo de tratamento de z serviços do tipo i numa máquina do tipo j. O problema consiste em determinar quais máquinas serão alugadas e em atribuir serviços a máquinas de forma a tratar todas as demandas e minimizar dois critérios simultaneamente: (1) soma dos custos de reserva e aluguel no tempo e (2) soma dos tempos de tratamento de todas as máquinas. Seja $x_i^j \in \mathbb{N}$ a quantidade de máquinas do tipo j atribuídas aos serviços do tipo i e $y_i^j \in \mathbb{N}$ a quantidade de serviços do tipo i atribuídos às máquinas do tipo j. Seja $z_i^j \geq 0$ a taxa média de serviços do tipo i tratados por uma máquina do tipo j e $w_i \geq 0$ o tempo de tratamento de cada serviço do tipo i. Uma modelagem matemática do PBASN é dada de (1) a (10).

(PBASN)

$$\textit{Min} \quad f^1 = \sum_{j \in J} \sum_{i \in I} (r_j x_i^j + c_j x_i^j t_i^j(z_i^j)) \tag{1}$$

$$\textit{Min} \quad f^2 = \sum_{i \in I} d_i w_i \tag{2}$$

sujeito a:

$$\sum_{i \in I} x_i^j \leq Q_j \qquad \forall j \in J \tag{3}$$

$$\sum_{j \in J} y_i^j = D_i \qquad \forall i \in I \tag{4}$$

$$y_i^j \leq C_{ij} \qquad \forall i \in I, \forall j \in J \tag{5}$$

$$x_i^j z_i^j = y_i^j \qquad \forall i \in I, \forall j \in J \tag{6}$$

$$t_i^j(z_i^j) \leq w_i \qquad \forall i \in I, \forall j \in J \tag{7}$$

$$x_i^j, y_i^j \in \mathbb{N} \qquad \forall i \in I, \forall j \in J \tag{8}$$

$$z_i^j \geq 0 \qquad \forall i \in I, \forall j \in J \tag{9}$$

$$w_i \geq 0 \qquad \forall i \in I \tag{10}$$

As equações (1) e (2) definem as funções objetivos; a primeira define a soma dos custos de reserva e aluguel no tempo e, a segunda, a soma dos tempos para processar as demandas. As restrições (3) garantem que cada máquina executa apenas um tipo de serviço. As restrições (4) asseguram o atendimento das demandas e as restrições (5) garantem que as capacidades das máquinas são respeitadas. As equações (6) determinam o número de cada tipo de serviço para cada tipo de máquina. As restrições (7) calculam o tempo de tratamento. As restrições (8), (9) e (10) asseguram os domínios das variáveis de decisão.

Neste trabalho, a função $t_i^j(z)$ é modelada como uma função linear. Isto permite a linearização do custo (1). O número médio de cada tipo de serviço para cada tipo de máquina (6) também pode ser linearizado.

3. Heurística construtiva e busca local

Devido às limitações do modelo matemático para resolver problemas com maiores dimensões, propõe-se uma abordagem heurística construtiva e uma busca local.

Dado a capacidade de processamento P_j de cada máquina do tipo j e a demanda D_i em serviços do tipo i a ser satisfeita, a heurística construtiva começa por calcular os pesos relativos: $p_j = P_j / \sum P_j$ et $d_i = D_i / \sum D_i$. Em seguida, a demanda D_i para cada tipo i de serviço é atribuído a cada tipo j de máquina de acordo com o seu peso relativo p_j. Assim, a quantidade atribuída é $d_{ij} = D_i p_j$. Da mesma forma, o número de máquinas do tipo j atribuídas para os serviços do tipo i é $q_{ij} = d_i Q_j$. A capacidade das máquinas é limitada. O número de máquinas q_{ij} não é, em geral, suficiente para atender o número de serviços d_{ij}, dependendo da quantidade média de serviços do tipo i assinalados a uma máquina do tipo j (d_{ij}/q_{ij}). Assim, tem-se que calcular o número de máquinas suplementares para cada par (i, j) de maneira a respeitar as capacidades de tratamento. Estas máquina adicionais estão consideradas com um preço mais alto e elas corespondem a uma relaxação da disponibilidade.

A solução inicial gerada respeita todas as restrições do problema (a exceção do número de máquinas) e atende as demandas de serviços. Uma busca local de tipo *Variable Neighborhood Descent* (VND) [Mladenović and Hansen(1997)] é aplicada para melhorá-la, utilizando quatro tipos de movimentos :

1. M1 (eliminar uma máquina): consiste em eliminar uma máquina da matriz de atribuições. Em seguida, percorre-se todos os elementos da matriz e verifica-se a melhor atribuição para eliminação de uma máquina que gera a maior melhoria na função de avaliação. Um movimento é considerado válido se respeita todas as restrições.
2. M2 (transferir máquinas entre tipos de serviços diferentes): consiste em realizar, para cada tipo de máquina, a transferência de máquinas entre atribuições de tipos de serviços diferentes. Para cada tipo de máquina j, verifica-se quais são os dois tipos de serviços $i1$ e $i2$ em que pode ser efetuada a transferência de máquinas entre as atribuições $sol_{i1,j}$ e $sol_{i2,j}$, de forma a melhorar a função de avaliação. A melhor quantidade de máquinas a serem transferidas também é verificada nessa etapa.
3. M3 (transferir serviços entre tipos de máquinas diferentes): movimento que, para cada tipo de serviço, realiza a transferência de serviços entre atribuições de tipos de máquinas diferentes. Assim como no segundo movimento, ele verifica, entre todas as possíveis combinações, a combinação que gera a maior melhoria na função de avaliação.
4. M4 (adicionar uma máquina e transferir serviços a uma atribuição): ao reduzir o número de máquinas a partir do M1, a nova solução pode conter uma quantidade de máquinas inferior a da solução ótima ou essa nova solução pode impedir que novas transferências de serviços sejam realizadas. Afim de perturbar a solução, M4 adiciona uma máquina a uma atribuição e tenta realizar a transferência de serviços a ela. Essa estratégia possibilita a transferência de serviços a atribuições que não poderiam recebê-los mais, devido à restrição de capacidade. Entre todos os elementos da matriz solução, o M4 verifica aquele que receberá uma nova máquina de forma a gerar uma solução com melhor qualidade.

A exploração das vizinhanças M1, M2, M3 e M4 é feita até se encontrar o melhor vizinho explorado em cada movimento (estratégia *best improvement*).

4. Heurística bi-critério

No PBASN, os dois critérios (custo e tempo) são otimizado simultaneamente. O objetivo é encontrar os pontos que representem os melhores compromissos, *i.e.* as soluções não dominadas. Uma solução $s1$ domina $s2$ se:

$$s1 \prec s2 \Leftrightarrow \begin{cases} \textit{custo}(s1) \leq \textit{custo}(s2) \quad \text{e} \quad \textit{tempo}(s1) < \textit{tempo}(s2) \\ \text{ou} \\ \textit{custo}(s1) < \textit{custo}(s2) \quad \text{e} \quad \textit{tempo}(s1) \leq \textit{tempo}(s2) \end{cases}$$

As soluções não dominadas definem a fronteira de Pareto do conjunto de soluções. A Figura 1 apresenta um espaço de soluções e a fronteira de Pareto que se deseja encontrar.

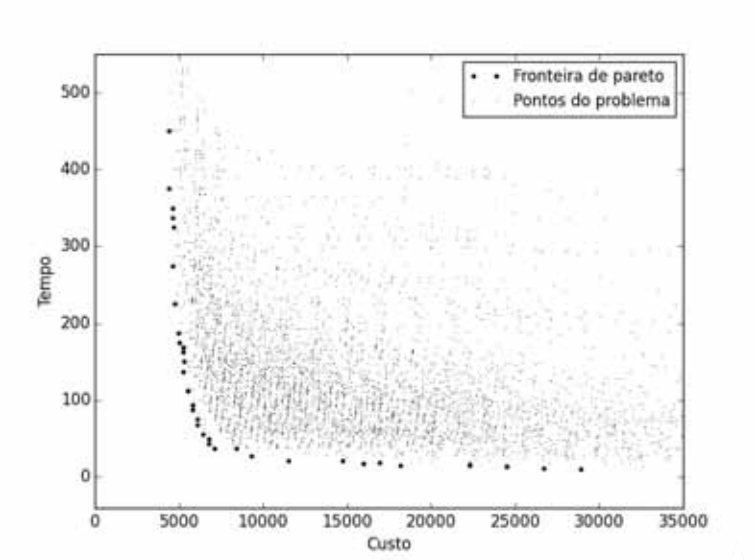


Figura 1: Espaço de soluções de um problema de otimização bi-critério.

O algoritmo começa com uma solução inicial gerada através da heurística construtiva descrita em 3. Em seguida, um dos quatro movimentos é realizado sobre essa solução, inicialmente minimizando em relação ao custo e, posteriormente, em relação ao tempo de tratamento. Se uma melhoria é realizada, as novas soluções geradas são adicionadas a uma lista de soluções a serem otimizadas. A solução atual é, em seguida, adicionada a um lista de soluções já otimizadas. A lista de soluções já otimizadas armazena apenas as soluções que possuem um número de dominância inferior ou igual a um valor máximo de dominância passado por parâmetro (no caso em que esse valor é zero, teremos todos os pontos da fronteira de Pareto). A cada iteração, uma solução é escolhida para ser explorada, realizando os passos anteriores. O processo é repetido até que a lista de soluções a otimizar esteja vazia. O pseudocódigo do algoritmo é apresentado no Algoritmo 1.

Explorar uma solução utilizando o movimento M1 pode piorar sua qualidade com relação ao tempo de execução, mas melhorar a qualidade do custo total. Isto é interessante visto que permite explorar, por exemplo, soluções que utilizam poucas máquinas além de remover máquinas suplementares, quando utilizadas.

A cada vez que uma solução é inserida na lista de soluções otimizadas, a função *checaDominancia* verifica o valor de dominância dessa solução e os novos valores de dominância das antigas soluções; se algum desses valores for maior que o parâmetro de dominância máximo, a solução é descartada. A escolha da próxima solução a ser explorada, realizada pela função *escolheProximaSolucao()*, pode gerar impactos no desempenho geral do algoritmo. Foram avaliadas duas formas de escolha dessas soluções. A primeira utiliza a estratégia *First-In, First-Out* e a outra explora o conceito de fecho convexo de um conjunto de pontos em um plano:

1. Estratégia *First-In, First-Out* (FIFO): Partindo de uma solução inicial, a heurística bicritério explora todas as outras soluções viáveis do problema geradas pelos movimentos implementados pela busca local, minimizando em relação aos dois critérios. Na estratégia FIFO, a cada iteração, a primeira solução da lista de soluções a serem otimizadas é escolhida. Uma análise da evolução das soluções exploradas pelo algoritmo pode ser vista na Figura 2. As soluções encontradas durante a execução das iterações do algoritmo foram divididas em quatro subconjuntos diferentes em que cada subconjunto representa uma etapa do gráfico de evolução das soluções exploradas. O gráfico da Figura 2(a) contém as soluções exploradas

Algoritmo 1 Algoritmo bicritério

f(s): custo/tempo da solução s (funçao de avaliação)
Todo $= \emptyset$: lista de soluções a serem otimizadas
Done$= \emptyset$: lista de soluções já otimizadas
DoneDominance$= \emptyset$: vetor que armazena o número de vezes que uma solução é dominada por outras soluções.

Entrada: solução inicial s
Begin
Todo = Todo $\cup$ $\{s\}$
while Todo $\neq \emptyset$ **do**
  $s \leftarrow$ *escolheProximaSolucao*(Todo)
  $s\prime \leftarrow$ *movimentoOtimizacaoCusto (s)*
  if f($s\prime$) < f(s) **then**
    Todo = Todo $\cup$ $\{s\prime\}$
  end if
  $s\prime \leftarrow$ *movimentoOtimizacaoTempo (s)*
  if f($s\prime$) < f(s) **then**
    Todo $\leftarrow$ Todo $\cup$ $\{s\prime\}$
  end if
  Done $\leftarrow$ Done $\cup$ $\{s\}$
  checaDominancia (DoneDominance, Done, s)
  Todo $\leftarrow$ Todo $\setminus$ $\{s\}$
end while
End

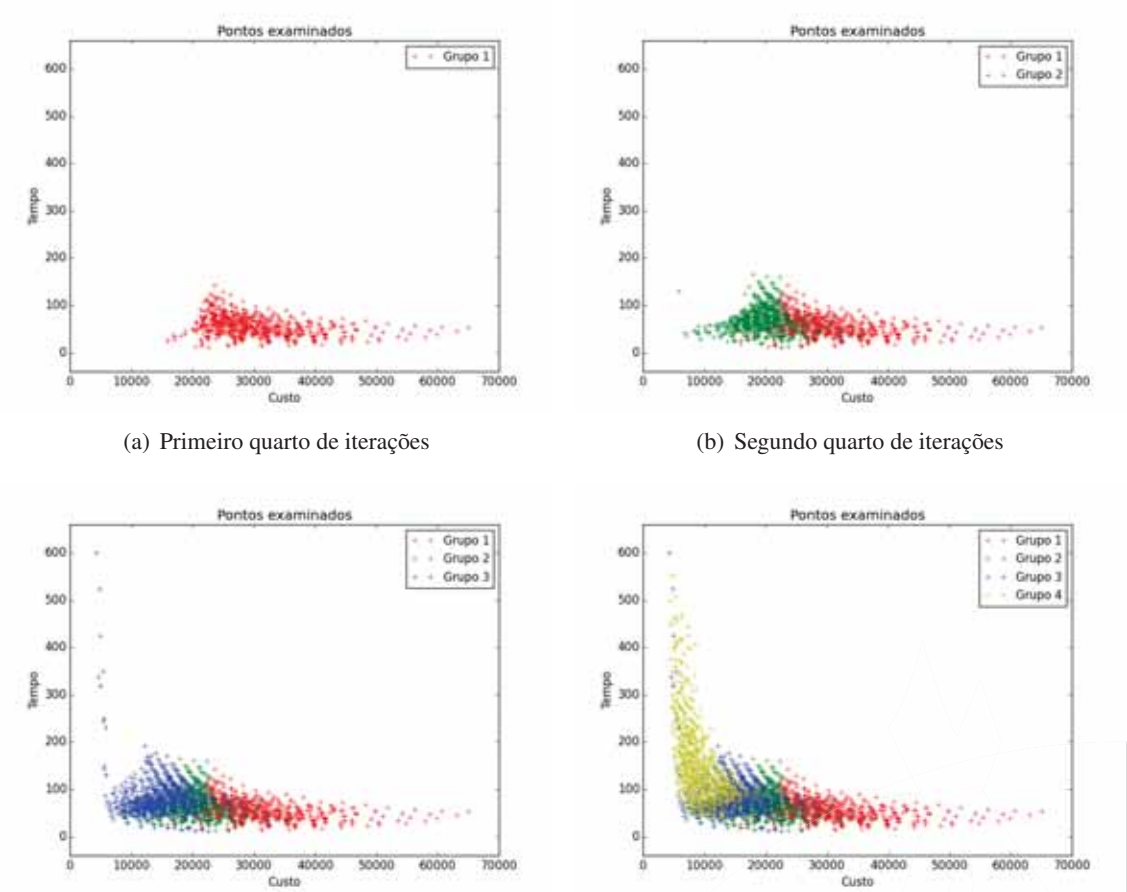
durante a primeira etapa da execução (primeiro quarto das iterações), ao passo que o gráfico da Figura 2(d) representa as soluções exploradas durante as quatro etapas.

2. Estratégia baseada no fecho convexo: O uso de uma abordagem utilizando o conceito de fecho convexo permite explorar soluções mais relevantes em relação à fronteira de Pareto desde o início da heurística. Ao invés de explorar os pontos na ordem *FIFO*, essa abordagem explora os pontos que pertencem ao fecho convexo das soluções a serem exploradas, priorizando as soluções mais próximas das extremidades do espaço de busca. A cada iteração a função *escolheProximaSolucao()* calcula o fecho convexo das soluções a serem otimizadas e escolhe os pontos pertencentes ao fecho convexo como próximos pontos a serem explorados. Foi escolhido o algoritmo de Chan, de complexidade $O(n \log h)$, para calcular o fecho convexo. Da mesma forma que na estratégia FIFO, as iterações do algoritmo foram separadas em quatro etapas de tamanhos iguais. A evolução das soluções exploradas é ilustrada na Figura 3.

Afim de permitir uma maior variedade das soluções exploradas pelo algoritmo, foi criada um método que gera um conjunto de soluções iniciais aleatórias. A ideia é modificar os vetores de porcentagens descritos em 3, inserindo neles valores aleatórios e realizando os mesmos passos definidos para a geração da solução. A Figura 4 ilustra as vantagens de utilizar essa abordagem. Observa-se que os pontos verdes e azuis completam os espaços entre os pontos vermelhos e a fronteira de Pareto; os novos pontos aumentam a qualidade da solução. Além disso, com a utilização de múltiplas soluções iniciais, pode-se terminar a execução do algoritmo mais precocemente, visto que o uso do fecho convexo examina os pontos mais externos antes de explorar aqueles que se encontram no interior.

5. Resultados preliminares

Diferentes versões da heurística foram implementadas em relação a estratégia do escolho (FIFO, fecho convexo) e ao número de pontos iniciais. Para medir a qualidade das versões utiliza-se



(a) Primeiro quarto de iterações

(b) Segundo quarto de iterações

(c) Terceiro quarto de iterações

(d) Último quarto de iterações

Figura 2: Evolução das soluções exploradas usando estratégia FIFO.

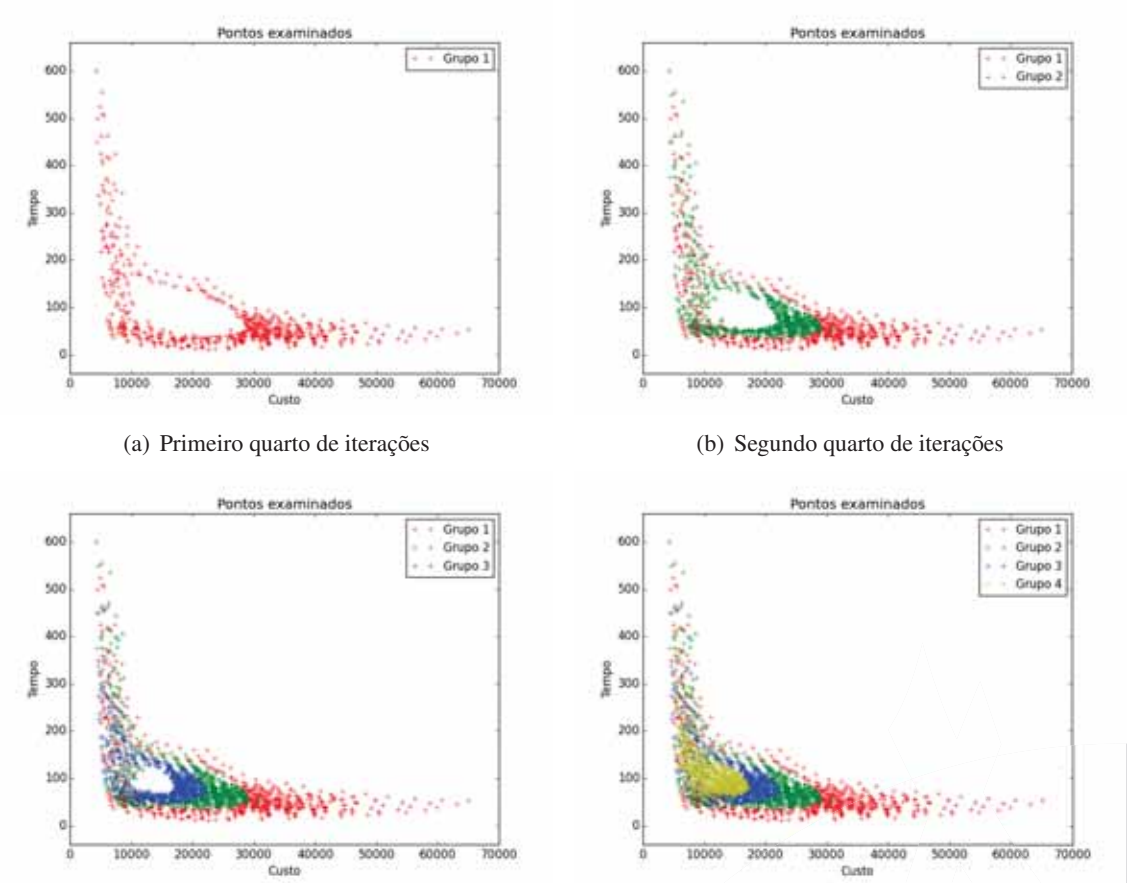
a métrica do hipervolume [Zitzler and Thiele(1999)]. As instâncias foram geradas aleatoriamente.

5.1. Impacto da estratégia do fecho convexo

Para avaliar o comportamento da heurística quatro versões foram testadas : uma versão sem fecho convexo e uma solução inicial (Figura 5(a)) e a mesma versão com múltiplas soluções iniciais aleatórias (Figura 5(b)), assim como uma versão com fecho convexo e solução inicial única (Figura 5(c)) e múltiplas soluções iniciais (Figura 5(d)).

As Figuras 5(a), 5(b), 5(c) e 5(d) mostram as fronteiras de Pareto obtidas a partir de cada versão da heurística. A área correspondente a cada curva está indicada em amarelo em cada um dos gráficos. Verifica-se que o hipervolume das fronteiras de Pareto obtidas pela execução das versões com uma única solução inicial possuem a mesma área, independente do uso do fecho convexo (Figuras 5(a) e 5(c)). Dentre as quatro combinações, observa-se que a menor área e portanto a que representa a melhor solução é obtida pela heurística que utiliza a combinação entre uso do fecho convexo e inicialização aléatória múltipla. Este experimento permite concluir que somente o fecho convexo não influencia o resultado. Porém, ao ser combinado com o uso de múltiplas soluções iniciais, o desempenho da heurística é melhor, *i.e.* uma melhor fronteira de Pareto é obtida.

Além disso, ao utilizar a estratégia FIFO, é preciso explorar uma quantidade elevada de soluções para encontrar uma fronteira de Pareto ao passo que, com a utilização do envelope convexo, encontra-se essa fronteira muito mais rapidamente. Em suma, no caso de ter-se um limite da quantidade de soluções que podem ser exploradas reduzido, a abordagem com envelope convexo gera uma melhor aproximação da fronteira.



(a) Primeiro quarto de iterações

(b) Segundo quarto de iterações

(c) Terceiro quarto de iterações

(d) Último quarto de iterações

Figura 3: Evolução das soluções exploradas usando a estratégia do fecho convexo.

5.2. Desempenho do método em função do número de iterações

As Figuras 6(a) 6(b) indicam a qualidade das soluções geradas pelas versões com e sem fecho convexo ao longo das iterações.

Através das Figuras 6(a) e 6(b) observa-se que, pelo fato de o fecho convexo priorizar soluções mais próximas da fronteira de Pareto, obtém-se uma melhor solução do que a versão sem fecho convexa. Ao explorar as soluções mais relevantes obtém-se uma fronteira de Pareto melhor definida e com menos iterações, o que faz com que o fecho convexo gere melhores soluções para instâncias de dimensões elevadas.

5.3. Influência do número de soluções iniciais

Para verificar a influência do número de soluções iniciais, optou-se por executar a heurística com uma solução inicial única até o seu fim, afim de verificar a quantidade de iterações necessárias para obter o resultado final (Figura 7(a)). Evidentemente, ao utilizar várias soluções aléatorias no início da execução, mais iterações são necessárias para se obter o resultado final, portanto optou-se por estabelecer uma quantidade limite de iterações igual ao número obtido logo que executou-se a versão com uma única solução inicial (Figura 7(b)).

Observa-se através das áreas das Figuras 7(a) e 7(b) que, apesar da implementação com várias soluções iniciais necessitar de mais iterações do que a versão com uma única solução inicial, pode-se obter uma solução de melhor qualidade com um número de iterações que a implementação com apenas uma solução inicial. Observa-se que isso se deve ao fato de que ter várias soluções iniciais contruídas aleatoriamente faz com que a fronteira de Pareto seja melhor preenchida com soluções que antes não eram ser alcançadas com apenas uma solução.

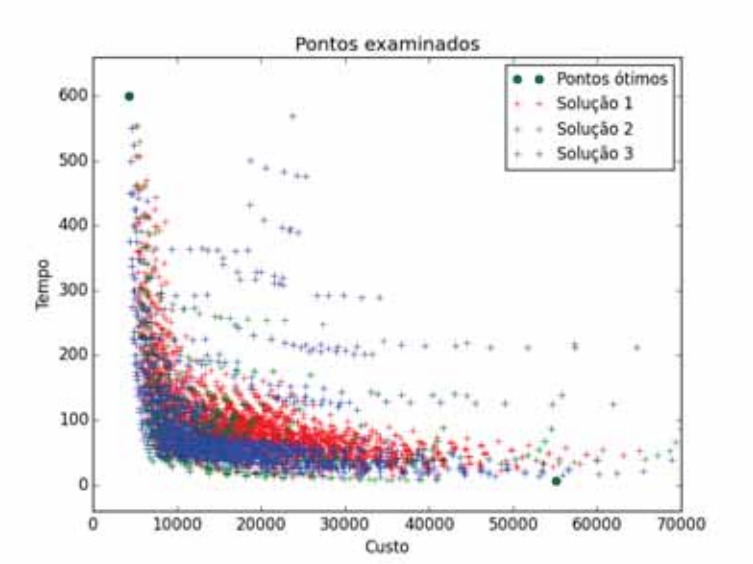


Figura 4: Motivação para a utilização de múltiplas soluções iniciais aleatórias.

	máquinas		serviços		resultados	
	# tipos	qtd.	# tipos	qtd.	area	tempo (s)
h	3	3	5	50	3276796.32	3.94
	3	3	5	100	5987703.79	5.64
	3	3	5	300	9182148.00	11.74
	3	3	5	500	17091014.09	17.45
	3	3	5	1000	39172825.22	32.00
	3	5	5	50	5394638.00	5.05
	3	5	5	100	6295812.00	8.04
	3	5	5	300	9280040.00	18.05
	3	5	5	500	13991723.25	24.44
	3	5	5	1000	36875653.72	42.45

Tabela 1: Influência do número de serviços e de máquinas

5.4. Influência do número de serviços e de máquinas

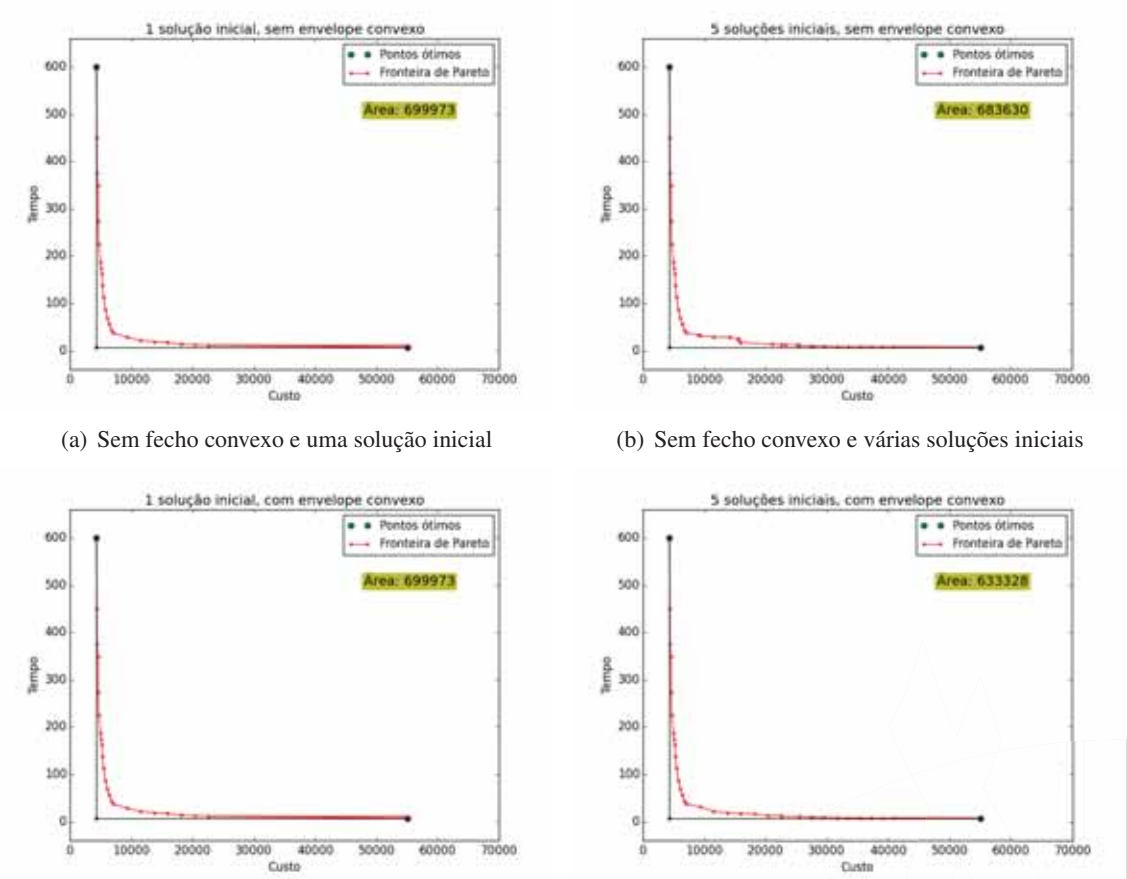
A Tabela 1 mostra a evolução do hipervolume e do tempo da heurística, dependendo do número de máquinas e de serviços. Pode-se notar que os tempos são razoáveis. No entanto, estas instâcias são de tamanho médio e terão de ser feitos estudos posteriores para avaliar o comportamento com instâncias de tamanhos maiores.

6. Conclusão

O estudo apresentado visa estudar a resolução da abordagem bicritério do problema de atribuição entre serviços e máquinas, decidindo a melhor quantidade de servidores alugados. Foram apresentados um modelo matemático e uma heurística para resolver o problema, utilizando a noção de fronteira de Pareto.

Foram implementadas quatro abordagens distintas na tentativa de aprimorar o desempenho da heurística, introduzindo no algoritmo o método baseado no fecho convexo e a utilização de múltiplas soluções aleatórias iniciais: (1) a versão simples sem mecanismos de melhorias, (2) a abordagem com fecho convexo, (3) a versão com múltiplas soluções iniciais apenas e (4) a abordagem com fecho convexo e múltiplas soluções iniciais.

Analizando os resultados obtidos ao utilizar as quatro versões observou-se que a abordagem utilizando o fecho convexo permite encontrar uma fronteira de Pareto em menos iterações que a



(a) Sem fecho convexo e uma solução inicial

(b) Sem fecho convexo e várias soluções iniciais

(c) Com fecho convexo e uma solução inicial

(d) Com fecho convexo e várias soluções iniciais

Figura 5: Influência do fecho convexo

primeira versão. Além disso, ao iniciar a heurística com múltiplas soluções aleatórias é possível preencher a fronteira de Pareto de forma mais eficiente, o que permite determinar uma solução de melhor qualidade.

No futuro, essa heurística será estendida em uma metaheurística bi-objetivo. Mais testes serão realizados com instâncias de maiores dimensões, variando os valores dos parâmetros do problema. As possibilidades de adaptação ao caso *on-line* e de abordagem robusta no caso de incertezas também serão estudadas.

7. Agradecimento

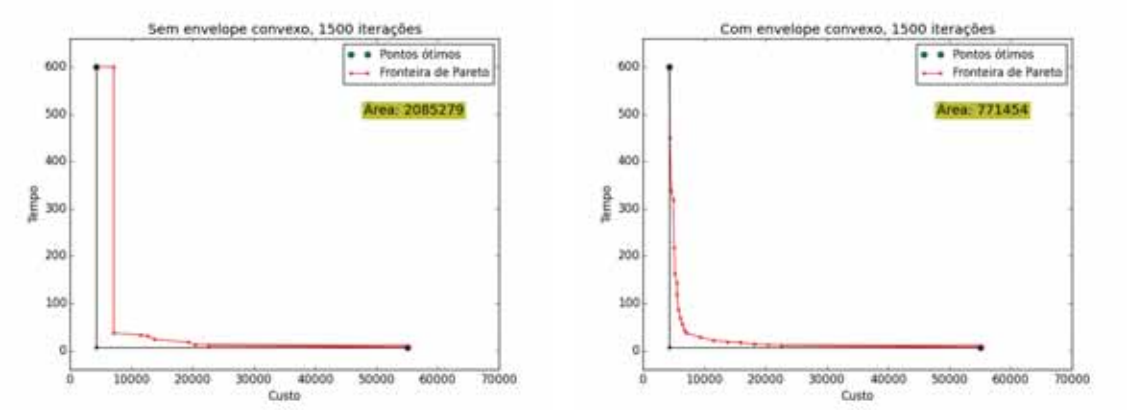
Os autores agradecem o Conselho Nacional de Desenvolvimento Científico e Tecnológico (CAPES, Brasil) pelo apoio recebido através do projeto BRAFITEC.

Referências

Arroyo, J. E. C., dos Santos Ottoni, R., and dos Santos, A. (2011). Multi-objective Variable Neighborhood Search algorithms for a just-in-time single machine scheduling problem. In *2011 11th International Conference on Intelligent Systems Design and Applications (ISDA)*, pages 1116–1121.

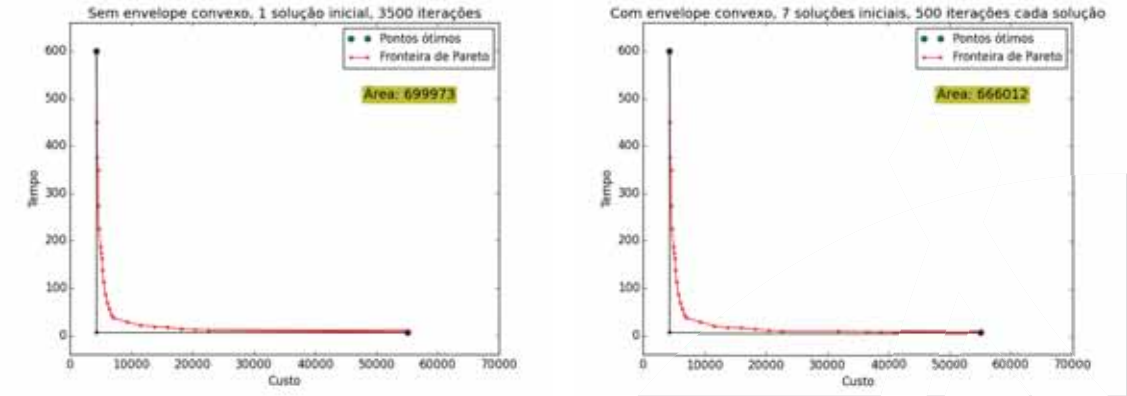
Deb, K., Agrawal, S., Pratap, A., and Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197.

Martí, R., Campos, V., Resende, M. G. C., and Duarte, A. (2015). Multiobjective GRASP with path relinking. *European Journal of Operational Research*, 240(1):54–71.



(a) Solução com 1500 iterações e sem fecho convexo (b) Solução com 1500 iterações e fecho convexo

Figura 6: Influência do fecho convexo em função do número de iterações



(a) Solução com 3500 iterações e solução inicial única (b) Solução com 3500 iterações e 7 soluções iniciais

Figura 7: Influência do número de soluções iniciais

Mell, P. M. and Grance, T. (2011). SP 800-145. the NIST Definition of Cloud Computing. Technical report, National Institute of Standards & Technology, Gaithersburg, MD, United States.

Mladenović, N. and Hansen, P. (1997). Variable neighborhood search. *Computers & Operations Research*, 24(11):1097 – 1100.

Visée, M., Teghem, J., Pirlot, M., and Ulungu, E. L. (1998). Two-phases method and branch and bound procedures to solve the bi-objective knapsack problem. *Journal Of Global Optimization*, 12:139–155.

Zhang, Q., Cheng, L., and Boutaba, R. (2010). Cloud computing: state-of-the-art and research challenges. *Journal of Internet Services and Applications*, 1(1):7–18.

Zitzler, E. and Thiele, L. (1999). Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach. *IEEE Transactions on Evolutionary Computation*, 3(4):257–271.