Estratégias de Obtenção de um Item Máximo em Computação por Humanos

Jeymisson Oliveira, Lesandro Ponciano, Nazareno Andrade, Francisco Brasileiro

[1]Departamento de Sistemas e Computação
Universidade Federal de Campina Grande – UFCG
Campina Grande – PB – Brasil

jeymisson.oliveira@ccc.ufcg.edu.br, lesandrop@lsd.ufcg.edu.br,

{nazareno,fubica}@dsc.ufcg.edu.br

Abstract. *Human computation is a distributed system that orchestrates the work of a group of workers willing to solve relatively simple tasks, whose solutions, once grouped, allow us to solve a computational problem that could not be resolved satisfactorily by using today's machine computation systems. In this work, we address a recurring problem in Human Computation, which is to identify a maximum item in a set of items candidates. We evaluate three algorithms in state-of-the-art for selecting a single candidate for each comparison (2-Max, tournament selection and tournament max) and we propose a multiple elimination strategy that allows workers to eliminate pairs of candidates in a single comparison. In an analytical study, we show that the proposed strategy can increase the efficiency of the algorithms 2-Max and tournament max in terms of the number of tasks required to achieve the maximum item. The analytical study is expanded in an experimental study where we assessed the accuracy of workers when the multiple elimination strategy is used.*

Resumo. *Um sistema de Computação por Humanos orquestra o trabalho de um grupo de trabalhadores dispostos a resolver tarefas relativamente simples, cujas soluções, uma vez agrupadas, resolverão um problema computacional que não poderia ser resolvido de forma satisfatória com os sistemas atuais. Neste trabalho, tratamos de um problema recorrente em Computação por Humanos que é identificar um item máximo em um conjunto de itens candidatos. Avaliamos três algoritmos no estado-da-arte que permitem selecionar um único candidato a cada comparação (2-Max, tournament selection e tournament max) e propomos uma estratégia de eliminação múltipla que permite eliminar pares de candidatos em uma única comparação. Em um estudo analítico mostramos que a estratégia proposta permite aumentar a eficiência dos algoritmos 2-Max e tournament max em termos do número de tarefas necessárias para se obter o item máximo. O estudo analítico é ampliado em um estudo experimental, no qual avaliamos a acurácia dos trabalhadores com uso da estratégia de eliminação múltipla.*

1. Introdução

Existem diversos problemas que os algoritmos e computadores atuais ainda não são capazes de resolver de forma satisfatória [Quinn and Bederson 2011,

Quinn and Bederson 2009, Ahn 2005]. Esses problemas incluem compreensão de linguagem natural, recuperação de informação em imagens e tarefas ligadas à criatividade. Seres humanos, por sua vez, possuem habilidades que permitem resolver esses problemas de forma rápida e precisa [Ahn 2005, Eickhoff and de Vries 2011]. Visando tirar proveito dessas habilidades, tem sido desenvolvido um novo modelo de computação distribuída que utiliza da cognição e da capacidade de raciocínio de seres humanos para resolver problemas que os computadores ainda não podem resolver de forma satisfatória. Esse modelo é denominado Computação por Humanos (*Human Computation*).

Um sistema de Computação por Humanos pode ser visto como um sistema computacional distribuído onde os processadores são seres humanos, chamados de trabalhadores. Tal sistema orquestra o poder cognitivo de um grupo de trabalhadores conectados à Internet dispostos a resolver tarefas relativamente simples, cujas soluções, uma vez agrupadas, resolverão um problema computacional que não poderia ser resolvido de forma satisfatória com os sistemas dotados de processadores convencionais [Ahn 2005]. Esses sistemas também são utilizados para execução de aplicações híbridas, nas quais algumas tarefas são executadas de forma mais eficiente por processadores de silício e outras por humanos [Schall et al. 2008, Dustdar and Truong 2012].

As aplicações de computação por humanos geralmente são divididas em pequenas tarefas descritas como micro tarefas (*microtasks*) ou tarefas que requerem inteligência humana (HIT, do inglês, *Human Intelligence Task*) [Little 2011]. Cada tarefa pode ser executada de forma independente por um trabalhador, e a agregação dos resultados provê a solução do um problema computacional que é objeto da aplicação [Spiekermann 2010, Quinn and Bederson 2011]. Trabalhadores podem apresentar diferentes características em termos de, por exemplo, localização geográfica, habilidades e motivação para executar tarefas. Considerando a motivação dos trabalhadores, os sistemas atuais podem ser amplamente divididos em mercados de trabalho online e pensamento distribuído[1]. Em mercados de trabalho online, os trabalhadores possuem uma motivação financeira, e um dos principais exemplos é a plataforma Amazon Mechanical Turk[2]. Por outro lado, em plataformas de pensamento distribuído, os trabalhadores executam tarefas como um trabalho voluntário. Um dos principais exemplos deste tipo de plataforma é o projeto Galaxy Zoo[3].

Das diversas aplicações para as quais computação por humanos tem sido utilizada, um tipo de aplicação recorrente é identificar um item máximo em um conjunto de candidatos. Isso ocorre por exemplo quando se deseja escolher a fotografia mais criativa em um conjunto de diversas fotografias [Venetis et al. 2012] ou quando se deseja obter a melhor tradução de uma frase em um conjunto de traduções candidatas [Sun et al. 2011]. Nesses casos, se realizadas todas as possíveis comparações entre os itens do conjunto, e se o resultado de cada comparação reflete o senso comum sobre os itens comparados, então o item máximo do conjunto é aquele selecionado como máximo no maior número de comparações. Um fator observado nesses contextos é que um ser humano não consegue (ou apresenta maior dificuldade para) comparar as diversas opções candidatas de uma só vez e identificar a melhor delas [Sweller et al. 1998]. Essa dificuldade agrava-se na

[1]Também chamado pensamento voluntário (*Volunteer Thinking*)

[2]`mturk.com`

[3]`galaxyzoo.org`

proporção em que cresce o número de opções a serem comparadas [Venetis et al. 2012].

Uma abordagem utilizada para resolver esse problema é combinar o conjunto de opções candidatas em pares e instruir trabalhadores para que escolham o item máximo em cada combinação [Sun et al. 2011, Venetis et al. 2012, Ajtai et al. 2009]. A partir da agregação destas comparações entre pares de opções, é possível identificar o item máximo. No caso mais simples, é possível comparar todas as opções em pares, e o item máximo é aquele que vencer mais comparações. Porém, é possível também encontrar o item máximo com outras estratégias que necessitam de um número de combinações menor do que a combinação de todas as opções com todas as outras. A otimização no número de comparações é desejável para otimizar o uso dos trabalhadores disponíveis. Entretanto, como os trabalhadores podem errar ao efetuar uma comparação, qualquer estratégia de otimização precisa também garantir que conseguirá lidar com tais erros. Neste contexto, mostra-se necessário avaliar em que proporção esses erros impactam na acurácia da escolha do item máximo.

Neste trabalho, tratamos do problema da escolha do item máximo no contexto de computação por humanos. Analisamos o desempenho de três algoritmos estado-da-arte de seleção do item máximo: *2-Max* [Ajtai et al. 2009], *tournament selection* [Sun et al. 2011] e *tournament max* [Venetis et al. 2012]. Propomos então uma extensão desses algoritmos a fim de utilizá-los em tarefas em que a eliminação múltipla de itens candidatos pode ser aplicada. Em um estudo analítico que considera diferentes tamanhos de conjuntos de itens candidatos, mostramos que essa extensão permite reduzir o número de tarefas necessárias para obtenção do item máximo. Nosso estudo analítico é ampliado com um estudo de caso, no qual avaliamos a acurácia e a convergência dos trabalhadores quando se utiliza a estratégia de eliminação múltipla proposta neste artigo.

O restante deste artigo está organizado da seguinte forma. Na Seção 2 apresentamos os trabalhos relacionados destacando os algoritmos *2-Max*, *tournament selection* e *tournament max*. Na Seção 3 apresentamos um estudo analítico no qual comparamos os algoritmos considerando diversos cenários de número de opções candidatas. Em seguida, na Seção 4 apresentamos um estudo experimental com uma avaliação da variação da acurácia dos trabalhadores quando se utiliza a estratégia de eliminação múltipla. Finalmente, na Seção 5 apresentamos as conclusões e trabalhos futuros.

2. Trabalhos Relacionados

O problema da obtenção de um item máximo em um conjunto de candidatos por meio de comparações entre pares de itens é recorrente na computação. Em razão disso, diversas abordagens têm sido propostas para solucionar esse problema em diferentes contextos, tais como: sistemas distribuídos compostos por máquinas [Borgstrom and Kosaraju 1993], pesquisa comportamental [Ajtai et al. 2009] e eliminação de resultados de baixa qualidade gerados por seres humanos [Sun et al. 2011, Venetis et al. 2012].

Em sistemas distribuídos compostos por máquinas, Borgstrom e Kosaraju [Borgstrom and Kosaraju 1993] apresentam um estudo analítico do problema que surge quando se deseja ordenar os itens de um conjunto S onde a comparação entre dois itens é imprecisa. O principal foco do estudo é modelar os limites de erros nas respostas para se obter um resultado. O estudo define o mínimo de tarefas necessárias

para se obter o item máximo em um conjunto S, que é dado por $O(\log |S|)$ quando a proporção de tarefas com respostas incorretas (e) não supera 50%, i.e. $e < \frac{1}{2}$. O estudo pressupõe que sempre há uma diferença entre os itens candidatos e essa diferença é detectada pelos processadores ao definir o item máximo. Entretanto, neste trabalho tratamos do problema de obtenção do item máximo em ambiente de computação em que os processadores são seres humanos. Seres humanos apresentam maior dificuldade para comparar opções que apresentam pequena variação entre elas [Sweller et al. 1998]. Focamos na análise de algoritmos para encontrar o item máximo no contexto onde existem comparações com diferentes níveis de dificuldade e em que os seres humanos estão sujeitos a diferentes situações de erro.

Estudos em ciência comportamental também têm sido dedicados às estratégias que permitem obter o item máximo em cenários em que a tarefa de definir qual o item máximo entre dois candidatos é realizada por seres humanos. Ajtai et al. [Ajtai et al. 2009] propõem o algoritmo *2-Max*. A abordagem implementada por esse algoritmo é realizar a busca pelo item máximo partindo de subconjuntos de itens candidatos escolhidos aleatoriamente. Por essa abordagem, obtém-se o item máximo por subconjunto e verifica-se se esse item se mantém quando comparado a todos os demais itens.

Mais próximo do contexto de computação por humanos, têm sido propostas abordagens que utilizam comparação entre pares com o propósito de descobrir o melhor item em um conjunto de itens candidatos gerados por seres humanos. Nesses casos, simplesmente eliminar os resultados menos frequentes não é suficiente, pois nem sempre um item gerado pela maioria dos trabalhadores é um item máximo [Sun et al. 2011, Venetis et al. 2012]. Sun et al. [Sun et al. 2011] apresentam o algoritmo *tournament selection*. Esse algoritmo é baseado na ideia de algoritmos genéticos em que o item mais apto sobressai diante dos menos aptos. No caso, seres humanos definem o item mais apto em comparações de dois itens. O algoritmo realiza r torneios (ou gerações). A cada torneio obtém-se os itens mais aptos selecionados em comparações de n pares escolhidos de forma aleatória, e apenas esses itens participarão das comparações do próximo torneio. O estudo não apresenta um método de definição e/ou otimização dos parâmetros r e n; há apenas a restrição de $n < |S|$.

Venetis et al. [Venetis et al. 2012] também apresentam uma solução baseada em torneios que chamamos neste trabalho de *tournament max*. Nessa solução, o primeiro torneio consiste em combates entre todos os candidatos. Os candidatos que venceram pelo menos um combate compõem um conjunto de itens máximos que duelarão entre si no próximo torneio. O algoritmo implementa esse processo iterativo que termina quando há apenas um item máximo. O algoritmo parametriza a quantidade de itens que serão comparados em cada tarefa, por tratarmos de comparações entre pares, neste artigo assumimos que o valor desse parâmetro é 2.

Neste trabalho analisamos o desempenho dos algoritmos *2-Max*, *tournament selection* e *tournament max* quando se varia o tamanho do conjunto de itens candidatos ($|S|$). Além disso, nós propomos uma extensão desses algoritmos que permite utilizar a estratégia de eliminação múltipla para aumentar a eficiência dos algoritmos *2-Max* e *tournament max* em termos do número de comparações necessárias para se obter o item máximo.

3. Acelerando a obtenção do item máximo

Nesta seção definimos e avaliamos uma estratégia de eliminação múltipla que tem por objetivo acelerar a obtenção do item máximo em um conjunto de itens candidatos. A Tabela 1 apresenta um resumo da notação utilizada ao longo desta seção. Uma das principais métricas de avaliação é o número de comparações necessárias para se obter o item máximo s do conjunto de itens candidatos S ($s \in S$). No contexto de computação por humanos, essa métrica é um indicador da quantidade de trabalho que precisará ser realizado e da quantidade de trabalhadores que serão necessários para se obter o resultado.

Tabela 1. Descrição das variáveis utilizadas

Símbolo	Descrição
S	Conjunto de itens candidatos
s	Item máximo, $s \in S$
r	Número de torneios
(a, b)	Tarefa que compara o item a e o item b, $a \in S$ e $b \in S$.
w	Tamanho máximo de um subconjunto de itens definido como $\lceil\sqrt{\lvert S\rvert}\rceil$

3.1. Definindo eliminação múltipla

Os trabalhos discutidos na seção anterior consideram que dado um conjunto de requisitos que define o que é um item máximo e um par de itens a ser comparado (a, b), um trabalhador é capaz de identificar qual desses itens está em maior conformidade com os requisitos. Entretanto, humanos apresentam dificuldades para identificar pequenas variações entre itens. Tarefas que requerem tomar uma decisão baseada na análise desse tipo de variação tendem a aumentar a fadiga do trabalhador [Eugene Wu 2011], o que tem impacto negativo no tempo necessário para se obter esse resultado e na acurácia do resultado obtido. Isso tende a se agravar quando diversas tarefas com essa característica são escalonadas seguidamente para um mesmo trabalhador.

Consideramos que em aplicações de Computação por Humanos é aceitável obter, como resultado de uma comparação entre dois itens a e b, tanto a quanto b, caso ambos atendam os requisitos para serem um item máximo e sejam indistinguíveis para um ser humano. Entretanto, na situação oposta, se esses itens são indistinguíveis, mas incompatíveis com os requisitos para serem um item máximo, propomos que é adequado eliminar ambos os itens. Isso significa que o resultado de uma comparação pode ser nulo. Com essa eliminação múltipla, previne-se a comparação de um desses itens com outros itens que foram escolhidos em outras comparações. Isso tende a aumentar o número de itens eliminados a cada iteração do algoritmo *2-Max* e reduzir o número de itens que serão comparados entre si em novos torneios no algoritmo *tournament max*. Se ao término do algoritmo nenhum item máximo foi encontrado, todos os itens são ditos insatisfatórios.

3.2. Modelando a eficiência da eliminação múltipla

Nosso propósito é analisar a quantidade de tarefas que precisam ser geradas para se obter o item máximo em um conjunto de itens candidatos S, quando se varia: (i) o número

de itens candidatos ($|S|$), (ii) os algoritmos de escolha do item máximo na versão original e (iii) os algoritmos de escolha do item máximo adaptados para permitir eliminação múltipla.

Nós modelamos o número de tarefas geradas pelos algoritmos *tournament max* e *2-Max* considerando o melhor caso m e o pior caso p de quando a eliminação múltipla é utilizada. O melhor caso é aquele em que, para todo conjunto S, sempre há um item $s \in S$ que atende todos os requisitos para ser um item máximo, enquanto os demais itens desse conjunto não atendem esses requisitos. No que se refere ao pior caso, é importante ressaltar que existe um pior caso extremo em que a opção múltipla não é utilizada pelos trabalhadores. Nesse caso, os algoritmos com eliminação múltipla terão desempenho igual aos algoritmos sem essa opção. Entretanto, neste trabalho consideramos o pior caso como aquele em que a eliminação múltipla é utilizada pelos trabalhadores pelo menos uma única vez a cada iteração do algoritmo. Como mostraremos na Seção 4.2, essa abordagem mostra-se adequada dado que a opção de eliminação múltipla é utilizada por pelo menos um trabalhador a cada conjunto de soluções. Nos próximos parágrafos analisamos esses casos para cada algoritmo.

Tournament Max (TM). Este algoritmo utiliza uma série de torneios para escolher o melhor candidato. O primeiro torneio consiste em confrontar todas as opções candidatas em S e implica na geração de $\binom{|S|}{2}$ tarefas. Os itens escolhidos nessas comparações são mantidos em um novo conjunto de candidatos S; os itens que não são escolhidos em nenhuma comparação deixam de fazer parte do novo conjunto de candidatos S, sendo consequentemente eliminados. Os itens que permanecerem em S são então combinados entre si em comparações no próximo torneio. O algoritmo implementa esse processo iterativo, que sempre remove de S os itens que não vencerem pelo menos uma comparação. O algoritmo termina quando o conjunto de itens candidatos possui apenas o item máximo, i.e., $|S| = 1$.

Na versão original do algoritmo, sempre se remove apenas um item de S a cada iteração, tanto no pior quanto no melhor caso. Em razão disso, o custo do algoritmo varia apenas em função do tamanho do conjunto de itens candidatos ($|S|$). Dessa forma, o custo é calculado reduzindo-se um item candidato a cada iteração, até que não haja mais torneios. Isso ocorre quando não há mais itens a serem combinados, mais formalmente $\binom{|S|-i}{2} = 0$, tal que i é a quantidade de itens candidatos já eliminados do conjunto S. Essa condição é satisfeita quando há apenas um item no conjunto S, i.e., $i = |S| - 1$. O custo desse algoritmo na versão original denominado por t^o é calculado pela Equação 1.

$$t^o = \sum_{i=0}^{|S|-1} \binom{|S| - i}{2} \tag{1}$$

No melhor caso, a versão do algoritmo TM que utiliza eliminação múltipla (TM2) obtém o item máximo com apenas uma combinação de todos os itens do conjunto S. Nessa combinação, o item máximo é escolhido como o melhor em todas as comparações das quais participa (i.e., $|S| - 1$ comparações) e os demais itens são definidos como incompatíveis com os requisitos quando comparados entre si. Neste caso após o primeiro torneio, o novo conjunto de candidatos S terá apenas um candidato, o item máximo. O

custo em termos do número de tarefas (comparações) necessárias é definido como t_m^a e pode ser calculado pela Equação 2. O ganho decorrente do uso da estratégia de eliminação múltipla é calculado como a diferença entre o número de tarefas geradas com a versão original e com a versão adaptada para o uso da opção de eliminação múltipla, i.e., $t^o - t_m^a$.

$$t_m^a = \binom{|S|}{2} \tag{2}$$

Considerando-se o uso da eliminação múltipla no pior caso do algoritmo, remove-se apenas 2 itens a cada iteração, exatamente os itens selecionados pela eliminação múltipla, dado que neste caso a opção de eliminação múltipla é utilizada apenas em uma comparação. Dessa forma, o número de tarefas geradas no pior caso é calculado reduzindo-se dois candidatos a cada torneio até que não haja mais torneios, i.e., $\binom{|S|-2\times i}{2} = 0$, onde i é a quantidade de torneios realizados. Essa condição é satisfeita quando é atingida a quantidade de torneios necessários para que todos os itens não máximos sejam removidos de S com a utilização da eliminação múltipla, que é expresso por $i = \lfloor \frac{|S|}{2} \rfloor$. Portanto o número de tarefas geradas utilizando a opção de eliminação múltipla no pior caso t_p^a é calculado pela Equação 3. Nesse caso, o ganho com o uso da estratégia de eliminação múltipla é calculado como $t^o - t_p^a$.

$$t_p^a = \sum_{i=0}^{\lfloor \frac{|S|}{2} \rfloor} \binom{|S| - 2 \times i}{2} \tag{3}$$

2-Max. O algoritmo 2-Max otimiza a busca pelo item máximo subdividindo o conjunto S em subconjuntos de itens escolhidos de forma aleatória. O tamanho dos subconjuntos é constante e definido como $w = \lceil \sqrt{|S|} \rceil$. Esse algoritmo realiza uma série de torneios e a cada torneio combina-se dois a dois todos os itens do subconjunto previamente escolhido. Cada combinação representa uma tarefa a ser executada por um trabalhador, então, neste caso o custo é de $\binom{w}{2}$ tarefas. Com as respostas para essas tarefas, obtém-se o item máximo $s \in S$ usando voto majoritário. Em seguida, combina-se s com todos os demais itens em S. Cada combinação é uma tarefa de comparação a ser executada por um trabalhador, o que resulta em $|S|-1$ tarefas. Para cada tarefa executada nessa etapa o candidato que não vencer s é eliminado do conjunto original S. Os torneios terminam quando o conjunto de candidatos S se reduz a um subconjunto ($|S| \leq w$). Após isso, combinam-se os itens remanescentes em S, gerando as últimas $\binom{w}{2}$ tarefas. O item máximo é aquele que obtiver o maior número de vitórias nessas últimas tarefas.

No pior caso, o algoritmo 2-Max elimina apenas um candidato por torneio na etapa em que se compara s com os demais candidatos em S. Isso gera $|S| - i$ tarefas, onde i é o número do torneio atual. Como os torneios terminam quando $|S| \leq w$, temos que o algoritmo 2-Max, no pior caso, termina quando $i > |S| - w$. O custo do 2-Max no pior caso é definido como d_p^o e calculado pela Equação 4. Já no melhor caso, após se obter s em S todos os candidatos são eliminados de S logo no primeiro torneio, pois não vencem s na etapa que se compara s com todos os candidatos em S. Dessa forma o custo no melhor caso denotado por d_m^o é de apenas um torneio, calculado pela Equação 5.

$$d_p^o = \sum_{i=1}^{|S|-w} \left(\binom{w}{2} + (|S| - i) \right) + \binom{w}{2} \tag{4}$$

$$d_m^o = \binom{w}{2} + |S| - 1 \tag{5}$$

Para utilizar a opção de múltipla eliminação no algoritmo 2-Max, propomos uma versão modificada (2-Max2). Nessa versão, após a etapa de combinação dos itens do subconjunto w, verifica-se se algum item não foi escolhido como item máximo e foi marcado pela opção de eliminação múltipla. Caso isso ocorra, esse item é removido do conjunto de candidatos S. Com isso, o algoritmo 2-Max2 permite a eliminação de 2 itens candidatos em uma única comparação. Isso permite que menos itens candidatos sejam promovidos acelerando a obtenção do item máximo.

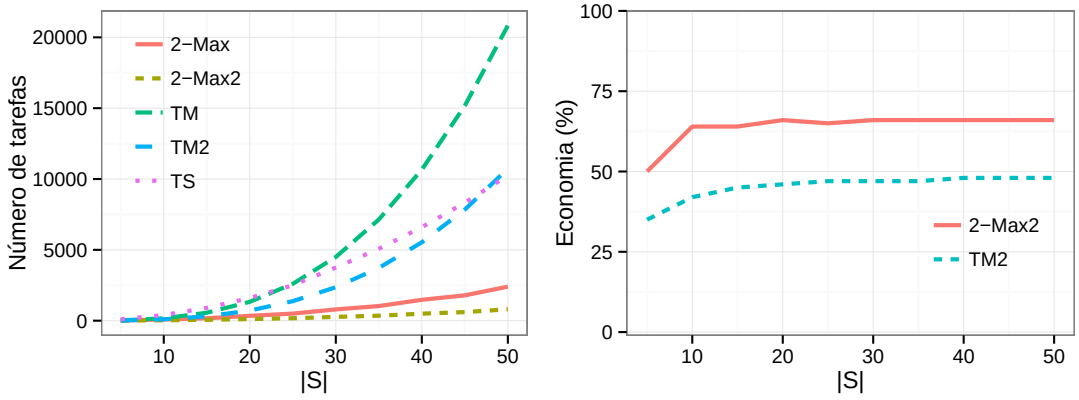
No pior caso, o algoritmo 2-Max2 gera a eliminação de 2 itens candidatos em que a opção de eliminação múltipla é utilizada apenas uma vez. Isso permite que menos itens candidatos sejam promovidos para a próxima etapa do torneio, em que s é comparado com todos os candidatos em $|S| - 2$. Dessa forma com a diminuição de dois candidatos mais a diminuição de 1 candidato por torneio oriunda do 2-Max original, temos a cada torneio $|S| - 3 \times i$ candidatos. Como os torneios terminam com $|S| \leq w$, temos que o 2-Max2 termina quando $i > \frac{|S|-w}{3}$, onde i é o número do torneio. Então, no pior caso, o custo do 2-Max2 pode ser calculado pela Equação 6. No melhor caso, na etapa de combinação dos itens do subconjunto de tamanho w, apenas um item é escolhido como máximo e todos os outros itens são marcados pela eliminação múltipla, então todos os w itens menos o item máximo s são removidos de S, i.e., $w - 1$. Em seguida s é comparado com todos os itens de $|S| - 1 - (w - 1)$. O custo do 2-Max2 no melhor caso é então calculado pela Equação 7. O ganho com o uso da estratégia de eliminação múltipla no melhor caso é calculado como $d_m^o - d_m^a$ e o ganho no pior caso é calculado como $d_p^o - d_p^a$.

$$d_p^a = \sum_{i=1}^{\frac{|S|-w}{3}} \left(\binom{w}{2} + (|S| - 3 \times i) \right) + \binom{w}{2} \tag{6}$$

$$d_m^a = \binom{w}{2} + (|S| - 1 - (w - 1)) \tag{7}$$

Visando analisar o ganho gerado pela opção de eliminação múltipla no pior e no melhor caso, analisamos o número de tarefas geradas e a economia obtida pela estratégia considerando conjuntos de diferentes tamanhos (Figuras 1 e 2). A Figura 1 apresenta os resultados do pior caso e a Figura 2 apresenta os resultados do melhor caso. Essas figuras mostram o número de tarefas geradas e a economia obtida quando se utiliza os algoritmos *2-Max* e TM e as versões adaptadas para permitir eliminação múltipla, 2-Max2 e TM2, respectivamente. Apenas para comparação, essas figuras também apresentam os número de tarefas geradas pelo *tournament selection* (TS). Como não há uma definição dos valores dos parâmetros r e n no algoritmo *tournament selection*, esses valores foram calculados de forma proporcional aos valores utilizados por Sun et al. [Sun et al. 2011] em relação ao tamanho do conjunto de itens candidatos.

No pior caso (Figura 1) os resultados mostram que, para $|S| \leq 25$, o algoritmo menos eficiente é o TS, no qual a estratégia de eliminação múltipla não gera ganho. Para $|S| > 25$ o algoritmo TM firma-se como o algoritmo menos eficiente em termos do número de tarefas e o algoritmo 2-Max2 é o mais eficiente. A Figura 1(b) mostra o ganho gerado pela versão adaptada em comparação à versão original. O algoritmo no qual se obtém maior economia é o algoritmo 2-Max, seguido pelo algoritmo TM. Desta forma o algoritmo 2-Max2 firma-se como o melhor tanto em termos de número de tarefas, quanto no ganho gerado em relação à versão original.



(a) Número de tarefas geradas no pior caso. (b) Percentual de economia em número de tarefas comparado à versão original no pior caso.

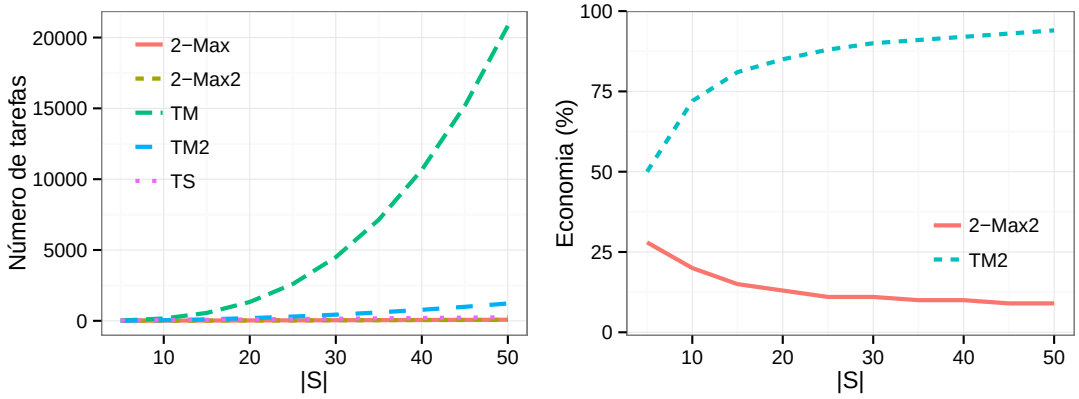
Figura 1. Economia em termos do número de tarefas gerada pela opção de eliminação múltipla no pior caso.

No melhor caso (Figura 2) os resultados mostram que, para $|S| > 10$, o algoritmo menos eficiente é o TM e o mais eficiente é o 2-Max2. A Figura 2(b) mostra o ganho gerado pela versão adaptada em comparação à versão original. O algoritmo adaptado com a opção de eliminação múltipla que se obtém maior economia é o TM, seguido pelo 2-Max.

4. Analisando a acurácia em comparações com eliminação múltipla

A partir dos resultados na seção anterior, identificamos a eliminação múltipla como estratégia promissora para otimizar o número de comparações na busca pelo item máximo em Computação por Humanos. Contudo, a aplicabilidade desta estratégia ainda depende de seu impacto na acurácia dos resultados da computação. Caso, por exemplo, trabalhadores julguem erroneamente que a eliminação múltipla deve ser aplicada com muita frequência, o efeito desta estratégia na aplicação pode ser negativo.

Nesta seção conduzimos um estudo de caso com uma aplicação de Computação por Humanos no qual investigamos o impacto da estratégia de eliminação múltipla na acurácia dos resultados da escolha do item máximo. Primeiro apresentamos o sistema de computação por humanos desenvolvido para conduzir o estudo. Em seguida, discutimos a acurácia dos resultados da escolha do item máximo quando se utiliza a opção de eliminação múltipla.



(a) Número de tarefas geradas no melhor caso. (b) Percentual de economia em número de tarefas comparado à versão original no melhor caso.

Figura 2. Economia em termos do número de tarefas gerada pela opção de eliminação múltipla no melhor caso.

4.1. Sistema de Computação por Humanos

Para avaliar a acurácia do item máximo obtido com comparações entre pares com exclusão múltipla, foi implementada uma aplicação de Computação por Humanos em que trabalhadores voluntários executam a avaliação dos itens. Essa aplicação foi implementada utilizando o PyBossa[4]. PyBossa é um middleware de código aberto para o desenvolvimento de aplicações de Computação por Humanos que lida com aspectos como identidade dos trabalhadores, fila de tarefas a serem executadas e escalonamento das tarefas para os trabalhadores. O PyBossa é escrito na linguagem de programação Python e é inspirado no sistema de middleware de Computação por Humanos *BOINC Open System for Skill Aggregation* (BOSSA) [Korpela 2012] que, por sua vez, é inspirado no sistema de middleware de computação voluntária *The Berkeley Open Infrastructure for Network Computing* (BOINC).

A aplicação desenvolvida consiste em identificar o item máximo em 5 itens candidatos. Os itens candidatos consistem em diferentes formas de reconhecimento das linhas e colunas de uma tabela existente em uma página digitalizada de um livro. Esses itens foram gerados com diferentes configurações de um algoritmo de visão computacional. Em alguns casos os itens são completamente diferentes entre si e em outros casos são mais semelhantes. Em uma comparação (a, b), o critério de escolha do reconhecimento máximo deve considerar as linhas em vermelho de cada reconhecimento que idealmente devem estar posicionadas entre as fronteiras que representam a separação entre linhas e colunas na tabela original. Geralmente, em um reconhecimento não máximo, as linhas vermelhas não estão posicionadas corretamente, ou estão faltando.

Essa aplicação é composta por diversas tarefas que são executadas em paralelo por trabalhadores diferentes. Cada tarefa é definida da seguinte forma: são apresentados os critérios que definem um reconhecimento máximo e dois reconhecimentos candidatos a serem avaliados. O trabalhador deve, então, fornecer como resultado da tarefa o reconhecimento que melhor atende os requisitos para ser um reconhecimento máximo

[4] `www.pybossa.com`

ou nenhum dos reconhecimentos candidatos se ambos os reconhecimentos não atendem os requisitos para ser definido como um reconhecimento máximo (eliminação múltipla). A Figura 3 apresenta um exemplo de um reconhecimento que pode ser considerado máximo (Figura 3(a)) e de um reconhecimento que não pode ser considerado máximo (Figura 3(b)). Esses exemplos são apresentados no tutorial que auxilia o trabalhador a executar as tarefas.

3	Salvador	BA	2.211.539	Centros Regionais		UF
4	Belo Horizonte	MG	2.091.448			
5	Fortaleza	CE	1.965.513	1	São Luis	MA
6	Brasília	DF	1.821.946	2	Maceió	AL
7	Curitiba	PR	1.476.253	3	Natal	RN
8	Recife	PE	1.346.045	4	Teresina	PI
9	Porto Alegre	RS	1.288.879	5	Campo Grande	MS
10	Belém	PA	1.144.312	6	João Pessoa	PB
11	Goiânia	GO	1.004.098	7	São José dos Campos	SP
12	Campinas	SP	908.906	8	Ribeirão Preto	SP
13	São Luís	MA	780.833	9	Cuiabá	MT
14	Maceió	AL	723.230	10	Aracaju	SE
15	Natal	RN	656.037	11	Londrina	PR

(a) Exemplo de reconhecimento que atende aos requisitos para ser um reconhecimento máximo

3	Salvador	BA	2.211.539	Centros Regionais		UF
4	Belo Horizonte	MG	2.091.448			
5	Fortaleza	CE	1.965.513	1	São Luis	MA
6	Brasília	DF	1.821.946	2	Maceió	AL
7	Curitiba	PR	1.476.253	3	Natal	RN
8	Recife	PE	1.346.045	4	Teresina	PI
9	Porto Alegre	RS	1.288.879	5	Campo Grande	MS
10	Belém	PA	1.144.312	6	João Pessoa	PB
11	Goiânia	GO	1.004.098	7	São José dos Campos	SP
12	Campinas	SP	908.906	8	Ribeirão Preto	SP
13	São Luís	MA	780.833	9	Cuiabá	MT
14	Maceió	AL	723.230	10	Aracaju	SE
15	Natal	RN	656.037	11	Londrina	PR

(b) Exemplo de reconhecimento que não atende aos requisitos para ser um reconhecimento máximo

Figura 3. Exemplo de reconhecimento que atende (a) e que não atende (b) os requisitos para ser um reconhecimento máximo.

Os trabalhadores que executaram as tarefas são alunos do curso de Ciência da Computação da Universidade Federal de Campina Grande convocados por e-mail a contribuir. Cada trabalhador, ao acessar o sistema pela primeira vez, é exposto a um tutorial que descreve a aplicação e exemplifica resultados desejáveis de avaliações de itens. No total, 108 trabalhadores executaram 2.397 tarefas. Essas tarefas foram geradas a partir de 52 tabelas diferentes e 5 itens candidatos por tabela, o que resulta em uma combinação de 10 pares por tabela e 520 tarefas diferentes. Em nosso experimento, o ambiente foi configurado de modo que cada comparação é executada por no mínimo 2 trabalhadores diferentes e um mesmo trabalhador não avalia mais de uma vez um mesmo par de itens candidatos. A Figura 4 apresenta a distribuição do número de tarefas executadas pelos trabalhadores.

4.2. Convergência e acurácia

A Figura 5 apresenta o percentual de trabalhadores que utilizaram a opção de eliminação múltipla em cada uma das 520 comparações de dois itens candidatos que foram executadas pelos trabalhadores. Nessa figura, as comparações encontram-se ordenadas de forma

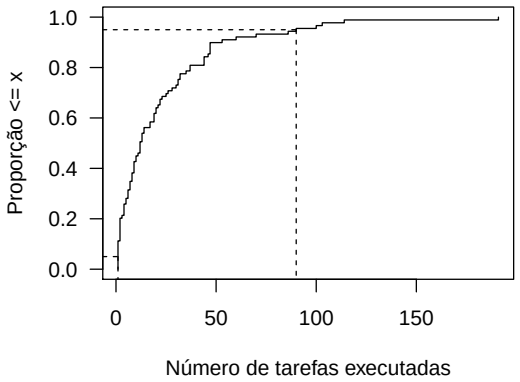


Figura 4. Função de distribuição acumulada do número de tarefas executadas pelos trabalhadores.

decrescente pelo percentual de uso da eliminação múltipla. Em 73 dessas comparações entre dois itens candidatos, 100% dos trabalhadores que executaram essas comparações optaram pela eliminação das duas imagens candidatas. Isso indica que quando a diferença entre as alternativas não é significativa e ambos os candidatos não atendem aos requisitos para serem um item máximo, os trabalhadores convergem na escolha da eliminação múltipla. De outro modo, quando há diferença entre os itens candidatos, os trabalhadores escolhem algum item resultando que 0% dos trabalhadores utilizam a opção de eliminação múltipla nessas comparações. Em nosso experimento, isso ocorre em 167 comparações de duas imagens candidatas. Nos demais casos, pelo menos um trabalhador divergiu dos demais. Nesses casos, o voto majoritário é usado para definir o item máximo entre os dois itens candidatos ou a eliminação de ambos.

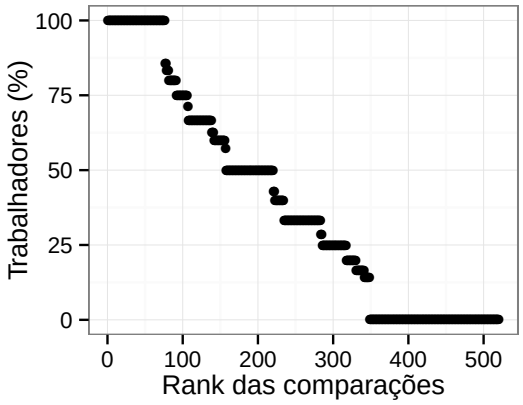


Figura 5. Proporção de uso da opção de eliminação múltipla em cada comparação de 2 itens candidatos.

Para definir a acurácia das respostas geradas pelos trabalhadores utilizamos as respostas corretas definidas por um especialista. O especialista a quem recorremos é o autor do algoritmo de visão computacional usado para gerar as tarefas, que por sua

vez não participou do desenvolvimento deste artigo. Nossa análise é conduzida de duas formas: (i) se em uma dada comparação os trabalhadores escolhem a mesma opção que um especialista escolhe e (ii) se no conjunto de 5 imagens, a imagem escolhida como o item máximo é a mesma escolhida pelo especialista. O resultado obtido é que, em média, os trabalhadores acertam 87.73% das comparações que executam, com um erro de $\pm 6.14\%$, para um nível de confiança de 95%. Ao processar o voto majoritário para se obter o item candidato que não foi eliminado e que foi escolhido como o item máximo o maior número vezes nas comparações com os demais itens candidatos, obteve-se que esse item máximo obtido em cada conjunto de 5 itens candidatos é o mesmo escolhido pelo especialista.

5. Conclusões e trabalhos futuros

Neste trabalho conduzimos um estudo de eficiência em aplicações que visam obter um item máximo em um conjunto de itens candidatos em Sistema de Computação por Humanos. Avaliamos dois algoritmos no estado-da-arte que permitem selecionar um único candidato a cada comparação (*2-Max* e *tournament max*) e propusemos uma estratégia de eliminação múltipla que tem como propósito permitir que esses algoritmos obtenham o item máximo em menos comparações. Nosso método é baseado em um estudo analítico e em um estudo experimental.

Nosso estudo analítico mostra que a versão adaptada dos algoritmos 2-Max e tournament max, que implementa a eliminação múltipla, permite reduzir o número de tarefas geradas por ambos os algoritmos em sua versão original. Entretanto, com o algoritmo 2-Max adaptado obtém-se o maior ganho em termos do número de tarefas gerada e da economia de tarefas em relação à versão original do algoritmo. Nossos resultados experimentais mostram que os trabalhadores convergem para a escolha do item máximo em conjuntos de itens candidatos com diferentes características. Isso indica que a estratégia de otimização proposta não tem efeito negativo na acurácia dos resultados gerados.

Este trabalho pode ser estendido em diversas perspectivas, como: escalonamento de tarefas, tolerância a falhas e qualidade dos resultados. Estratégias de escalonamento podem ser propostas com o objetivo de aumentar a eficiência em termos, por exemplo, do tempo de resposta das tarefas. No que se refere às estratégias de tolerância a falhas, nosso estudo pode ser estendido para identificar se a probabilidade de erro varia em aplicações que se referem a outros tipos de itens, como: fotos, frases e vídeos. Por fim, com relação qualidade dos resultados, pode-se haver um compromisso entre a velocidade em que o item máximo é encontrado e o intervalo de confiança da precisão dos resultados.

Finalmente, os resultados obtidos neste trabalho serão utilizados no contexto do projeto Memória Estatística (www.memoria.org.br). Esse projeto utilizará Computação por Humanos para selecionar e transcrever para arquivos editáveis, publicações históricas relevantes para a história estatística do Brasil presentes na Biblioteca do Ministério da Fazenda.

Agradecimentos. Os autores agradecem ao IPEA pelas imagens utilizadas no experimento, a Guilherme Gadelha pelo algoritmo de reconhecimento das tabelas e pela atuação como especialista na avaliação dos resultados. Lesandro Ponciano é bolsista CAPES/Brasil e Francisco Brasileiro é pesquisador do CNPq/Brasil.

Referências

Ahn, L. (2005). *Human computation*. PhD thesis, Carnegie Mellon University. UMI Order Number: AAI3205378.

Ajtai, M., Feldman, V., Hassidim, A., and Nelson, J. (2009). Sorting and selection with imprecise comparisons. In *Proceedings of the 36th International Colloquium on Automata, Languages and Programming: Part I*, ICALP '09, pages 37–48, Berlin, Heidelberg. Springer-Verlag.

Borgstrom, R. S. and Kosaraju, S. R. (1993). Comparison-based search in the presence of errors. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, STOC '93, pages 130–136, New York, NY, USA. ACM.

Dustdar, S. and Truong, H.-L. (2012). Virtualizing software and humans for elastic processes in multiple cloudsâ a service management perspective. *International Journal of Next-Generation Computing*, 3(2).

Eickhoff, C. and de Vries, A. (2011). How Crowdsourcable is Your Task? In Lease, M., Carvalho, V., and Yilmaz, E., editors, *Proceedings of the Workshop on Crowdsourcing for Search and Data Mining (CSDM) at the Fourth ACM International Conference on Web Search and Data Mining (WSDM)*, pages 11–14, Hong Kong, China.

Eugene Wu, David R. Karger, S. M. R. C. M. (2011). Platform considerations in human computation. In *CHI 2011 Workshop on Crowdsourcing and Human Computation*, New York, NY, USA. ACM.

Korpela, E. J. (2012). Seti@home, boinc, and volunteer distributed computing. *Annual Review of Earth and Planetary Sciences*, 40(1):69–87.

Little, G. (2011). *Programming with Human Computation*. PhD thesis, Massachusetts Institute of Technology.

Quinn, A. J. and Bederson, B. B. (2009). A taxonomy of distributed human computation. Technical report.

Quinn, A. J. and Bederson, B. B. (2011). Human computation: a survey and taxonomy of a growing field. In *Proceedings of the 2011 annual conference on Human factors in computing system*, CHI '11, pages 1403–1412, Vancourver/BC/Canada. ACM.

Schall, D., Truong, H.-L., and Dustdar, S. (2008). Unifying human and software services in web-scale collaborations. *Internet Computing, IEEE*, 12(3):62 –68.

Spiekermann, K. (2010). Judgement aggregation and distributed thinking. *AI Society.*, 25(4):401–412.

Sun, Y.-A., Dance, C. R., Roy, S., and Little, G. (2011). How to assure the quality of human computation tasks when majority voting fails? In *Proceedings of the Workshop on Computational Social Science and the Wisdom of Crowds (NIPS)*.

Sweller, J., Merrienboer, J. J. G. V., and Paas, F. G. W. C. (1998). Cognitive architecture and instructional design. *Educational Psychology Review*, 10:251–296.

Venetis, P., Garcia-Molina, H., Huang, K., and Polyzotis, N. (2012). Max algorithms in crowdsourcing environments. In *Proceedings of the 21st international conference on World Wide Web*, WWW '12, pages 989–998, New York, NY, USA. ACM.