Vol.2017-DBS-164 No.9
2017/1/17

統計量に基づくマイクロタスク設計の最適化の検討

岩本 英一[1,a)]　池田 光雪[2,b)]　森嶋 厚行[3,c)]

概要：本稿では，クラウドソーシングにおける選択式マイクロタスク設計の最適化について議論する．まず，この問題が，タスクにおける選択肢が選ばれる確率分布に依存する事を示す．次に，マイクロタスクの等価性を定義し，等価ないくつかのマイクロタスク設計を提案する．シミュレーションの結果，正解確率の分布により設計手法を切り替えることはタスク数削減という観点において有効であることが示された．

1. はじめに

近年，ネットワークを通じた問題解決の手法としてクラウドソーシングが注目を集めている．クラウドソーシングとは，問題を解決するために必要な作業をタスクと呼ばれる形で不特定多数の人々 (以下，ワーカ) に依頼し，タスクを処理させる手法である．特に，タスクが非常に短時間で終わるものをマイクロタスク型クラウドソーシングと呼ぶ．一般に，同じ結果を得るタスク設計は無数にあるが，良いタスク設計の条件は三つ考えられる．

第一に，問題解決までに要するコストが小さいことである．例えば，ワーカに報酬として金銭を支払いタスクを依頼することは多い．その場合，より少額でより多くの結果を得ることができることが望ましい．第二に，作業を完了するまでに要する時間が短いことである．同様の結果を得られるならば，かかる時間はより短い方が望ましい．第三に，タスク結果の品質が高くなるような設計であることである．ワーカの誤りを誘発するタスク設計では品質が下がってしまうため，ワーカが回答を誤りにくいようなタスク設計を行うことが理想である．

本稿では，マイクロタスク型クラウドソーシングで広く用いられる**選択型タスク設計の最適化**について議論する．この選択型タスクは十分に簡単であり，タスク完了に必要なコストと時間は共にタスク数に比例すると仮定し，タスク数を対象とした議論を行う．

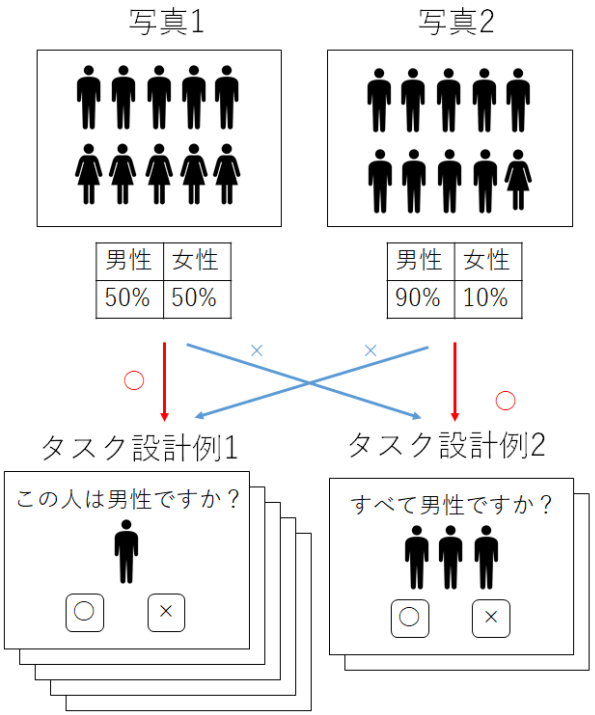

図 1　確率分布によって適切なタスク設計が異なる例

選択式タスクのタスク設計において，問題解決に必要なタスク数は選択肢の正解確率の分布次第であり，自明ではない．例えば，人が何十人と写っている人混みの写真の中からすべての男性を探し出すようなタスクを考える．このタスクの例を図 1 に示す．まず，写っている人物の性別の分布が男女半々になっているある写真を扱うことを考える．この場合はごく単純な二択問題にすればよく，人物の部分を切り出して「この人は男性ですか，女性ですか」と質問していけば良い．次に，工業高校の全校生徒の集合写真のような，一般的な人混みに比べれば男性の比率が非常

[1] 筑波大学 情報学群 情報メディア創成学類
1-2 Kasuga, Tsukuba, Ibaraki 305–8550, Japan
[2] 千葉大学 アカデミック・リンク・センター
1-33 Yayoi, Inage, Chiba 263–8522, Japan
[3] 筑波大学 知的コミュニティ基盤研究センター
1-2 Kasuga, Tsukuba, Ibaraki 305–8550, Japan
[a)] eiichi.iwamoto.2016b@mlab.info
[b)] lumely@chiba-u.jp
[c)] mori@slis.tsukuba.ac.jp

に高い分布となる写真を考える．この場合，写真から切り出した人物を何人か並べ，「すべて男性ですか？」と聞くことで，一度に複数の人物の判定をすることができる．一方で，男女が半々の分布で同じ手法を用いた場合は，男性と女性両方が写っている確率が高く，一度に複数人の判定ができる確率が低いため，効率の向上は期待できない．このように，同じ問題でも，必要なタスク数はタスクの設計と選択肢の正解確率の分布によって決まる．本研究では，タスクの選択肢の正解確率の分布を用いて，タスク設計の最適化を行うことについて検討する．

ただし，選択式タスクにおいて，タスクの選択肢の正解確率の分布は必ずしも手に入るとは限らない．しかし，確率分布が正確である必要はなく，より正確であればそれだけタスク数を削減できる可能性が上がる．したがって，選択肢の正解確率の分布が与えられない場合でも，これまでのタスク結果を利用し，確率分布を推測するといった手法と組み合わせる事で，本研究の活用が期待できる．また，文字認識ソフトウェアの OCR のように，出力として確率分布を与える応用も存在する．

本稿の貢献は次の通りである．(1) 確率分布を用いて，等価なタスク設計を切り替える事を提案する初めての論文である．(2) シミュレーションを用いて，これらを切り替える事が有効なケースを示す．

2. 関連研究

クラウドソーシングにおいて，タスクの設計が品質やコストに影響することは過去の研究により示されている [3], [4]．タスク設計に関する研究として，タスクの設計を簡単にすることで日常空間に組み込んだ研究 [8] や，タスクの質問文を改善する試みを行っている研究 [6], [9]，タスク設計の様々な要素の変化がタスク品質に与える影響を調査している研究 [1], [2] がある．また，コスト削減についても様々なアプローチが行われており，ワーカに与える報酬とタスク完了までにかかる時間のトレードオフに関する研究 [5] や，ワーカの信頼度を独自の確率モデルを用いて推定することで，コストの削減と品質の向上を図っている研究 [7] が存在する．しかし，我々の知る限りでは，選択肢の正解確率の分布からタスク設計の最適化を図るというアプローチでコスト削減を行っている研究は存在しない．

3. タスク設計と等価性

本節では，異なるタスク設計の等価性について議論する．

3.1 タスク設計の定義

まず，タスク設計とタスク結果，およびタスクのドメインについて定義する．

定義 3.1 タスク設計

Algorithm 1 タスク設計例

Input: Q, C
Output: R
1: **for** $i = 0$ to $Q.length$ **do**
2: $\quad R[Q[i]] \Leftarrow generateTask(Q[i], C(Q[i]))$
3: **end for**

タスク設計 T とは，タスクの出力順を規定するアルゴリズムである．タスクの出力順は，途中までに得られたタスクの結果に基づいて決定される．

Algorithm 1 はタスク設計の一例である．全てのタスクでありうる選択肢の集合を $Cand = \{c_1, c_2, \ldots, c_n\}$ とする．このとき，このアルゴリズムは，入力として，質問列 $Q = [q_1, \ldots, q_n]$ と関数 $C : Q \times 2^{Cand}$ を受け取り，タスクを発行し，その結果を出力するものである．ただし，関数 C は，入力として質問 q_i を受け取り，q_i の選択肢の集合を出力する．$generateTask$ 関数は，質問文と選択肢を渡すことでタスクを生成する． □

定義 3.2 タスク結果
あるタスク設計 T のタスク結果 R とは，T が出力するタスクを行う事によって得られた (タスクの質問, タスク結果) のペアの集合である．

例えば，Algorithm 1 のタスク設計では，$R = \{(\text{この文字はなんですか？”あ”}, \text{”あ”}), \ldots\}$ といった形のタスク結果が得られる． □

ここで，$RDom(T)$ を，R が取り得る値の集合とする．すなわち，$RDom(T)$ はタスク結果の集合の集合である．これを，本稿では**タスク設計 T のタスク結果のドメイン**と呼ぶ．

3.2 タスク設計の等価性

次に，タスク設計の等価性を定義する．

定義 3.3 タスク設計の等価性
二つのタスク設計 T_i と T_j が**等価であるとは**，$RDom(T_i) = RDom(T_j)$ であることである． □

3.3 タスク結果のドメインの表記

本稿では，タスク結果のドメインを $RDom(T) = (C, q : a)$ の形式で記述する．それぞれの構成要素は次の通りである．

- C は，タスクの結果に含まれる可能性がある “質問と回答のペアの集合” を規定する．これは，例えば，タスクに現れる質問文 q_i に対して，その選択肢の列 $C(q_i) = [c_{i_1}, \ldots, c_{i_n}]$ を規定するような関数 C で表現

図 2 タスク設計毎のタスク画面例

可能である．例えば，全てのタスクの質問に対して，選択肢が Yes, No のどちらかの場合には，関数 C は全ての q_i に対して $[Yes, No]$ を返す関数となる．

- $q:a$ は，タスク結果に含まれる質問文と回答の関連の多重度を表す．例えば，$1:3$ は，各質問に対して 3 つの回答が得られる必要があることを示している．特別の場合として，数の代わりに*を記述した場合には，数に制限がない任意個を表す．例えば，*:*が指定された場合には，タスク結果では，どのような質問文に対しても，任意個の選択肢を結果に含むことが出来る．

本稿では，最も簡単なケースとして，$RDom(T) = (C, *:1)$ の場合，すなわち，すべてのタスクの質問に対して，回答が 1 つだけ選ばれるようなタスクについて議論する．

4. タスク設計の種類

本節では，質問文の列 $Q = [q_1, \ldots, q_n]$ と選択肢関数 C，および各 $c_{ij} \in C(q_i)$ に含まれる選択肢が正しい確率 $p(c_{ij})$ が与えられた時に，$RDom(T) = (C, *:1)$ となるようなタスク結果を出力するタスク設計を複数説明する．これらのタスク画面の例を図 2 に示す．

- T_BASIC: 各 q_i の $C(q_i)$ 中の全ての選択肢 $[c_{i1}, \ldots, c_{in}]$ に対して，各選択肢が答えであるかどうかを〇か×かの選択で，〇である確率が高い順に問うタスクである．〇が一つ選択された時点でタスクは終了する．
- T_RANK: $C(q_i)$ 中の選択肢を正解確率が高い順に N 個提示するタスクである．提示した選択肢に正解が含まれない場合には，代わりにこれまで表示されていない選択肢を N 個ずつ提示していく．〇が一つ選択された時点でタスクを終了する．
- T_BSEARCH: $C(q_i)$ 中の選択肢を複数提示し，二分探索で正解を絞り込むタスクである．
- T_CONCAT: 一つの q_i ではなく，複数の q_i と，それらの選択肢 $C(q_i)$ から一つずつ取り出し，それを並べて作るタスクである．

Algorithm 2 T_BASIC

Input: Q, C
Output: R

```
1: for i = 0 to Q.length do
2:     S[i] ⇐ sort(C(Q[i]))
3: end for
4: for i = 0 to Q.length do
5:     j ⇐ 0
6:     while true do
7:         if generateTask((Q[i], S[i][j]), [yes, no]) = yes then
8:             break
9:         end if
10:        j ⇐ j + 1
11:    end while
12:    R[Q[i]] ⇐ S[i][j]
13: end for
```

表 1 文字「あ」の文字認識結果の例

候補	あ	め	ぬ	け
確率	35%	25%	20%	20%

4.1 T_BASIC

T_BASIC は単純な〇×問題であり，候補の中で確率の高いものから順に〇か×かの選択でワーカに問い合わせてゆく．例えば，「これは『あ』ですか？」という問題文に対し，ワーカは〇か×かで回答を行う． T_BASIC の処理を，Algorithm 2 に示す．1 行目から 3 行目の for 文では，二次元配列 S に，それぞれの質問について，選択肢を確率が高い順に並び替えて格納する．6 行目から 11 行目の $while$ 文では質問文とひとつの選択肢の組を用いて，タスクを生成している．12 行目にてワーカの答えをハッシュ R に格納する．この処理を 4 行目から 13 行目の for 文ですべての質問に対して行っている．2 行目で利用している関数

$$sort : 2^{Cand} \rightarrow S$$

は，選択肢の集合を与えることで，集合を正解確率の降順の列に変換する．入力として C の出力である選択肢を受け取り，正解である確率の高い順番に並び替えた列を出力する．

4.2 T_RANK

T_RANK は，確率の高いものから順に候補を直接選択肢として与える．例えば，「あ」という文字に対し文字認識の正解確率の分布が表 1 のように与えられていたことを考え

Algorithm 3 T_RANK

```
Input: Q, C
Output: R
 1: for i = 0 to Q.length do
 2:     S[i] ⇐ sort(C(Q[i]))
 3: end for
 4: for i = 0 to Q.length do
 5:     j ⇐ 0
 6:     while true do
 7:         Ans_tmp ⇐ generateTask(Q[i], [S[i][j], S[i][j + 1], other])
 8:         if Ans_tmp ≠ other then
 9:             R[Q[i]] ⇐ Ans_tmp
10:             break
11:         end if
12:         j ⇐ j + 2
13:     end while
14: end for
```

る．この場合，質問文は「この文字は何ですか？」であり，選択肢は「あ」か「め」か「どちらでもない」を表す *other* となる．T_RANK の処理を Algorithm 3 に示す．確率の高いものから順に問い合わせていくという点は T_BASIC と同様であるため似た処理を行うが，7 行目において，タスクの選択肢が選択肢そのものと *other* の 3 択になっている．8 行目から 13 行目で，正解として選ばれた選択肢をハッシュ R に格納している．

4.3 T_BSEARCH

T_BSEARCH は，すべての選択肢を二分探索することで正解を絞り込む．例えば，「あ」という文字の文字認識結果として「あ，め，ぬ，け，な，お，わ，ね」の 8 候補が与えられた場合，「あ，め，ぬ，け」の 4 つを提示し「この中に『あ』はありますか？」と問い半分ずつ確認し，候補から削除していく手法である．T_BSEARCH の処理を Algorithm 4 に示す．3 行目の $from$ 変数と 4 行目の to 変数は現在探索している区間の開始地点と終了地点を示しており，5 行目の $times$ 変数は何度目かの探索であるかを示している．6 行目の *while* 文の終了条件は，探索区間の開始地点と終了地点が一致していることが候補がただひとつに定まっていることと同値であるため $from = to$ とした．7 行目から 9 行目の *for* 文で探索する選択肢を配列 $TmpArray$ に格納し，11 行目から 16 行目でタスクを生成しつつ，その結果に応じて探索区間を変更する．答えがただひとつに定まったとき，18 行目でハッシュ R に答えが格納される．

4.4 T_CONCAT

T_CONCAT では，いくつかの問題を結合してタスクを生成する．例えば，文字「あ」「り」「が」「と」の文字認識結果がそれぞれ「あ」「い」「が」「と」だった場合，質問文は「この文字列は『あいがと』ですか」となる．このように，1 文字でも正しくない文字認識結果がある場合はすべての問題を個別にもう一度確認する手間が増えてしまう．一方，正しく「あ」「り」「が」「と」が与えられた場合，質問文は「この文字列は『ありがと』ですか」となり，1 タスクで 4 文字を処理することができる．このような T_CONCAT の処理を Algorithm 5 に示す．ここで，パラメータ d は一度にいくつの問題を結合させ 1 つのタスクを生成するかを示し，$JoinQuestion$ 関数は，問題の結合に合わせて問題文を改変する．文字認識の例では，$JoinQuestion$ 関数は，「この文字は x ですか」の x に引数として渡した選択肢を並べて格納する．$generateTaskAsRank$ 関数は，引数として問題と選択肢を受け取り，先述の T_RANK を呼び出してワーカに問い合わせることで解決を行う関数である．5 行目から 7 行目では，d 個の質問において最も確率の高い選択肢を配列 $TmpArray$ に格納する．ここで格納された選択肢がすべて正解であった場合，9 行目から 11 行目でハッシュ R に答えとして格納し，不正解であった場合は，13 行目から 15 行目でそれぞれの質問について T_RANK を用いてタスクを生成し再確認を行う．

Algorithm 4 T_BSEARCH

```
Input: Q, C
Output: R
 1: for i = 0 to Q.length do
 2:     S[i] ⇐ sort(C(Q[i]))
 3:     from ⇐ 0
 4:     to ⇐ S[i].length / 2 − 1
 5:     times ⇐ 0
 6:     while from ≠ to do
 7:         for j = from to to do
 8:             TmpArray.push(S[i][j])
 9:         end for
10:         times ⇐ times + 1
11:         if generateTask((Q[i], TmpArray), [yes, no]) = yes then
12:             to ⇐ from + S[i].length / (2 × times) − 1
13:         else
14:             from ⇐ to + 1
15:             to ⇐ from + S[i].length / (2 × times) − 1
16:         end if
17:     end while
18:     R[Q[i]] ⇐ S[i][from]
19: end for
```

4.5 定理

本節で提案した 4 つの設計について，以下の定理が成り立つ．

定理 1 タスク設計のドメイン
質問列 Q と選択肢関数 C が与えられたとする．この時，T = T_BASIC (Q, C), T_RANK (Q, C), T_BSEARCH (Q, C), T_CONCAT (Q, C) の全てにおいて，$RDom(T) = (C, *: 1)$ である．

□

Algorithm 5 T_CONCAT

Input: Q, C, d
Output: R

```
for i = 0 to Q.length do
    S[i] ⇐ sort(C(Q[i]))
end for
for i = 0 to Q.length do
    for j = i to d + i do
        TmpArray.push(S[j][0])
    end for
    if generateTask(JoinQuestion(TmpArray)) = yes then
        for k = i to d + i do
            R[Q[k]] ⇐ S[k][0]
        end for
    else
        for k = i to d + i do
            generateTaskAsRank(Q[k], S[k])
        end for
    end if
    i ⇐ i + d
end for
```

5. シミュレーション

本節では，4 節で提案した各設計について，必要タスク数のシミュレーションを行う．選択肢に対し確率分布を与え，パラメータで偏り方が変化する確率分布を与える式を定義し，パラメータの変化に伴って必要タスク数がどのように変化するかをシミュレーションする．

5.1 設定

シミュレーションで用いる，タスクにおける選択肢の正解確率の分布のデータの生成方法を定義する．ここで，n は選択肢の数とし，それぞれの選択肢が何番目かを示す識別子を $x(1 \leq x \leq n)$ とする．また，t はパラメータであり，t の値が大きくなるほど確率分布における偏りが大きくなる．

$$p(c_x) = \frac{\frac{1}{x^t}}{\sum_{i=1}^{n} \frac{1}{i^t}}$$

この式では，選択肢の識別子である x の値が $1, 2, 3, \cdots$ と増えていくのに伴って，選択肢 c_x の正解確率として用いられる $p(c_x)$ の値が累乗的に減少していく．さらに，パラメータ t の値が大きくなればなるほど値の偏り方が大きくなる．例えば，選択肢が 2 択，つまり $n = 2$ の場合，この関数を用いて生成した確率分布がどのようになるかを図 3 に示す．$t = 0$ のときは 50%と 50%であり，t を 0.3 ずつ増加させる度に偏りが大きくなっている．この式を用いて，$n = 2, n = 5, n = 10$ のそれぞれの場合について，10 タスクを終了するために必要なタスク数の t の値の増加に伴う変化を調べることで，シミュレーションを行った．本シミュレーションは，最も一般的で単純な T_BASIC に比べ他の 3 手法がどの程度必要タスク数を削減できたかを確認することを目的とする．また，簡略化のため，ワーカがタスクにおいて誤った選択肢を選ぶ場合は考慮しないものとした．

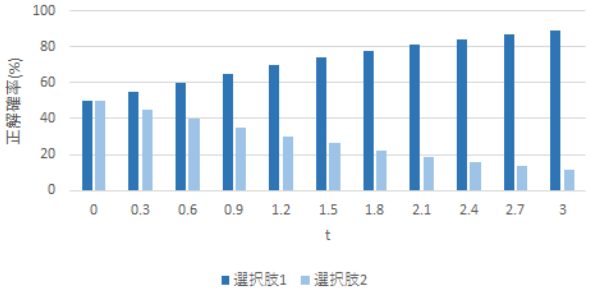

図 3 t の値の変化に伴う確率分布の変化

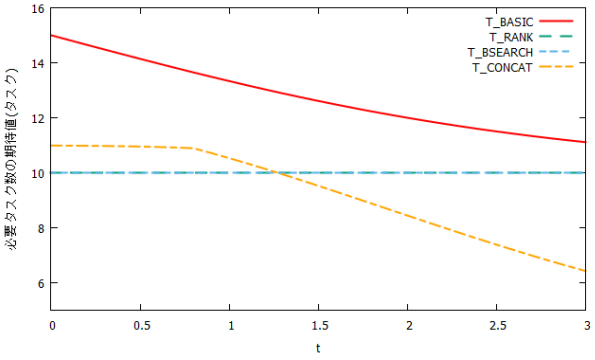

図 4 選択肢が 2 択の場合の必要タスク数の変化

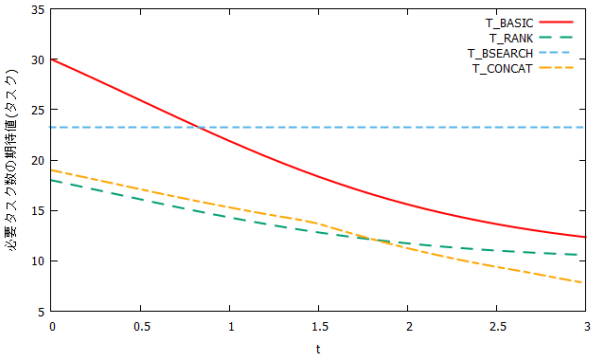

図 5 選択肢が 5 択の場合の必要タスク数の変化

5.2 結果と考察

5.1 節にて定義した式で生成される確率分布を選択肢の正解確率の分布として，選択肢が 2 択の場合，5 択の場合，10 択の場合のそれぞれについて，t の値の変化に伴い 10 個の問題の処理を完了するまでに必要なタスク数の期待値がどのように変化するかを調べた．2 択の場合の結果を図 4 に，5 択の場合の結果を図 5 に，10 択の場合の結果を図 6 に示す．この結果をもとに，選択肢の数が 2 択の場合，5 択または 10 択の場合に分けて考察を行う．

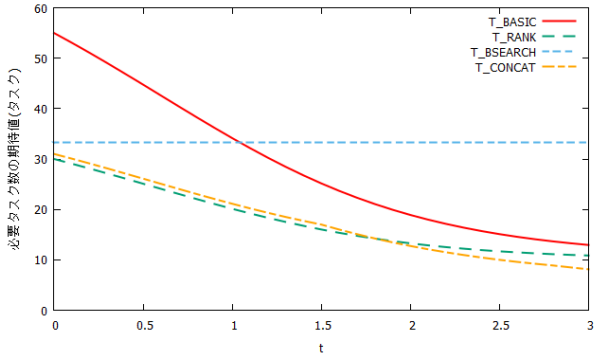

図 6　選択肢が 10 択の場合の必要タスク数の変化

5.2.1　選択肢が 2 択の場合

選択肢が 2 択のタスクならば，T_RANK と T_BSEARCH は必ず 1 つの質問を 1 タスクで処理することができる．今回の設定では質問の数は 10 個であるため，どちらも 10 タスクで終了する結果となった．$t < 1.3$ の領域ではこれら T_RANK と T_BSEARCH の効率が良いが，偏りが大きい $t > 1.3$ の領域においては T_CONCAT が有利になっている．これは，T_CONCAT の，確率分布の中で最大の値が大きいほど複数の問題を 1 タスクで処理することができる確率が高くなるという特徴のためである．

5.2.2　選択肢が 5 択および 10 択の場合

選択肢が 5 択，および 10 択の場合でも 2 択の場合と類似したシミュレーション結果が得られた．偏りが小さいうちは T_RANK が最も効率がよく，5 択と 10 択のどちらの場合でも $t = 1.9$ にて T_CONCAT が T_RANK を上回っている．T_BASIC と T_BSEARCH については，偏りが小さいうちは T_BSEARCH の方が優れているが，5 択の場合は $t = 0.9$，10 択の場合は $t = 1.1$ で T_BASIC が T_BSEARCH を上回る結果となった．T_BSEARCH は確率分布から正解が全く予想できないような状態でも一定のタスク数で問題を処理することができるが，確率分布から正解の目星をつけられるような状況では他のどの設計よりも効率が悪くなってしまうという結果が得られた．

5.2.3　議論

今回の必要タスク数のシミュレーションでは，選択肢の数が 2 つである場合を除き，T_RANK と T_CONCAT の 2 つの効率がよいという結果が得られた．さらに，T_RANK と T_CONCAT のどちらがより有効であるかは正解確率の分布に影響を受けており，タスク設計において，正解確率の分布により設計手法を切り替えることはタスク数削減という観点において有効であることがわかった．

ただし，本稿ではワーカが実際にこれらのタスクを行った際に解答を誤ってしまうことを想定していない．T_CONCAT は内容によっては非常にわかりにくくワーカが誤りやすい複雑な質問になってしまう可能性があるため，ワーカが回答を誤ることを考慮に入れたとき，この順位が大きく入れ替わる可能性がある．そのため，今後の課題としてワーカが回答を誤る確率を考慮し必要タスク数に重みづけを行うということが挙げられる．

6.　まとめと今後の課題

本稿では，タスクの選択肢の正解確率の確率分布を利用することでタスク数を削減できることに着目し，確率分布によったタスク設計を提案することでタスク数を削減できることを示した．

シミュレーションの結果，T_BASIC の必要タスク数を基準として，T_RANK はどのような場合でもタスク数を削減することができた．また，与えられる確率分布により T_CONCAT が有効な場合も多かった．この結果から，選択肢の確率分布によりタスク設計を切り替えることで必要タスク数を削減することが可能であることがわかった．

ただし，これらの結果はあくまでも必要タスク数の期待値についての話であり，タスクの設計を変更することによって品質が低下することはないとした仮定の元に成り立っているものである．そのため，今後の課題としては，実際に評価実験を行い，タスク設計の変化によりタスクの品質が低下していないかどうかを調べ，品質を考慮に入れた更によいタスク設計を考えることが挙げられる．

謝辞　本研究の一部は，JST CREST および JSPS 科研費 (#25240012) の支援による．

参考文献

[1] Alagarai Sampath, H., Rajeshuni, R. and Indurkhya, B.:Cognitively inspired task design to improve user performance on crowdsourcing platforms, *In Proc. CHI2014*, ACM, pp. 3665–3674 (2014).
[2] Brambilla, M., Ceri, S., Mauri, A. and Volonterio, R.:An Explorative Approach for Crowdsourcing Tasks Design, *In Proc. WWW2015*, ACM, pp. 1125–1130 (2015).
[3] Finnerty, A., Kucherbaev, P., Tranquillini, S. and Convertino, G.: Keep it simple: Reward and task design in crowdsourcing, *In Proc. CHItaly2013*, ACM, p. 14 (2013).
[4] Gadiraju, U.: Make Hay While the Crowd Shines: Towards Efficient Crowdsourcing on the Web, *In Proc. WWW2015*,ACM, pp. 493–497 (2015).
[5] Gao, Y. and Parameswaran, A.: Finish them!: Pricingalgorithms for human computation, *In Proc. PVLDB2014*, Vol. 7, No. 14, pp. 1965–1976 (2014).
[6] Hayashi, R., Shimizu, N. and Morishima, A.: Obtaining Rephrased Microtask Questions from Crowds, *International Conference on Social Informatics*, Springer, pp.323–336 (2016).
[7] Hung, N., Thang, D., Matthias, A. and Aberer, K.: Minimizing efforts in validating crowd answer, *In Proc. SIGMOD2015*, ACM, 999-1014 (2015)
[8] 品川有輝，森嶋厚行，中村聡史，寺田努：日常空間に組み込んだ Human Computation 環境によりクラウドソーシングタスク処理，情報処理学会 インタラクション 2014 論文集 pp706–707 (2014)
[9] 丹治寛佳，清水伸幸，森嶋厚行，北川博之：クラウドソーシングにおけるマイクロタスクの質問文の改善手法の提案，JSAI2014 オーガナイズドセッション，pp. 1–4 (2014)