クラウドソーシングによるデータ列挙のための分割統治手法

青木　秀人†　　森嶋　厚行††,†††

† 筑波大学大学院 図書館情報メディア研究科　〒 305–8550 茨城県つくば市春日 1–2
†† 筑波大学 図書館情報メディア系/知的コミュニティ基盤研究センター　〒 305–8550 茨城県つくば市春日 1–2
††† JST さきがけ
E-mail: †{aoki,mori}@slis.tsukuba.ac.jp

あらまし　近年，Web 上の多くのサービスにおいて，クラウドソーシングによるデータ列挙が広く行われている．例えば，レストランのレビューサイトにおけるレストラン名の列挙は，不特定多数の人々によって行われている．このように，クラウドソーシングによるデータ列挙は，機械的にデータを列挙することが困難である場合に有用である．本論文では，マイクロタスク型のクラウドソーシングによるデータ列挙処理について焦点を当て，再現率が高くなるようにタスク分割を行いながらデータ列挙を行う手法を提案する．提案手法では，分割統治のプロセスに群衆が参加するという点に特徴がある．本論文は，提案手法の説明に加え，シミュレーションを用いた提案手法の詳細な評価について報告する．

キーワード　クラウドソーシング

1. はじめに

近年，機械では検索困難な情報を人手で検索する人力検索 (Human-powered Search) が広く使われている．本論文では特に，検索結果のデータ量が多く，かつ再現率を重視するような人力検索（例えば，日本の 1000m 以上の山を列挙すること）を人力データ列挙 (Human-powered Data Enumeration) と呼ぶ．また，クラウドソーシングによって人力データ列挙を行うことをクラウドソーシングによるデータ列挙 (Crowdsourced Data Enumeration) と呼ぶ．

クラウドソーシングによるデータ列挙は多くの Web アプリケーションにとって重要な要素の１つとなっている．例えば，レストランのレビューサイト [1] においてユーザの力を借りてレストラン名を列挙することや，Wikipedia のページなどにおいて，ある分野の項目を網羅的に列挙することが挙げられる．

本論文では，近年注目を集めている，マイクロタスク型のクラウドソーシングプラットフォームを用いたクラウドソーシングによるデータ列挙の処理について焦点を当てる．マイクロタスク型のクラウドソーシングプラットフォームとは，比較的短時間で処理可能なタスク (マイクロタスク) を格納するタスクプールを持つソフトウェアプラットフォームである．リクエスタと呼ばれるタスクの依頼者がマイクロタスクをタスクプールに登録しておき，ワーカはこのタスクプールに格納されているタスクを行う．Amazon Mechanical Turk [2] や，国内では Yahoo!クラウドソーシング [3] などが代表的なものである．

マイクロタスク型のクラウドソーシングプラットフォームでデータ列挙を行う場合，単純には図 1 のようなタスクをワーカに処理してもらうことが考えられる．しかし，このような単純なデータ列挙タスクでは，日本で一番高い山を聞くような，一般的な Human-powered Search のように少数のデータを集める場合には有効であるが，再現率が重要なデータ列挙タスクには適していない．なぜならば，列挙されるべきデータが多くなるとワーカは既に入力されているデータの中から未入力のデータや間違いがあるデータを認識することが困難になり，結果として，データの入力が少なくなり，再現率が低くなってしまうからである．

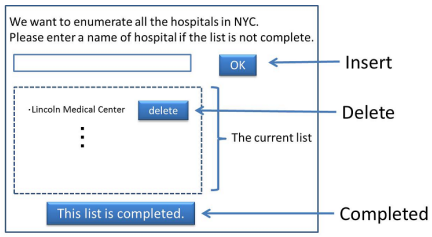

図 1　データ列挙を行なうためのタスクの例

我々は，大規模な問題を解決する重要なアプローチである**分割統治**を適用する事を提案している [4] [5]．具体的には，マイクロタスク型のクラウドソーシングプラットフォームに，列挙の範囲を限定したタスクを多数登録し，全体として広い範囲のデータ列挙を行うというものである．例えば，図 2 のように，ある市の病院の名前のデータ列挙を行いたい場合を考える．この時，市全体の病院のデータ列挙をしてもらうのではなく，区ごとに病院の名前のデータ列挙を行う複数のタスクをタスクプールに登録し，全体として市全体のデータ列挙を行う．

しかし，分割統治手法によるデータ列挙には二つの問題がある．第一に，問題分割の方法は問題ごとに異なるので，あらかじめ分解された多数のタスクをリクエスタが用意することは現実的でないことである．第二に，分割統治した結果が単純なデータ列挙処理と同じ結果になることを保証しなければならないことである．

これらの問題の解として，我々は，論文 [4] [5] において，ワーカがデータ列挙の問題分割に参加する手法の提案を行い，マイ

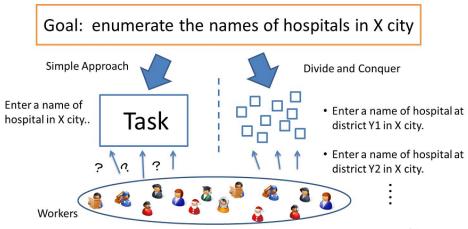

図 2　小さなタスクを使ったデータ列挙

クロタスク型クラウドソーシングプラットフォーム上でのアルゴリズムを示した．また，提案アルゴリズムが，ある条件の下で単純なデータ列挙処理と同じ結果になることの証明を行った．

本論文では，シミュレーションによる提案アルゴリズムの詳細な評価結果を報告する．実験結果より，全体として，ワーカのエラー率がよほど高くならない限り，提案手法が高い再現率を示すことがわかった．

本論文の構成は次の通りである．2. 節では関連研究について説明する．3. 節ではマイクロタスク型のクラウドソーシングプラットフォームとタスクの表現方法について説明する．4. 節ではマイクロタスク型のクラウドソーシングプラットフォームにおけるデータ列挙処理について説明する．5. 節では提案手法について説明する．6. 節ではシミュレーション評価について説明する．7. 節はまとめと今後の課題である．

2.　関連研究

1. 節で説明した通り，Human-powered Data Enumeration は Human-powered Search の一種である．Human-powered Search 以外の人の力を用いた (Human-powered) データ処理としては，選択 (filtering)，結合 (join)，ソート (sort) などのデータベース演算の処理が知られており，これらの演算についての効率化の議論が行われてきた. [6] [7] [8] [9].

特に，Human-powered Search と Human-powered Filtering は関係が深く，どちらも条件を満たすデータを集める処理である．Human-powered Filtering と Human-powered Search の違いは，条件を満たすデータを絞る際に元となる対象のデータの違いである．Human-powered Filtering では計算機に格納されたデータから条件を満たすデータを選択する．一方，Human-powered Search は，データが格納されている場所を限定せずに，データを収集する．

Human-powered Search の処理は多くのシステムで使われている [10] [11] [12]．1. 節で説明した通り，もし，Human-powered Search の結果で再現率を重視せず，いくつかのデータだけが欲しい場合は単純なタスクでも十分であると言える．しかし，再現率が重要である場合や条件を満たすデータが多く欲しい場合，一般的な Human-powered Search ではなくデータ列挙毎に焦点を当てた方法が必要である．

クラウドソーシングによるデータ列挙は既に先行研究において研究がなされている．Trushkowsky ら [13] は生物学や 統計学の分野のテクニックを用いて列挙クエリ (データ列挙) の完了を推定する手法を提案している．本研究との違いは，本研究ではデータ列挙するためのタスク生成に焦点を当てていることや，タスクの完了を推定していないことである．Trushkowsky らの手法と組み合わせてデータ列挙を行うことは今後の興味深い課題の 1 つである．

クラウドソーシングにおけるタスク分割の研究では，Kulkarni ら [14] の研究がある．Kulkarni らは一般的な設定でクラウドソーシングのタスクに分割統治法を適用をし実験・評価を行った．彼らは考察として，ワーカだけでは，適切なガイドなしに問題を分割する事が困難であることを報告している．我々が提案する手法は彼らの考察を基に，ワーカがタスク分割するためのガイドとして，タスク生成プランを与えている．

データ中心型のクラウドソーシングにおける一般的な問題として，データ品質の管理がある．データ品質を向上させる手法はこれまでにいくつか提案されている．例えば，多くのクラウドソーシングでは大数の法則を使った多数決 [6] を採用しデータの品質向上を行っている．別のアプローチとしては，合理的なワーカが適切な値を入力する同調ゲーム [15] [16] がある．論文 [17] では，GWAP（Game With A Purpose）に対して，ゲーム理論による分析，および，インセンティブ構造を変えることで得られるデータが変化することについて議論している．我々の提案手法とこれらのデータ品質向上のための手法は，組み合わせて利用することができる．

3.　マイクロタスク型のクラウドソーシングプラットフォームとタスク表現

本節では，まず，マイクロタスク型のクラウドソーシングプラットフォームのモデルについて説明する．次に本論文で用いるタスクの表現方法について説明する．

3.1　マイクロタスク型のクラウドソーシングプラットフォーム

マイクロタスク型のクラウドソーシングプラットフォームではワーカはタスクプールに存在するマイクロタスク（以下，タスク）を処理する (図 3)．本論文では個々のタスクを t_i, 個々のワーカを w_j, タスクプール P と表す．タスク t_i はタスクプール P に格納される．格納されているタスクは明示的にタスクプール P から削除する処理があるときのみタスクプール P から削除される．したがって，タスクプール P 中のタスク t_i は複数のワーカによって処理されることがある．

マイクロタスク型のクラウドソーシングプラットフォームのアルゴリズムを図 4 に示す．1 行目ではタスクプール P に初期タスク T_0 を格納している．2 行目では，タスクプール P にタスク t_i が格納されている間，次の 3-6 行目の処理を繰り返す：タスクを処理するワーカ w_j を決定（3 行目）し，ワーカ w_j に対してタスク t_i を割り当てる（4 行目）．ワーカ w_j によってタスク処理が行われるのを待ち（5 行目），t_i の処理が行われたら，タスク結果をデータベースに格納するなどの後処理を行う (6 行目).

図 4 のアルゴリズムではタスクを割り当てられたワーカがタスクを行うまで，他のワーカがタスクを行うことがないように単純化している．しかし，ワーカがタスク完了を待つ必要のない，イベント駆動型のアルゴリズムに拡張することも可能である．本論文ではどちらのアルゴリズムにおいても適用可能な議

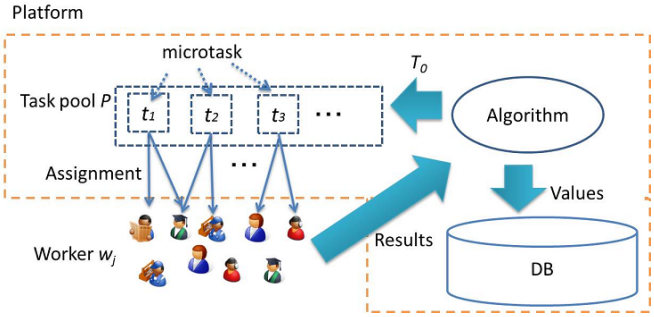

図 3　マイクロタスク型のクラウドソーシングプラットフォームの概要

```
1. P.store(T_0);
2. While !P.isEmpty() {
3.    w_j=getWorker();
4.    t_i=P.assignTask(w_j);
5.    wait until we get r=t_i.performedBy(w_j);
6.    postprocess(t_i, r);
7. }
```

図 4　マイクロタスク型のクラウドソーシングプラットフォームのアルゴリズム

論を行う．

3.2　タスク表現

本手法において，ワーカに割り当て・処理されるタスク t_i は，**タスククラス**のインスタンスとしてモデル化する．タスククラスとはパラメータを与えることでインスタンスを作成するためのテンプレートである．例えば，*EntryTask(item_type, scope)* をデータ列挙を意味するタスククラスとすると，*item_type*, *scope* の 2 つがインスタンスを作成するためのパラメータである．ここで *item_type* は列挙するデータのタイプを意味し，*scope* はデータを列挙する範囲を意味している．この時，ニューヨーク市の病院名を列挙するタスク（インスタンス）は *EntryTask*(`Hospital_name`, `"NYC"`) と表現される．(図 1) *EntryTask* ではワーカに次の 3 つの処理のうち 1 つを行ってもらう．

- データの追加: 入力済みの病院名のリストに存在してない病院名を入力し，“Insert” ボタンを押す．
- データの削除: 入力済みの病院名のリストの中の間違ったデータや重複データを選択し，“Delete” ボタンを押す．
- タスクの完了: 入力済みの病院名のリストが完成していると判断した場合．“Completed” ボタンを押す．

4.　マイクロタスク型のクラウドソーシングプラットフォームにおけるデータ列挙

本節では，本論文で扱うデータ列挙について定義し，単純なデータ列挙について説明する．初めにデータ列挙の定義について説明する．次に単純なデータ列挙の処理の流れを説明し，最後に単純なデータ列挙の問題について説明する．

4.1　データ列挙

データ列挙を次のように定義する．

定義 1. *scope* が表す範囲の *item_type* のデータが全て揃っている集合を *Items(item_type, scope)* するとき，*scope* 範囲における *item_type* のデータ列挙は *Items(item_type, scope)* の全てのデータを列挙することである．

例えば，筑波大学の研究トピックの集合を *Items*(`Research_topic`,`"University of Tsukuba"`) とすると，この集合のデータを列挙することが，筑波大学の範囲における研究トピックのデータ列挙である．　□

```
1. postprocess(t, r){
2.    Let r be the result of
3.              t:TaskEntry(item_type, scope)
4.    switch r.pressed_button {
5.    case insert:
6.        DB.insert(item_type, r.dataitem);
7.        break;
8.    case delete:
9.        DB.delete(item_type, r.dataitem);
10.       break;
11.   case completed:
12.       P.remove(t);
13.   }
14.}
```

図 5　単純なデータ列挙処理の後処理

4.2　単純データ列挙の処理の流れ

本節では単純なデータ列挙 (simple crowdsouced data enumeration, 以下，S-DE) について説明する．この説明では，筑波大学の研究トピック (*Items*(`Research_topic`, `"University of Tsukuba"`)) を例に用いる．

（1） 初期タスク T_0 を {*EntryTask*(`Research_topic`, `"University of Tsukuba"`)} としてタスクプール P に格納する（図 4 の 1 行目）．

（2） タスクプール P に格納された {*EntryTask*(`Research_topic`, `"University of Tsukuba"`)} をワーカに割り当てる（図 4 の 3,4 行目）．

（3） ワーカが処理したタスクの結果の後処理を行う．ワーカが行った処理によって次の 3 つの後処理のうち 1 つを行う（図 4 の 6 行目，図 5）．

- ワーカが新しくデータ入力した場合は結果をデータベースに格納する（図 5 の 5-7 行目）．
- ワーカが既に入力されているデータを削除した場合はデータベースからデータを削除する（図 5 の 8-10 行目）．
- ワーカがタスクが完了していると判断した場合，タスクプール P から対応するタスクを削除する（図 5 の 11-12 行目）．

4.3　単純な手法における問題

単純なデータ列挙では適切にタスクを完了させることは難しく，再現率も低くなる可能性が高い．なぜならば，入力済みのリストのデータが増加し，ワーカは次の 3 つのことが難しくなるからである．(1) 現在の入力済みのリストから不足しているデータを見つけて追加すること．(2) 現在の入力済みのリストから不適切なデータを削除すること．(3) 現在の入力済みのリストのデータが全て揃っているか判断することが．

5.　提案手法

本節では，提案する分割統治によるデータ列挙 (divide-and-conquer crowdsourced data enumaeration, 以下 DC-DE) について説明する．まず，初めに提案手法の 2 つの特徴について説明する．次に提案手法で用いるタスク生成プランについて説明し，最後にタスクを生成するアルゴリズムについて説明する．

5.1 提案手法の概要

提案手法の特徴は分割統治をしてデータ列挙を行うことと，ワーカがタスク生成に参加することである．まず，タスク分割について説明し，次にワーカによるタスク生成を説明する．

(1) タスク分割.

定義 2. s をデータ列挙をする範囲，$item\_type$ を列挙するデータのタイプ，$s_1, \ldots s_n$ を $item\_type$ の別のデータ列挙範囲とする．このとき，$\{s_1, \ldots s_n\}$ が s の $item\_type$ を分割するとは次の式を満たすことである．

$$Items(item\_type, s) = Items(item\_type, s_1) \cup \ldots \cup Items(item\_type, s_n). \quad \square$$

もし，$\{s_1, \ldots s_n\}$ が s の $item\_type$ を分割していた場合，タスク分割を用いて，データ列挙するには，$\{$ *EntryTask*(*item_type*, s_1), …, *EntryTask*(*item_type*, s_n)$\}$ のタスクを用意し，全てのタスクを行えば良い．例えば，筑波大学が $A, B \ldots$ の研究科から構成されていると仮定し，筑波大学の研究トピックのデータ列挙をする場合 $\{$ *EntryTask*(`ResearchTopic`, `"Institute A"`), *EntryTask*(`ResearchTopic`, `"Institute B"`) ,…$\}$ を用意して，全てのタスクを行うことで，筑波大学の研究トピックのデータ列挙ができる．

(2) ワーカによるタスク生成の参加. 一般に，タスク分割を行うことで再現率が高くなることが予想される．しかし，どのように分割したタスクを用意するかは明らかではない．なぜならば，タスクを分割・生成するためにはデータ列挙をする範囲の知識が必要になるが，その知識は問題によって異なり，予め分割されたタスクを用意しておくことは難しいからである．

そこで，提案手法ではワーカがタスクの生成に参加することを提案する．これはタスクを処理することができるワーカは，そのタスクに関する知識を持っているという想定の下，そのワーカが持つ知識を利用して，データ列挙の問題を小さなタスクに分割を行っていくものである．そのため，リクエスタは予め，タスクを分割して用意しておく必要がない．

しかし，どのようにワーカにタスク生成に参加してもらうかはマイクロタスク型クラウドソーシングでは明らかではない．本手法では分割統治によるワーカのタスク生成を実現するために予めシステムに**タスク生成プラン**を与える（図 7. 詳細については 5.2 節で説明する）．タスク生成プランとは，データ列挙する範囲に合わせた階層構造を持ち，ワーカからの入力を基にタスクを生成するための設計図である．

図 6 は，提案手法の概要である．提案手法は，単純なマイクロタスク型クラウドソーシングプラットフォーム (図 3) と次の 2 つの点が異なる．(1) リクエスタがシステムにタスク生成プランを入力すること．(2) ワーカが処理したタスク結果とタスク生成プランを基に，新しいタスクを生成し，タスクプールに追加すること（詳細については 5.3 節で説明する）．

本手法は 1. 節で説明した通り，既に次の 2 つの事を確認している．(1) 理論的評価においてワーカが正しくタスク処理をする場合に提案手法と単純手法のデータ列挙の結果が等しくなること [5]．(2) 被験者実験による評価において，実際にデータ列挙を行い，単純な手法よりも再現率が高くなることやマイクロタスク型のクラウドソーシングに適していること [4].

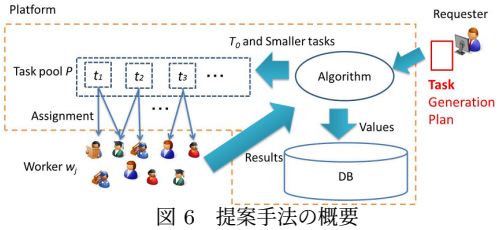

図 6 提案手法の概要

5.2 タスク生成プラン

タスク生成プランとはデータ列挙のための小さなタスクを生成する設計図である．具体的には，データ列挙範囲の階層を記述している．例えば，筑波大学の研究トピックでは，筑波大学の全ての研究科によって階層を構成することができる．図 7 にタスク生成プランのイメージを示す．この図のタスク生成プランはある大学の研究トピックをデータ列挙するための階層が書かれている．ここで末端の “Topic” ノードだけ形が異なるのは，これが最終的に収集したいデータであることを示しているからである．

図 7 の各ノード間のエッジは異なるデータのタイプのデータ列挙タスクに対応する．つまり，ここでは 4 つのエッジがあり，次の 4 つの種類の *EntryTask*(詳細は 5.2.1 節) を生成する．

(Task1) それぞれの大学の研究科のデータ列挙を行うタスク
(Task2) それぞれの研究科の専攻のデータ列挙を行うタスク
(Task3) それぞれの研究科の研究トピックのデータ列挙を行うタスク
(Task4) それぞれの専攻の研究トピックのデータ列挙を行うタスク

次に，各ノードの横に書かれている $!x$ は**完了同意回数**であり，そのデータ列挙が完了したことを x 名のワーカで確認する必要があることを表している．例えば，図 7 においては，研究科の研究トピックが既に全て列挙されたかの確認は 2 名のワーカで同意を取る必要があることが書かれている．

また，エッジの横に書かれている $?y$ は**生成同意回数** であり，タスクを生成する前に *DivisionTask*(詳細は 5.2.1 節) を用いて，そのタスクを行う必要があることを y 名のワーカで同意を得なければならないことを表している．例えば，図 7(Task5) においては，研究科を専攻に分割するかタスク判断するタスクは 2 名のワーカで同意を取る必要があることを表している．

DivisionTask を導入する目的は，データ収集分割のための階層構造が必ずしも均一とは限らない場合があるからである．例えば，物理研究科には専攻があるが，社会学研究科には専攻がない場合，全ての場合に専攻に分割すると，社会学研究科に関して不適切なタスク生成を行なってしまう．

記法. 実際には，タスク生成プランは図 8 のように $[n_1, n_2, \ldots, n_m]$ と記述する．これは，基本的には図 7 と同じ情報を保持しており，各 n_i は各ノードを表す．*Topic* だけは，先頭に*が着いているが，これは末端ノードであることを表す．各ノードの直後の [] の中には，完了同意回数，生成同意回数，末端ノードへのエッジが存在するか (*) を記入する．

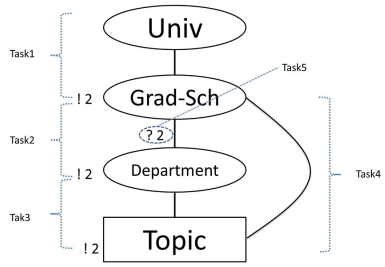

図 7　タスク生成プランのイメージ

```
[Univ{University of Tsukuba}<uname>,
   Grad-Sch[!2, *]<gname>,
   Department[!2, ?2]<dname>,
   *Topic[!2]<topic>]
]
```

図 8　タスク生成プランの表記

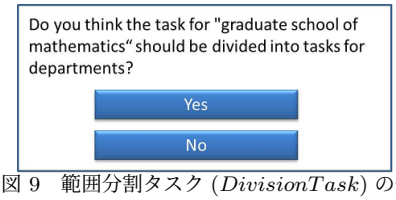

図 9　範囲分割タスク (*DivisionTask*) の例

さらに，実際のタスク生成プランは，図 7 には存在しない次の 2 つの追加情報を持つ．

（1） 最も上位のタスク (Univ-Grad-Sch) の Univ の初期値 (この例では “University of Tsukuba”) が {...} に書かれている．

（2） 各ノードが表すデータを格納するリレーションの属性名が，’< ... >’ に書かれている (例えば，リレーション *Grad-Sch* は属性として *gname* という研究科名を格納する属性を持っている)．

5.2.1　生成するタスク

提案手法では，そのプロセスにおいて 2 種類のタスクを生成する．1 つ目のタスクはデータの入力を求めるタスクである．2 つ目のタスクは新しくタスクを生成を行っていいか確認するタスクである．本節ではそれらを説明する．

(1) データ列挙タスク (EntryTask) このタスク (図 1) では次の 3 つのうち 1 つをワーカが行う．

- (*Insert*) 対象となるデータを入力
- (*Delete*) 既に入力されたデータを削除
- (*Completed*) 対象のデータが全て揃っているか判断

例えば，筑波大学の研究トピックのデータ収集を行うタスク $EntryTask(Topic, \texttt{"University of Tsukuba"})$ の場合 , 具体的にワーカが行うことができる行動は次の通りである．

- (*Insert*) 筑波大学の研究トピックである「クラウドソーシング」などを入力する．
- (*Completed*) 入力されている研究トピックを見て，これらのデータで筑波大学の研究トピックが全て揃っているか判断する．
- (*Delete*) 入力されている研究トピックを見て，間違っている研究トピックや重複している研究トピックを削除する．

提案手法においては，上位のタスク (研究科の収集等) で漏れがあると最終的なデータ収集に大きな影響を与えるため，*Completed* が正しく処理されるかどうかは特に重要である．

(2) 範囲分割タスク (DivisionTask) 範囲分割タスク $DivideTask(scope, n_i)$ は，*scope* の収集範囲をさらに n_i 単位のタスクに分割する前に，そのタスクが必要かどうかに関する同意を求めるものである．図 9 は，数学研究科が専攻に分割できるか問い合わせるタスク $DivideTask(\texttt{"数学研究科"}, Department)$ の画面である．ワーカは「分割できる」か「分割できない」の 2 つから 1 つを選択する．

5.3　タスク生成アルゴリズム

本節では，タスク生成プラン $[n_1\{v_1\}, n_2, \ldots, n_m]$ が与えられた時のタスク生成アルゴリズムについて説明する．本アルゴリズムは，図 4 に沿った流れで行われる．具体的には次の通りである．

（1） 1 行目では，初期タスク $T_0 = \{EntryTask(n_2, v_1)\}$ を格納する．このとき v_1 は n_1 の初期の値であり，データ列挙する範囲を示している．また n_2 は n_1 の子ノードにあたる．例えば，図 8 の場合は，$EntryTask(Grad\text{-}Sch, \texttt{"Tsukuba University"})$ を T_0 に登録する．

（2） 4 行目では，ワーカへのタスクの割り当て (`P.AssignTask(w)`) を行う．このとき，割り当てるタスクはタスクプール P にもっとも早く追加されたタスクとする．つまり，この割り当て方法はデータ列挙を階層化したときに幅優先で行って実行していることになる．

（3） 6 行目では，ワーカが処理したタスクの後処理 (`postprocess(t, r)`) を行う．ワーカが行った処理により次の 4 つの異なった後処理を行う (図 10)．

[Case 1: EntryTask においてワーカが “Insert” を行った場合] 図 10 の 6 行目では，追加されたデータをデータベースに格納する．7-11 行目では，追加されたデータが末端ノードのデータ n_m でなければ，子ノードと追加されたデータから新しくタスクを生成する．このとき，新しく生成されるタスクは次の 2 つの場合には分かれる．

(1) 対応するエッジに生成同意回数 ($?y$) が記述されていない場合は *EntryTask* をタスクプール P に追加する (7-8 行目)．
(2) 対応するエッジに生成同意回数 ($?y$) が記述されている場合，DivisionTask をタスクプール P に追加する（9-10 行目）．

[Case 2: EntryTask においてワーカが “Delete” を行った場合] まず，ワーカによって「削除する」と選択されたデータをデータベースから削除する (13 行目)．次に，削除されたデータを基に生成されたタスクをタスクプール P から削除する (14 行目)．

[Case 3: EntryTask においてワーカが “Completed” を行った場合] ワーカが「完了している」と判断したタスクが完了同意回数（$!x$）を満たしている場合，タスクプール P からタスクを削除する (16-17 行目)．

[Case 4: DivisionTask を行った場合] ワーカが $DivisionTask(r.data\_item, n_i)$ で「分割できる」と生成同意回数 ($?y$) を満たした場合，$EntryTask(n_i, r.dataitem)$ を生成し，タスクプール P に追加する（21-22 行目）．

```
1. postprocess(t, r){
2. Let [n_1{scope}, ..., item_type] be the task generation plan.
3.  If t is EntryTask(n_i, scope) {
4.    switch(r.pressed_button) {
5.        case Insert:   // Case 1
6.          DB.insert(n_i, r.dataitem);
7.          if ((n_{i+1}.?y==0) && (n_i != n_m)){
8.             P.insert(EntryTask(n_{i+1},r.dataitem));
9.          }else if (n_i != n_m){
10.            P.insert(DivisionTask(n_{i+1}, r.dataitem));
11.         }
12.       case Delete:   // Case 2
13.         DB.delete(n_i, r.dataitem);
14.         P.delete(EntryTask(n_{i+1}, r.dataitem));
15.       case completed:   // Case 3
16.         if(completion is agreed) {
17.               P.delete(t);
18.         }
19.    }
20. } else if t is DivisionTask(n_i, r.dataitem) {   // Case 4
21.         if (scope division is agreed) {
22.            P.insert(EntryTask(n_i, r.dataitem));
23.         }
24. }
25. }
```

図 10　提案手法の後処理

6. シミュレーション評価

論文 [4] では，提案手法をクラウドソーシングプラットフォームで実装して実際に適用した結果を報告したが，本論文では，パラメータを様々に変更したした場合の影響を調べるため，シミュレーションによる評価の結果を説明する．本シミュレーションでは，ワーカがタスクを実際に行う事はせず，確率的に振る舞うワーカのモデルを用いて結果を計算する．

本シミュレーションでは，次の評価を行う．(1) ワーカがタスクを間違えて処理する確率をいろいろと変化させた場合，提案手法は単純な手法と比べて再現率は高くなるのか．また，適合率はどうなるか．(2) 分割が多いタスク生成プランと分割が少ないタスク生成プランのどちらが高い再現率になるのか．また，適合率はどうなるか．

6.1 シミュレーション方法

6.1.1 シミュレーションの概要

本シミュレーションではワーカが本来行うタスクについて，ワーカの行動が確率的に決定するモデルを用いてシミュレーションを行う．最初に，シミュレーションにおいてはワーカが間違った処理をする確率 $error\_rate$ を与えておく．シミュレーションの手順は次の通りである．

(1) $error\_rate$ に基づき次のいずれかに決める．(A) ワーカが正しい処理を行う．この場合は次の (2-A) を行う．(B) 間違った処理を行う．この場合は次の (2-B) を行う．

(2-A) (1) で (A) に決定した場合，表 6.1.1 の上部の中からタスクの種類に応じて具体的な処理を行う．このとき具体的な処理内容は処理可能な処理の中から均等な確率で決定する．例えば，$EntryTask$ において入力済みのリストに不足データや間違ったデータがある場合は，タスク完了判断はできない．そのため，不足しているデータの追加か間違ったデータの削除がそれぞれ 0.5 の確率で行う．

表 1　シミュレーションにおけるワーカモデルの行動の種類

処理の正誤	タスクの種類	処理内容
（A）正しい処理	$EntryTask$	不足しているデータを追加
		入力されているリストから間違ったデータを削除
		タスクの完了判断
	$DivisionTask$	正しい分割判断
（B）間違った処理	$EntryTask$	不適切なデータの追加
		入力されているリストから正しいデータを削除
		タスクの完了判断
	$DivisionTask$	間違った分割判断

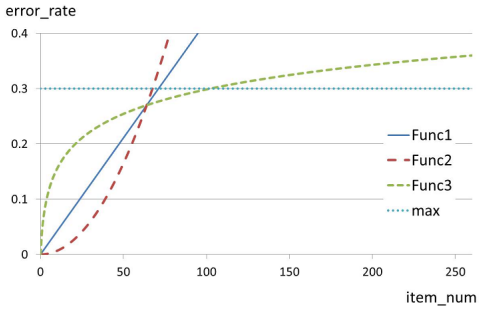

図 11　error_rate が変動する場合の関数

(2-B) (1) で (B) に決定した場合，表 6.1.1 の下部の中からタスクの種類に応じて具体的な処理を行う．このとき具体的な処理内容は（2-A）と同様に処理可能な処理の中から均等な確率で決定する．

6.1.2 $error\_rate$ の与え方

$error\_rate$ は，次の 2 種類の方法で与えた．

(1. 固定) シミュレーション中は常に一定させる．

(2. 変動) 入力済みのデータ数に応じて変化させる．$error\_rate$ が変動する場合は入力済みのデータ数 $item\_num$ を引数とする次の 3 つの関数 (図 11) を用いた．

$Func1$：線形関数 $error\_rate = min(4.21875 * 10^{-3} * item\_num, 0.3)$

$Func2$：指数関数 $error\_rate = min(6.59180 * 10^{-5} * item\_num^2, 0.3)$

$Func3$：対数関数 $error\_rate = min(1.48931 * 10^{-1} * \log(item\_num + 1), 0.3)$

上記 3 つの関数においては，被験者実験の結果 [4] を元に $item\_num = 64$ の時に $error\_rate \div 0.27$ になるように定数項を決めた．また，$error\_rate$ の上限は 0.3 と設定した．なぜならば，S-DE においては入力すべきデータ数が多いため，入力済みのデータ数も多くなり，$error\_rate$ が非常に高くなってしまい現実的な値でなくなってしまうからである．つまり，$error\_rate$ が非常に高く，絶対に間違った処理を行う状態になる．このような状態は実際にワーカに行ってもらう場合では発生しないため，$error\_rate$ に上限を設けた．

6.2 データセット

データ列挙の対象として次の 3 種類のデータセットを利用した．

(データセット A): 新潟県の公営 (国営・県営・市営・町営・外部委託を含む) のプール名． 新潟県の公営 (国営・県営・市営・町営・外部委託を含む) のプール名のデータである．正解集合のデータ数は 101 個である．このデータセットは本提案手法を用

いて被験者実験の際に使用されたデータ [5] と同様であり，タスク生成プランについても同様のものを使用する．

(データセット B): 国連の加盟国の国名．国連に加盟している国名のデータである．正解集合のデータ数は 193 個（[18] の 2011 年加盟まで）である．国連の加盟国のデータは Trushkowsky ら [13] の実験でもデータ列挙の対象としている．タスク生成プランについては，外務省のサイト [19] に記載されている地域を用いて，3 階層とし，各階層の完了合意回数は 2 回とするタスク生成プランを作成し，使用する．

(データセット C): 256 個の要素からなる合成データ．機械的に作成した合成データである．このデータセットは末端ノードがデータ数と同じ 256 の完全 2，4，16 分木で構成できる構造を持つ．タスク生成プランについては，その構造（2，4，16 分木）を用いて，9, 5, 3 階層とし，各階層の完了合意回数は 2 回とするタスク生成プランを作成し，使用する．

6.3 比較方法

単純手法と提案手法の再現率と適合率を比較する際に，条件を合わせるために，単純手法では，提案手法のデータ生成プランの完了合意回数と同じ回数の完了合意が行われる時にタスクが完了するように設定をした．つまり，本シミュレーションにおいては全てのタスク生成プランにおいて完了合意回数は 2 回なので，単純手法でタスクが完了する際には 2 回の完了合意を必要とする．また，それぞれのシミュレーションは 1000 回実行し，それぞれの結果を平均したもので比較をする．ただし，シミュレーションの出力結果が 0 になる場合の適合率については 1 として扱う．なぜならば，出力がなく，間違ったデータも出力していないからである．

6.4 シミュレーション実験の結果

6.4.1 データセット A, B のデータ列挙終了時の再現率と適合率

本節では，ワーカがタスクを間違えて処理する確率をいろいろと変化させた場合の提案手法と単純手法のシミュレーション結果（再現率と適合率）について説明する．

error_rate が固定の場合．図 12, 13 は *error_rate* が固定の場合のデータセット A, B の再現率と適合率を示したグラフである．再現率の共通の特徴として *error_rate* が高い時と低い時では S-DE と DC-DE がともに 1 または 0 に収束している．これは，タスクの完了の合意に多数決を用いているからである．なぜならば，*error_rate* が 0 に近いときは，間違いが非常に起きにくいため再現率は 1 となり，逆に *error_rate* が 1 に近いときは正しい処理が非常に起きにくいため，再現率は 0 となるからである．そのため，*error_rate* が 0.2 − 0.4 程度の中間に注目すると，DC-DE は S-DE よりも高い再現率であることが確認できる．一方，適合率については，S-DE のほうが DC-DE よりも高い値になった．なぜならば，DC-DE は出力されるデータも多く，間違いも多く含まれるため適合率は低くなるからである．逆に，S-DE では入力されるデータ数が少なく，全体的にも間違いの数は少ないため適合率は高くなる．

error_rate が変動の場合．*error_rate* を 6.1.2 節の関数を用いて変化させた場合の再現率・適合率を図 14 に示す．実際に

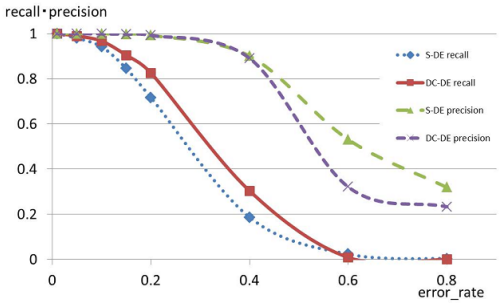

図 12　*error_rate* が固定の場合のデータセット A のシミュレーション結果

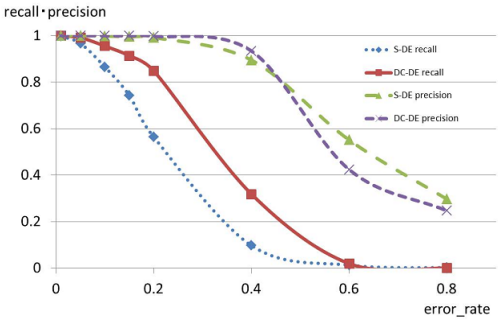

図 13　*error_rate* が固定の場合のデータセット B のシミュレーション結果

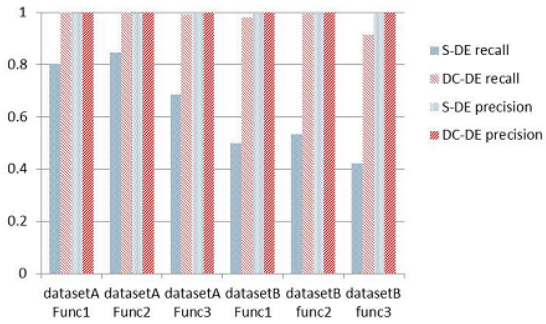

図 14　*error_rate* に *Func* を用いた場合のデータセット A，Bのシミュレーション結果

はワーカがタスクを間違える確率 (*error_rate*) が入力されているデータ数に応じてどのように振る舞うか予測は困難である．しかし，シミュレーションを行った 3 つの関数においては DC-DE のほうが S-DE より再現率が高く，適合率においてはどちらも 0.9 以上の高い値であることから，S-DE より DC-DE のほうが一定の有用性があることが言える．

6.4.2 データセット C のデータ列挙終了時の再現率

本節では，提案手法における分割数の違いによるデータ列挙の再現率をシミュレーションした結果について説明する．データセット C において *error_rate* が一定の場合に階層が 9，5，3（2,4,16 分木）のタスク生成プランを用いて，シミュレーションを行った．その結果を図 15 に示す．*error_rate* = 0.1 までは全ての木においての DC-DE の再現率に大きな違いは見られないが 0.2 のあたりから他のプランと比較して 4 分木の再現率が高くなっていることがわかる．

4 分木の再現率が高い原因を詳しく調べるために，データセットを木構造にした場合の各高さ（階層）における最終結果の最大再現率を調べた．ここで，各高さにおける最終結果の最大再

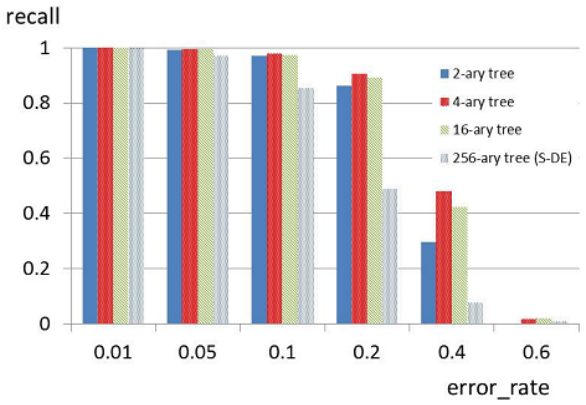

図 15　データセット C のシミュレーション結果

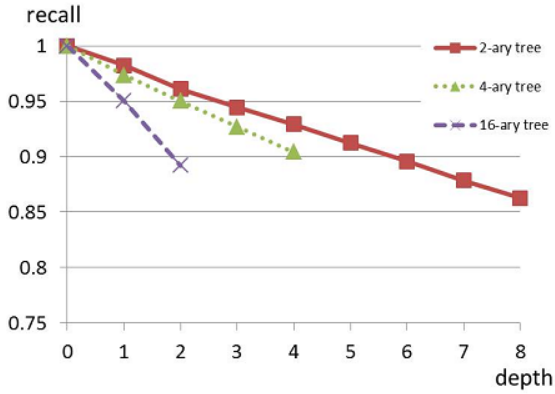

図 16　$error\_rate = 0.2$ の時の各高さにおける最終結果の最大再現率

現率とは，ある高さのデータ列挙が終了したときに，この高さの結果から末端ノードまで間違いがない場合に得られる最大の再現率のことである．各高における最終結果の最大再現率の変化を図 16 に示す．各階層毎に 2，4，16 分木はそれぞれ各階層毎に平均で 0.017，0.024，0.054 ずつ再現率が下がっていることがわかる．この結果から，集めるデータ数（分木数）が大きいほど，各階層毎に下がる再現率は大きくなる．また，最終的な再現率は 1 − (各階層で下がる再現率の値 × 階層の高さ) で求めることができることがわかる．このように，何階層に分割すれば高い再現率になるかは，各タスクで列挙すべきデータ数と階層数が関係するため，一概に分割を「多くした方が良い」や「少なくした方が良い」等は言えない．

今回のシミュレーションにおいては，4 分木で 5 階層のタスク生成プランを用いたシミュレーションが各階層毎で下がる再現率の値と階層の高さの掛け合わせた値が一番小さく，一番再現率が高いという結果となった．

7. おわりに

本稿では，マイクロタスク型クラウドウドソーシングクラウドソーシングプラットフォームにおいて分割統治法を用いたデータ列挙の手法について説明した．また，本提案手法において次の 2 つを確認するためのシミュレーションを行った．(1) ワーカがタスクを間違えて処理する確率をいろいろと変化させた場合，提案手法は単純な手法と比べて再現率は高くなるのか．(2) 分割が多いタスク生成プランと分割が少ないタスク生成プランのどちらが高い再現率になるのか．

シミュレーションの結果，(1) ワーカがタスクを間違えて処理する確率がよほど高くならない限り，提案手法のほうが再現率が高くなることが確認できた．(2) 何階層に分割すれば，高い再現率になるかは，各タスクで列挙すべきデータ数と階層数が関係するため，一概に分割を多くした方が良いことや少なくした方が良いことは言えないことがわかった．今後の課題としては，タスク生成プランの作成支援やタスク生成プランの妥当性の判断手法の検討，他の関連問題 (インセンティブなど) と組み合わせた検証などがあげられる．

謝　　辞

ゼミなどにおいてコメントいただきました，筑波大学 杉本重雄教授，阪口哲男准教授，永森光晴講師に感謝いたします．本研究の一部は JST さきがけおよび科研費基盤研究 (#25240012) の支援による．

文　　献

[1] 食べログ. http://tabelog.com/
[2] Amazon Mechanical Turk. https://www.mturk.com/mturk/
[3] Yahoo!クラウドソーシング. http://crowdsourcing.yahoo.co.jp/
[4] 青木秀人，森嶋厚行，三津石智巳 ”クラウドソーシングによるデータ収集のためのタスク生成手法の提案” deim2013, pp. 1-7, 2013-3
[5] Aoki Hideto, Atsuyuki Morishima. A Divide-and-Conquer Approach for Crowdsourced Data Enumerataion. SocInfo 2013, pp. 60-74, November 2013
[6] Adam Marcus, Eugene Wu, David R. Karger, Samuel Madden, Robert C. Miller: Human-powered Sorts and Joins. PVLDB 5(1), 13-24 (2011)
[7] Adam Marcus, Eugene Wu, Samuel Madden, Robert C. Miller: Crowdsourced Databases: Query Processing with People. CIDR 2011, 211-214 (2011)
[8] Michael J. Franklin, Donald Kossmann, Tim Kraska, Sukriti Ramesh, Reynold Xin: CrowdDB: answering queries with crowdsourcing. SIGMOD 2011, 61-72 (2011)
[9] Aditya G. Parameswaran, Neoklis Polyzotis: Answering Queries using Humans, Algorithms and Databases. CIDR 2011, 160-166 (2011)
[10] Yahoo! ANSWERS. http://answers.yahoo.com/
[11] OK Wave. http://okwave.jp/
[12] 人力検索はてな. http://q.hatena.ne.jp/
[13] Beth Trushkowsky, Tim Kraska, Michael J. Franklin, Purnamrita Sarkar: Crowdsourced enumeration queries. ICDE 2013, 673-684 (2013)
[14] Anand Pramod Kulkarni, Matthew Can, Bjorn Hartmann: Collaboratively crowdsourcing workflows with turkomatic. CSCW 2012, 1003-1012 (2012)
[15] Atsuyuki Morishima, Norihide Shinagawa, Shoji Mochizuki. “The Power of Integrated Abstraction for Data-Centric Human/Machine Computations”. VLDS 2011: 7-10
[16] gwap.com. “ESP Game” http://www.gwap.com/gwap/gamesPreview/espgame/
[17] Shaili Jain, David C. Parkes: A Game-Theoretic Analysis of Games with a Purpose. WINE 2008: 342-350
[18] 国際連合広報センター 国連加盟国加盟年順序. http://www.unic.or.jp/info/un/un_organization/member_nations/chronologicalorder/
[19] 外務省 各国・地域情勢. http://www.mofa.go.jp/mofaj/area/index.html