クラウドソーシングによるデータ列挙処理に関する研究

筑波大学
図書館情報メディア研究科
2014 年 3 月
青木 秀人

目次

第 1 章

はじめに

近年，機械では検索困難な情報を人手で用いて検索する人力検索 (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 Enumerattion) 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 1.1 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できず，結果として，適切な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．

本論文では，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**をクラウドソーシングによるデータ列挙に適用する．我々が知る限り，マイクロタスク型のクラウドソーシングプラット

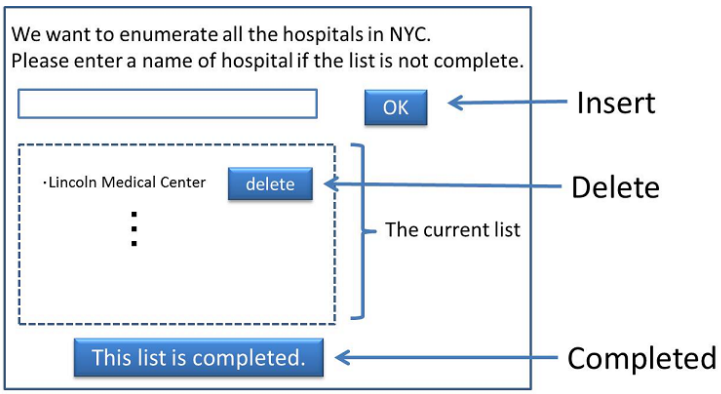

図 1.1　A task for datascript enumeration

フォーム上における分割統治手法によるデータ列挙処理は存在しない.

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 1.2 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に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.

本論文の貢献は次の 2 つである.

(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上における分割統治手法によるデータ列挙処理. 提案手法の特徴はワーカが分割統治によるデータ列挙のためのタスクの生成に参加することである．つまり，リクエスタは予め分割されたタスクを用意する必要がない．このアイディアは，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.

また，どのように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上において分割統治手法をデータ列挙に適用するかは明らかではない．そこで，本手法では分割統治によるデータ列挙を実現するために，分割の基準となる**タスク生成プラン**を与える．タスク生成プランとはデータ列挙の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設計図である.

(2) 提案手法の評価

本論文では提案手法を評価するために理論的評価・被験者実験による評価・シミュレーションによる評価の 3 つの評価を行った．まず，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正しくなることを示した. 次に，被験者実験による評価に

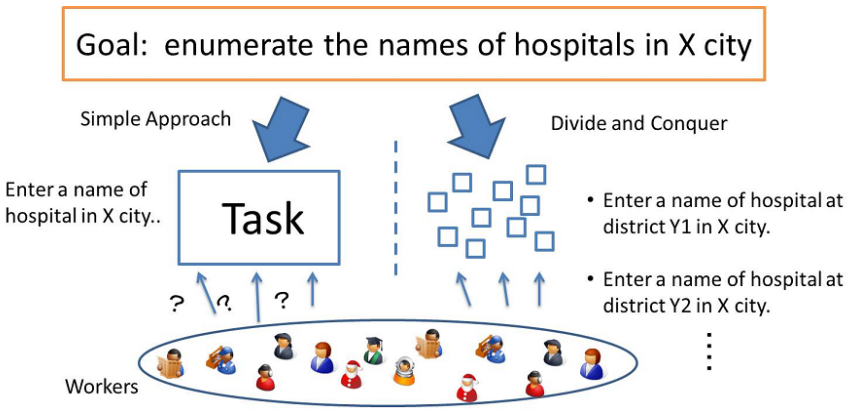

図 1.2　Data enumeration by a set of smaller tasks

おいて，実際にデータ列挙を被験者に行ってもらい，次の 2 つのことを確認した．(1) 分割統治によるデータ列挙手法のほうが単純な手法よりも再現率が高くなること (2) 提案手法は，ワーカが 1 つのタスクを処理するのに要する時間が短く，さらにタスクを並列して処理することが可能であることから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．最後に，シミュレーションによる評価においていくつかの条件で提案手法と単純手法の比較を行った．シミュレーションの結果より，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 章では関連研究について説明する．3 章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 章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 章では提案手法について説明する．6 章では提案手法の評価について説明する．7 章では本論文全体のまとめについて述べる．

第 2 章

関連研究

1 章で説明した通り，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，ソート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. [4] [5] [6] [7].

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 Q&A サービスで使われている [8][9][10]．1 章で説明した通り，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 [11] は生物学や 統計学の分野のテクニックを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も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 [12] 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 [4] 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 [13] [14] がある．論文 [15] 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．

第 3 章

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

本章では，本研究で対象とする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 3.1)．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 3.2 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 3.2 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論を行う．

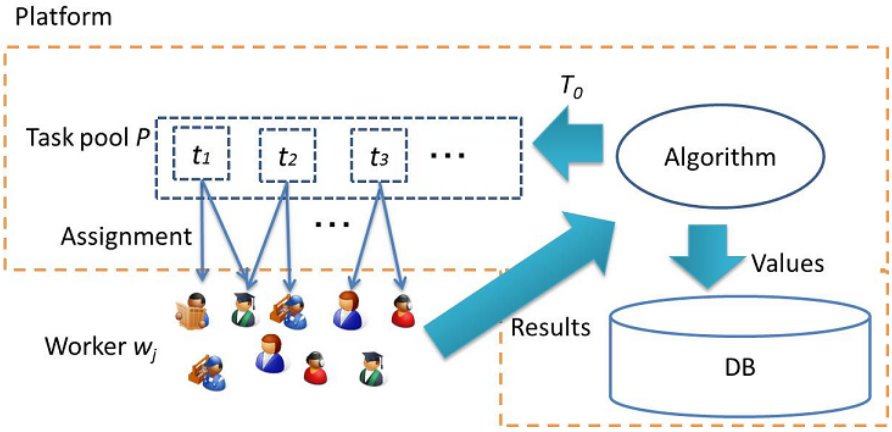

図 3.1　Overview of microtask-based crowdsourcing platform

```
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 3.2　Algorithm for microtask-based crowdsourcing platform

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 1.1) *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.

第 4 章

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

本章では，本論文で扱うデータ列挙について定義し，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単純なデータ列挙について説明する．初めにデータ列挙について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． □

4.2 単純データ列挙の処理の流れ

単純なデータ列挙 (simple crowdsouced data enumeration, 以下,S-DE) の処理の流れについて，筑波大学の研究トピック (*Items(*`Research_topic, "University of Tsukuba"`*)*) を例に挙げて説明する．

1. 初期タスク T_0 を {*EntryTask(*`Research_topic, "University of Tsukuba"`*)*} としてタスクプールに格納する（図 3.2 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 3.2 の 3,4 行目）.
3. ワーカが処理したタスクの結果の処理を行う．ワーカの処理の内容によって次の 3 つの処理の中から 1 つ処理を行う（図 3.2 の 6 行目，図 4.1）.
   - 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 4.1 の 6 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 4.1 の 9 行目).
   - ワーカがタスクが完了していると判断したら，タスクプール P から対応するタスクを削除する（図 4.1 の 12 行目）.

```
 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 4.1　Postprocess for S-DE

4.3　単純な手法における問題

単純なデータ列挙では再現率を重視することは難しい．なぜならば，入力済みのリストのデータが増加し，次の 3 つのことが起きるからである．(1) ワーカは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が難しい．(2) ワーカは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が難しい．(3) ワーカは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難しい．

第 5 章

提案手法

本章では，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おこなうために，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．

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

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．

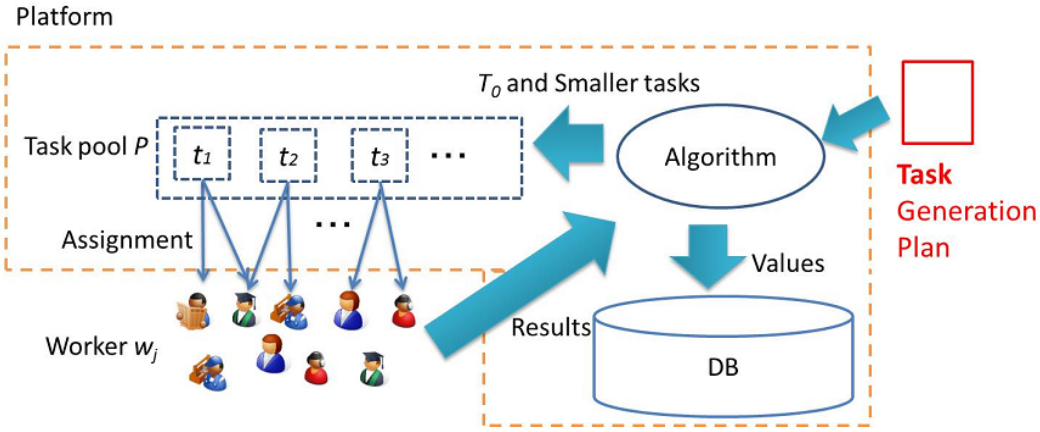

図 5.1　How DC-DE works in microtask-based crowdsourcing platform

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 5.2. 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 5.1 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 3.1) 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）．

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，データ列挙範囲を分割するため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 5.2 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．

図 5.2 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 つの種類のタスクを生成する．(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 5.2 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事が書かれている．

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 $DivisionTask$(詳細について 5.2.1) 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 5.2 においては，研究科単位でなくさらに専攻毎に分割してデータを収集するタスクを生成するかどうか (Task5) は，2 名のワーカで同意を取る必要がある事が書かれている．

生成同意を導入する目的は，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 5.3 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 5.2 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．

さらに，実際のタスク生成プランは，図 5.2 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 $\{\ldots\}$ 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，’$< \ldots >$’ に書かれている (例えば，リレーション $Grad\text{-}Sch$ 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良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 1.1) 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
- ($Completed$) 対象のデータが全て揃っているか判断

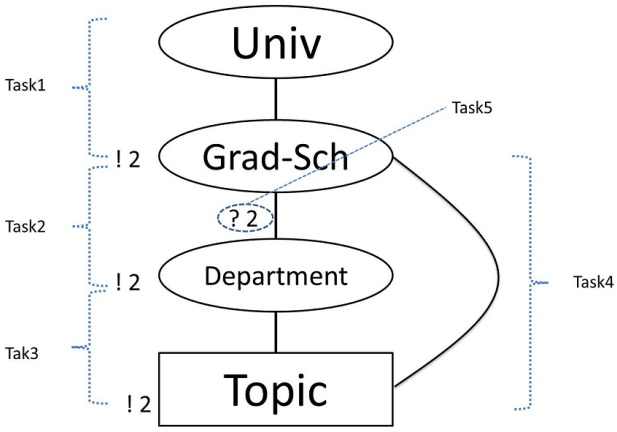

図 5.2　Illustration of a task generation plan

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

図 5.3　Description of a task generation plan

- ($Delete$) 既に入力されたデータを削除

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病院を削除する.

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 (DivideTask) 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 5.4 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

$DivideTask($**"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．

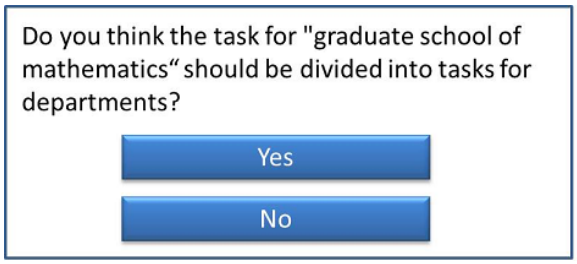

図 5.4　Example of a DivisionTask

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．本アルゴリズムにおいても，図 3.2 に沿った流れで行われる．具体的には次の通りである．

1. 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 5.3 の場合は，$EntryTask(Grad\text{-}Sch,$ `"Tsukuba University"`$)$ を T に登録する．
2. 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. 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 5.5).

   [Case 1: EntryTask においてワーカが “Insert” を行った場合] 追加されたデータをデータベースに格納する (6 行目). 追加されたデータが末端ノード n_m のデータ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．例えば，$n_1 - n_2$ のエッジに対応するタスクをワーカが処理し，データ v_2 を追加したと仮定する．このとき $n_2 - n_3$ のエッジに対応し $EntryTask(n_3, v_2)$ が生成される．タスクを生成する際にさらに 2 つのケースにわかれる 1 つ目のケースは，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，EntryTask をタスクプール P に追加する（8 行目）．2 つ目のケースは，対応するエッジに生成同意回数 $(?y)$ が記述されていル場合，DivisionTask をタスクプール P に追加する（10 行目）．これらの生成したタスクは再度ワーカによって処理され，もとのタスクを小さくしていく．

   [Case 2: EntryTask においてワーカが “Delete” を行った場合] ワーカによって削除すると選

```
 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 5.5　Postprocess for DC-DE

択されたデータをデータベースから削除する (13 行目)．また，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 (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（22 行目）.

第 6 章

評価

本章では提案手法を評価するためにが行った評価実験について説明する．提案手法において次の 2 つのことを評価する．(A) マイクロタスク型クラウドプラットフォームにおいて正しい分割統治によるデータ列挙が実現できるか．(B) 提案手法を行うことで再現率が実際に高くなるのか．この 2 つを確認するために 3 つの評価方法で評価を行った．評価方法とその結果の概要は次の通りである．詳細については次節から順次説明する．

(1) 理論的評価．適切なタスクプランを提案手法に与え，ワーカは適切にタスク処理をするとき，提案手法と単純手法のデータ列挙の結果は等しくなる．

(2) 被験者実験による評価．被験者実験を行い実際に単純手法と提案手法を比較した．提案手法は単純手法よりも高い再現率であった．また，短い時間で処理できるタスクが多いことから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がわかった．

(3) シミュレーションによる評価．ワーカが本来行うタスクの処理をシミュレーションすることで単純手法と提案手法を比較した．その結果，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，提案手法のほうが高い再現率となった．

6.1　理論的評価

本節は提案手法が正しくデータ列挙をすることを示すために理論的評価を行う．
次の定理を示した．

定理 1. ワーカが適切にタスクをおこなうという仮定のもと提案手法に適切なタスク生成プラン *(*$scope_0, item\_type_0$ *を含む)* を与えると出力 *(*V*)* は *Items(*$item\_type_0, scope_0$*)* となる．ただし，ここで適切なタスク生成プランとは最も上の階層の初期値が $scope_0$ と等しく，タスク生成プランから作られるデータ *(*インスタンス*)* の末端が $item\_type_0$ と同じプランのことである．　□

証明 1. 提案手法のプログラムに定理に沿った条件（アサーション）を導入し，それらの条件を満たしていることを確認し，定理を証明する．図 *6.1* に条件を追加したプログラムを示す．

```
1.   P. store(T_0);
//* Q ∧ S *//
2.   While !P.isEmpty() {
//* S ∧ !P.isEmpty() *//
3.       w=getWorker();
4.       t_i=P.assignTask(w);
5.       wait until we get r=t_i.resultOfTaskBy(w);
6.       postprocess(t_i, r);
7.   }
//* S ∧ P.isEmpty → R *//
```

図 6.1　DC-DE algorithm with assertions

ここで V はデータベースに格納された値の集合を表す．また V は末端ノードを列挙するタスクが行われるまでは空である．このプログラムが開始時に前提条件 Q を満たしており，ループを抜けてプログラムが終了するときに事後条件 R を満たせば正しく，データ列挙で行える．条件 S, R は次の通りである．

初期条件 Q：$P = \{EntryTask(n\_2, scope_0)\}$
事後条件 R：V *contains* $Items(item\_type_0, scope_0)$

これらを証明するために，新たに不変条件 S を導入し，常にプログラム中で常に条件を満たしているかを確認する．不変条件 S は次の通りである．

不変条件 S： $Items(item\_type, scope) \subseteq \cup_{t_i \in P} Items(item\_type, t_i.scope) \cup V$

不変条件 S が示していることはタスクプール P にデータを収集する適切なタスクが格納されている，または既にデータが結果として列挙されていることを示している．また，ループの中で常に不変条件 S を満たし，プログラムが正常に終了した場合は事後条件 R を満たすことになる．正常に終了とは $S \wedge P.isEmpty$ となることである．すなわち，タスクプール P に格納されていたタスクが全て終了したこと（タスクプール P が空）である．まず，ループ内ので不変条件 S が常に成立することを示し，次にプログラムが終了することを示す．

ループ開始時点では *1* 行目において初期条件 R を満たしているので，不変条件 S が成立する．ループの中で不変条件 S に影響を与えるのは *6* 行目の $postprocess$ の関数だけである．なぜならば，$postprocess$ の関数内だけでタスクの追加や削除を行っているからである（図 5.5）．*postprocess* のそれぞれのケースについて不変条件 S が成立するか順次確認する．ただし，*Case3* については間違った処理を行わないと生じない *Case* であり，本定理では間違った処理を行わない仮定があるので除く．*Case1, 4* ではタスクはタスクプール P に追加されるだけであるため，不変条件 S が成立している．*Case3* ではタスク完了処理が完了合意回数回行われたときに，タスクプール P から削除されるため，不変条件 S が成立しなくなる可能性がある．このとき，末端ノードのデータを入力するタスクであれば，完了判断時にそのタスクの $scope$ の範囲を網羅したデータが *V* に格納されている．タスクが末端ノードに対するタスクではない場合には，タスク分割された小さなタスクがタスクプール P に格納されている．どちらの場合でも，不変条件 S の条件が成立している．これらのことから *postprocess* の関数の中では常に不変条件 S が成立する．

タスクプール P に格納されるタスクの数は最終的に *0* になるので常にループから終了する．前段落で説明した通り，タスクがタスクプール P に追加されるのは，*Case1, 4* である．追加されるタスクの数は列挙するデータ数が有限であれば，有限になる．なぜならば，生成されるタスクの数は基本的に，与えられたタスク生成プランの階層構造によって作られた木の内部のノード数になるからです．また，$postprocess$ では，どの *Case* でもタスクをタスクプール P に追加，もしくはタスクプール P から削除している．これらのことからタスクプール P は最終的に空になり，プログラムが終了する． □

重要な特徴として，提案手法はワーカがタスク生成に参加する．したがって，不適切なタスクを生成する頻度は 1 つの関心事である．次の定理に従うことは明らかである．

定理 2. タスク t とタスク生成プラン p が与えられたとき，$correct(t)$ はタスク t が適切なタスクである事を示す述語である．また，N をタスク生成プラン p で与えられた最小の同意回数とする．このとき，タスク生成プラン p によって生成される全てのタスクが $correct(t)$ となる必要な条件は，ワーカが不適切な処理を同じタスクに対して N 回連続して行わないことである． □

この定理は適切に合意回数を設定することが重要である事を示している，ただし，同意回数を何回に設定すべきかについては本研究では扱わない．

6.2　被験者実験による評価

提案手法を評価するために，実際に被験者に提案手法と単純手法を用いてデータ列挙を行い，比較した．

6.2.1　実験方法

データ列挙対象. 本実験では，収集対象として「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」を選択した．このデータ収集を選択した理由は，Web 上にこの項目だけをまとめて提供しているページが存在せず (簡単に一覧が手に入らず)，単純には見つからないことが確認できているからである．データ入力者は Web 検索エンジンの利用が許されるため，データ列挙の結果がそのまま Web 掲載されているようなデータ列挙では実験目的にはそぐわない．したがって，この条件は重要である．事前に，筆者が Web 検索などを通じて手作業で正解集合を作成した．正解集合のプール名の個数は 101 個である．

ワーカ． 本実験のために，2 つのワーカのグループ A, B (それぞれ 6 人) を用意した．グループ A は，分割を行わない単純な手法（以下，単純手法）でデータ収集を行ない，一方，グループ B は，提案手法（以下，分割手法）を用いてデータ収集を行う．実験中は，ワーカ同士のコミュニケーションを許可しなかった．ワーカには事前にタスクのタイプと操作方法について説明を行なった．

タスク生成プラン． 分割手法では，図 6.2 に示すタスク生成プランを利用した．

```
[ 県{新潟県},
    地方 [!2]<地方名>,
    市町村 [!2,*]<市町村名>,
    区 [!2,?2]<区名>,
    *公営プール [!2]<公営プール名>]
```

図 6.2　Task-generation plan for the experiment

提案手法合わせて，単純手法においても 2 回の完了合意が行なわれるとタスクがが完了するように設定し実験をおこなった．

6.2.2　被験者実験の結果

再現率・適合率． 実験の結果，単純な手法の場合は 62 個，分割手法では 78 個のプールがワーカによって入力された．正解集合と比較した場合の再現率・適合率を表 6.1 に示す．予想通り，再現率

表 6.1　Recall and precision

	recall	precision	F-measure
S-DE	0.57	0.97	0.72
DC-DE	0.75	0.99	0.85

表 6.2　Number of performed tasks

	EntryTask			DivisionTask	Total
	insert	completed	delete		
S-DE	81	5	17	0	103
DC-DE	179	141	44	64	428
S-DE’	101	2	0	0	103
DC-DE’	133	86	0	60	279

は分割手法の方が向上していることがわかる．単純手法の方が適合率が高かった理由は，主に 2 人のワーカによってデータの入力が行なわれて，他のユーザは積極的にデータ入力せずに，データの削除やデータが揃っているかの判断をしために，重複等が生じにくかったと考えられる．

総タスク処理数．単純・分割手法の実験終了時までに行われたタスク処理数とその内訳を表 6.1 に内訳を示す．単純な手法において行われたタスク数は 103，提案手法は 428 であった．これとは別に，ワーカがタスクを行なう際に，理想的にタスクを行なった (間違えなかった) 場合の単純，提案手法タスク処理数を $S-DE', DC-DE'$ とする．この場合，$S-DE'$ のタスク処理数は 93, $DC-DE'$ のタスク処理数 は 279 となる（3, 4 行目）．実際に実験を行った結果のタスク処理数であるので，間違った値を入力したり取り消したりするため，理想的な場合に比べると数が増える．

タスク処理時間とタスク処理時間分布．あるタスク t があるとき，ワーカが t を処理するためのタスク画面を表示してからボタンを押すまでの時間を処理時間とする．タスク処理時間分布を図 6.3 に示す．単純手法における 1 タスクあたりの平均処理時間は約 97 秒であり，分割手法では 58 秒であった．分割手法では，タスク処理時間が 3 分以下のタスクが 92 %と短いタスクが多く，クラウドソーシングに向いたマイクロタスクを生成できていると言える．

提案手法のほうが 1 タスクあたりの平均処理時間が短かった原因として，1 つのタスクで列挙すべきデータ数の違いが挙げられる．提案手法は 1 タスクあたり平均して，3.1 個のデータを列挙する必要がある．一方，単純手法では 1 つのタスクで全てのデータを列挙する必要があるので 101 個となる．提案手法における 1 タスクあたりの列挙すべきデータ数の分布を図 6.4 に示す．

タスク処理時間の単純総和を計算すると，単純手法の総タスク処理時間は 10043 秒（約 167 分）であり，分割手法の総タスク処理時間は 24672 秒（約 411 分）であった．すなわち，タスク処理時間の単純総和に関しては，分割手法は単純手法と比較して約 2.5 倍の総タスク処理時間を必要とした．

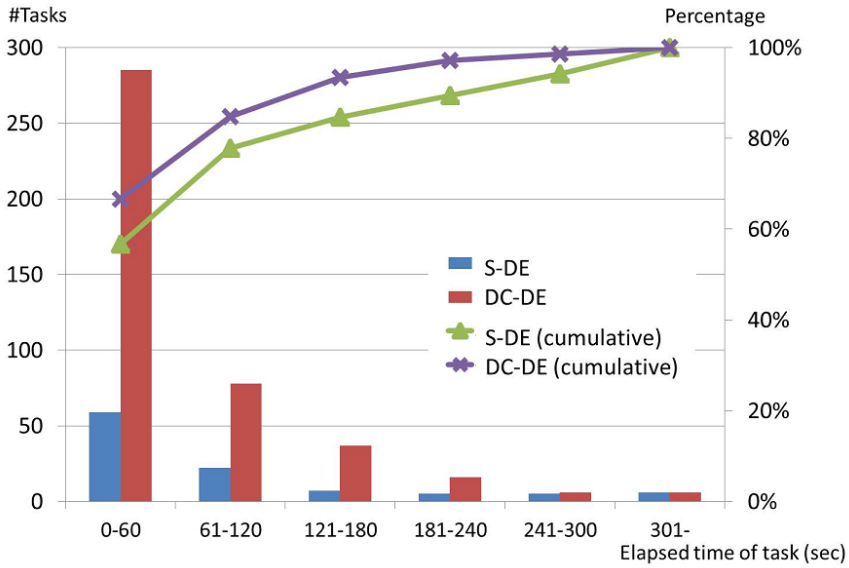

図 6.3　Distribution of elapsed times for performing tasks

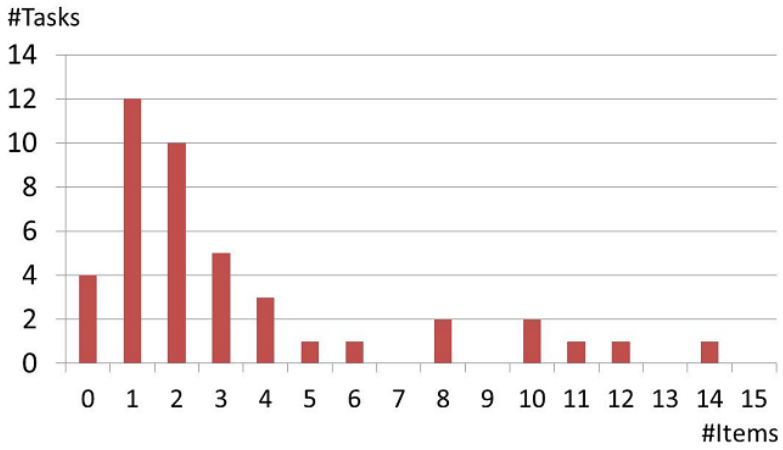

図 6.4　Distribution of tasks for having items

しかし，分割手法においては，単純手法と比較してタスク間の依存関係が低くなり，独立に (並列して) 処理可能なタスクが数多く生成される．したがって，そのため，クラウドソーシングのワーカが十分多く確保できる場合には，並列して処理を行う事により，より短い時間で処理を終了可能である．独立作業可能なタスクの処理を並列化した場合をシュミュレーションすると，30 人以上のワーカが存在する場合には，分割手法は 3652 秒（約 61 分）で処理が終了することがわかった．

6.3　シミュレーションによる評価

本節では，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．

6.3.1　シミュレーション方法

6.3.1.1　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 6.3.1.1 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．

(2-B) (1) で (B) に決定した場合，表 6.3.1.1 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．

表 6.3　Pattern of worker’s action

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
		データ列挙の完了判断
	$DivisionTask$	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
		データ列挙の完了判断
	$DivisionTask$	間違った分割判断

6.3.1.2　$error\_rate$ 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 6.5) 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，被験者実験の結果を元に $item\_num = 64$ の時に $error\_rate \fallingdotseq 0.27$ 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．

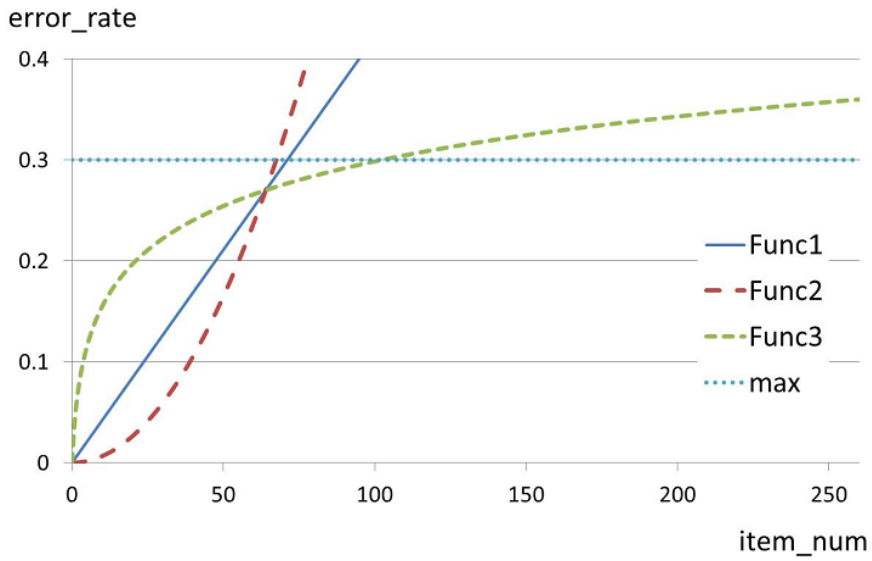

図 6.5　Function of error_rate

6.3.2　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 [**?**] 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（[16] の 2011 年加盟まで）である．国連の加盟国のデータは Trushkowsky ら [11] 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 [17] 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．

6.3.3　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た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．

6.3.4　シミュレーション実験の結果

6.3.4.1　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 6.6, 6.7 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．

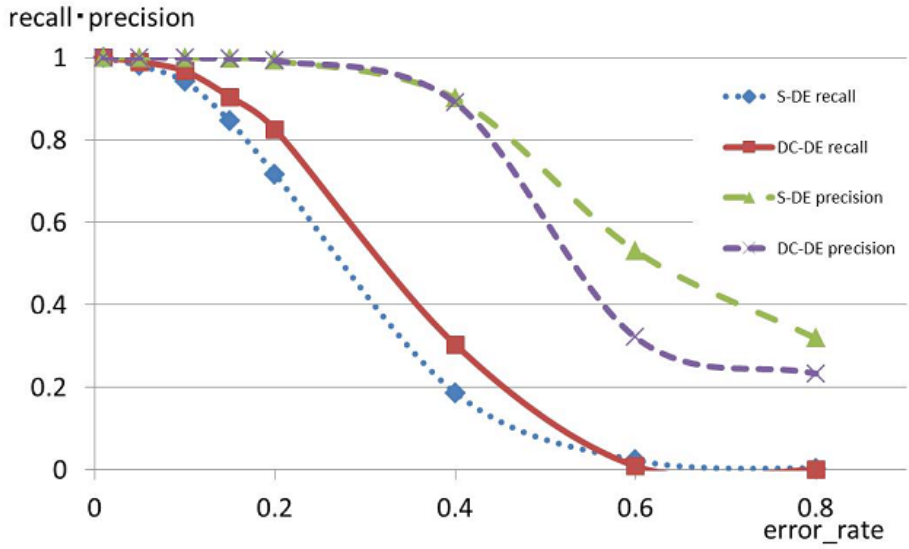

図 6.6　Result of data enumeration for datasetA

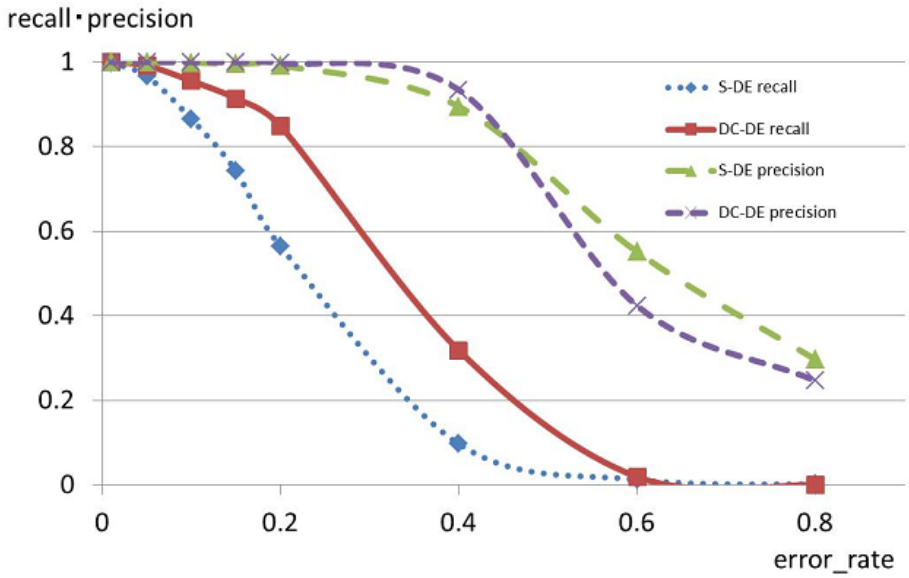

図 6.7　Result of data enumeration for datasetB

error_rate が変動の場合．$error\_rate$ を 6.3.1.2 節の関数を用いて変化させた場合の再現率・適合率を図 6.8 に示す．実際に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．

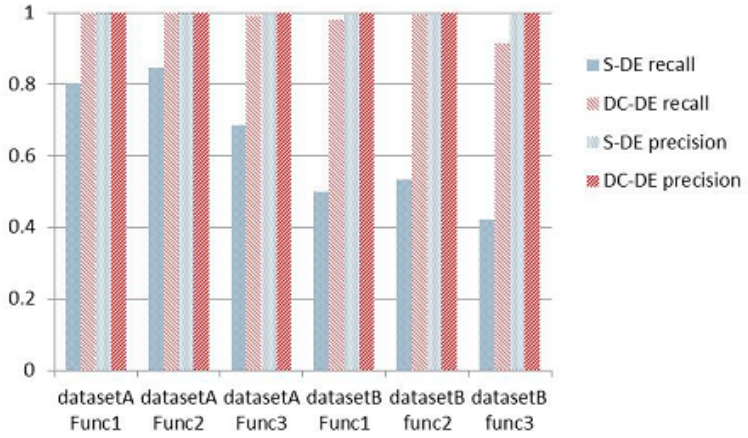

図 6.8　Result of data enumeration on the functions

6.3.4.2　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 6.9 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．

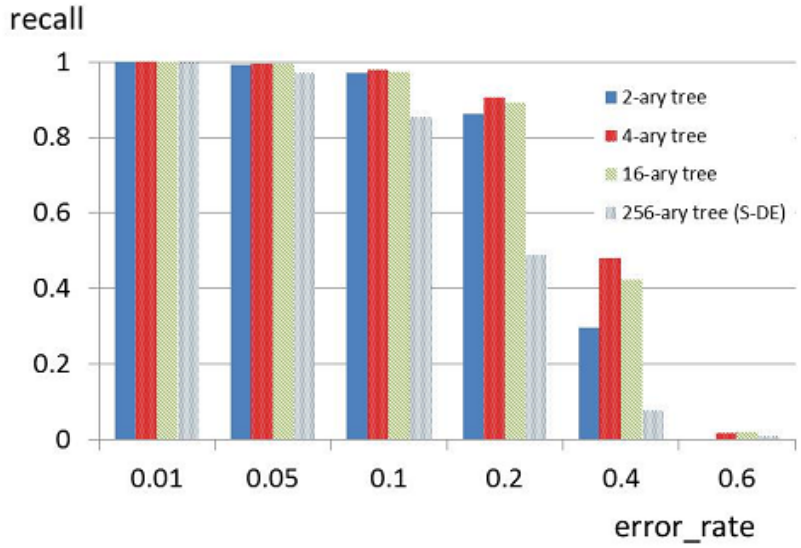

図 6.9　Result of data enumeration for datasetC

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 6.10 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．

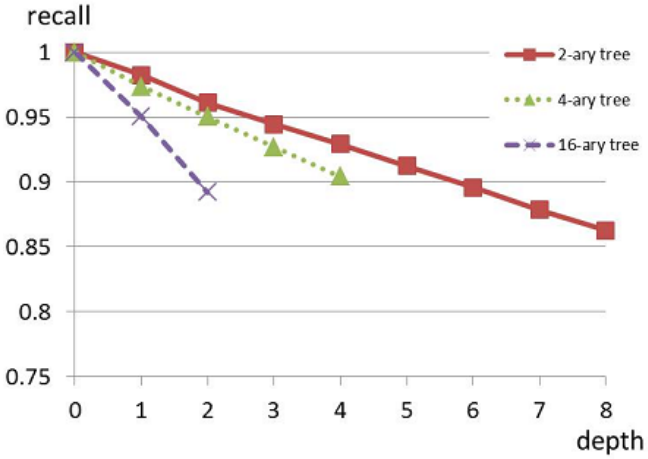

図 6.10　Recall of each layer on the tree

第 7 章

おわりに

本論文では，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を手法を提案した．データ収集の再現率を上げるためには問題を小さく限定したタスクを数多く用意する事が重要であるが，具体的にどのタスクを用意すべきかは問題固有であり，かつ一般にタスク数も多くなることから，個々の問題毎にリクエスタがそれらのタスクを用意する事は煩わしい．提案手法の特徴は，リクエスタが個々のタスクを用意する代わりに事前にタスク分割の設計図であるタスク生成プランをシステムに与えることで，ワーカのタスク処理の結果に応じて問題を小さく限定したタスクを動的に生成する事である．

本論文では，本手法の説明に加え，本手法の有効性を確認するために次の 3 つの評価結果を示した．(1) 理論的評価．提案手法が，ある条件の下では単純にデータ列挙を行った場合と同じ結果をもたらすことを示した．(2) 被験者実験による評価．あるデータを対象に単純手法と比較を行い，提案手法のほうが再現率が高くなることを示した．(3) シミュレーション評価．様々な条件でシミュレーションを行い，ワーカのエラー率がよほど高くならないかぎり，提案手法が単純手法より一般的に再現率が高くなることを示した．

参考文献

[1] 食べログ. http://tabelog.com/ , (参照 2014-01-15)

[2] Amazon Mechanical Turk. https://www.mturk.com/mturk/ , (参照 2014-01-15)

[3] Yahoo!クラウドソーシング. http://crowdsourcing.yahoo.co.jp/ , (参照 2014-01-15)

[4] Adam Marcus, Eugene Wu, David R. Karger, Samuel Madden, Robert C. Miller: Human-powered Sorts and Joins. PVLDB 5(1), pp.13-24 (2011)

[5] Adam Marcus, Eugene Wu, Samuel Madden, Robert C. Miller: Crowdsourced Databases: Query Processing with People. CIDR 2011, pp.211-214 (2011)

[6] Michael J. Franklin, Donald Kossmann, Tim Kraska, Sukriti Ramesh, Reynold Xin: CrowdDB: answering queries with crowdsourcing. SIGMOD 2011, pp.61-72 (2011)

[7] Aditya G. Parameswaran, Neoklis Polyzotis: Answering Queries using Humans, Algorithms and Databases. CIDR 2011, pp.160-166 (2011)

[8] Yahoo! ANSWERS. http://answers.yahoo.com/ , (参照 2014-01-15)

[9] OK Wave. http://okwave.jp/ , (参照 2014-01-15)

[10] 人力検索はてな. http://q.hatena.ne.jp/, (参照 2014-01-15)

[11] Beth Trushkowsky, Tim Kraska, Michael J. Franklin, Purnamrita Sarkar: Crowdsourced enumeration queries. ICDE 2013, pp.673-684 (2013)

[12] Anand Pramod Kulkarni, Matthew Can, Bjorn Hartmann: Collaboratively crowdsourcing workflows with turkomatic. CSCW 2012, pp.1003-1012 (2012)

[13] Atsuyuki Morishima, Norihide Shinagawa, Shoji Mochizuki. “The Power of Integrated Abstraction for Data-Centric Human/Machine Computations”. VLDS 2011, pp.7-10

[14] gwap.com. “ESP Game” http://www.gwap.com/gwap/gamesPreview/espgame/ , (参照 2014-01-15)

[15] Shaili Jain, David C. Parkes: A Game-Theoretic Analysis of Games with a Purpose. WINE 2008 pp.342-350

[16] 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/ , (参照 2014-01-15)

[17] 外務省 各国・地域情勢. http://www.mofa.go.jp/mofaj/area/index.html , (参照 2014-01-15)

研究に関連する発表論文

国際会議論文

- Aoki Hideto, Atsuyuki Morishima. A Divide-and-Conquer Approach for Crowdsourced Data Enumerataion. SocInfo 2013, pp. 60-74, November 2013.

国内研究会論文

- 三津石智巳, 青木秀人, 福角駿, 宮田愛, 森嶋厚行, 品川徳秀, 逸村裕.「筑波大学におけるクラ ウドソーシング技術を用いたマイクロボランティアの試み」, 大学 ICT 推進協議会 2012 年度年次大会, pp. 1-5, 2012-12.
- 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.
- 青木秀人，森嶋厚行，”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” DEIM2014, pp. 1-8, 2014-3.(投稿中)

謝辞

本研究を進めるにあたり，指導教員である森嶋厚行教授に深く感謝致します．大学学群から 3 年間，私を熱心に御指導して下さいました．ここまで研究を進め，成長することができたのは森嶋先生の熱心なご指導のおかげです．ありがとうございました．

また，森嶋研究室に所属する方々にも大変お世話になりました．2 年間ご一緒した品川先生には，ゼミの時や打ち合わせの時に様々なアドバイスをいただきました．既に卒業されている方も含みますが，三津石さん，安永さんの先輩方には研究に困った時やゼミなどで日ごろからアドバイスをいただきました．先輩方から頂く適切なアドバイスは，私の研究生活においてなくてはならないものでした．さらに，同期である福角君，後輩である権守君，丹治君，あべ松君，品川君，平木さん，櫻井さんにも大変お世話になりました．論文の添削から研究に関する鋭い指摘，日々の雑談まで，充実した研究生活が送れたのは皆さんのおかげです．その他，異なった研究分野の話をしてくださった松原先生，学会参加時の書類作成時等にお世話になりました秘書の篠崎さん，研究プロジェクトを支えてくださった樋口さんにも感謝いたします．

最後になりましたが，合同ゼミなどでご指導いただきました杉本重雄教授，阪口哲夫准教授，永森光晴講師に，深く感謝いたします．

付録 A

評価で使用したデータセット

本付録では，評価の際に用いたデータセットの詳細について説明する．ただし，データセット C に関しては，完全 N 分木であり，特別な構造を持たないので説明しない．

A.1　データセット A

データセット A は新潟県の公営 (国営・県営・市営・町営・外部委託を含む) のプールである．データの内容は 2012 年 11 月時点で，各市町村等の公式ページおよび，観光サイト等を用いて調査し，運営が確認されたプール (101 個) である．データセット A の正解集合を地方毎 (上越地方，中越地方，下越地方，佐渡地方) に表 A1 - A4 に示す．

A.2　データセット B

データセット B は国連の加盟国である．加盟国は, 国際連合広報センター [16] の国連加盟国加盟年順序に記載されている 2011 年加盟分 (193 カ国) までとした．地域については，外務省のサイト [17] に記載されている 7 地域を用いた．データセット B の正解集合の階層毎の個数について表 A5 に示す．ただし，具体的な国名については，シミュレーションには不要であるため，本付録では記載しない．

表 A.1　Contents of datasetA in Joetsu area

市町村	区	
上越市		上越市須城Ｂ&Ｇ海洋センター 上越市立オールシーズンプール 上越市安塚 B&G 海洋センタープール 上越市浦川原プール 上越市牧プール 上越市柿崎屋内水泳プール 市民プール（リージョンプラザ上越） 上越市板倉洗心プール
糸魚川市		青海屋内水泳プールサンドリームおうみ 能生 B&G 海洋センタープール
妙高市		水夢ランドあらい

表 A.2　Contents of datasetA in Chuetsu area

市町村	区	
長岡市		越路 B&G 海洋センター 悠久山プール 希望が丘プール 和島 B&G 海洋センター 小国プール 寺泊海浜公園プール 新潟県立長岡屋内総合プール アクアーレ長岡 長岡市青少年文化センタープール えちご川口温泉内健康増進回復施設プール
三条市		三条市民プール
柏崎市		港公園（プール） 新潟県立柏崎アクアパーク 柏崎市スポーツハウス
小千谷市		小千谷市市民プール
加茂市		加茂市市民プール 加茂市子供プール 加茂市温水プール
十日町市		松代プール 十日町市民プール 室野プール 松之山プール 健康増進施設「ひだまりプール」 十日町体力づくり支援センター
見附市		見附市市民プール
魚沼市		小出北部プール 広神プール 下条プール
南魚沼市		寿和温泉温水プール 大和 B&G 海洋センター 南魚沼スポーツコミュニティーセンター（ディスポート南魚沼）
出雲崎町		出雲崎町町民プール
湯沢町		レジャープール（オーロラ）
津南町		クアハウス津南（津南温泉スポーツランド） グリーンピア津南
刈羽村		「ラピカ」体育施設

表 A.3　Contents of datasetA in Kaetsu area

市町村	区	
新潟市	北区	遊水館
	東区	下山スポーツセンター
		山の下海浜公園
	中央区	東公園児童プール
		西海岸公園市営プール
		鳥屋野総合体育館
	江南区	亀田総合体育館
		新潟県障害者交流センター温水プール
	秋葉区	新津 B&G 海洋センター
		新津プール
	南区	味方 B&G 海洋センター
	西区	西総合スポーツセンター
		アクアパークにいがた
	西蒲区	中之口 B&G 海洋センタープール
新発田市		新発田市市民プール
		紫雲寺プール
村上市		荒川温水プール
		神林プール
燕市		ビジョンよしだ
		B&G 海洋センタープール
		燕市市民プール
		分水プール
		吉田プール
五泉市		五泉市市民プール
		五泉市村松プール
		五泉市戸倉プール
阿賀野市		（安田）B&G 海洋センター
胎内市		B&G プール
		クアハウスたいない
阿賀町		当麻プール
		阿賀町上川 B&G 海洋センター
		阿賀町津川 B&G プール
		阿賀町三川 B&G 海洋センター
関川村		下関地区水泳プール
		大島地区水泳プール
		四ヶ字地区水泳プール
		高田地区水泳プール
		片貝地区水泳プール
		金丸地区水泳プール
		女川地区水泳プール
		川北地区水泳プール
		霧出地区水泳プール
		七ヶ谷地区水泳プール

表 A.4　Contents of datasetA in Sado area

市町村	区	
佐渡市		佐和田プール 金井プール 畑野プール 小倉プール 羽茂 B&G 海洋センター 小木プール 小木 B&G 海洋センター 佐渡スポーツハウス 羽茂プール 赤泊プール ワイドブルーあいかわ

表 A.5　Number of member countries of UN by area

地域	数
アジア	23
大洋州	14
北米	2
中南米	33
欧州	52
中東	15
アフリカ	54
合計	193