Crowdsourced Join Pre-filterによる Human-powered Join処理効率化の評価

三津石智巳† 森嶋 厚行†† 青木 秀人†

† 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
E-mail: †{tomomi,aoki}@slis.tsukuba.ac.jp ††{mori,siena}@slis.tsukuba.ac.jp

あらまし 近年，群衆の知や力を利用して計算機だけでは困難な問題の解決を行うクラウドソーシングシステムが数多く登場している．クラウドソーシングシステムには，選択演算や結合演算など DB 操作に対応する人手の演算（Human-powered 演算）がしばしば含まれる．本論文ではそのような演算のひとつである Human-powered Join の効率化について議論する．これまで，Human-powered Join の効率化手法のひとつとして Join Pre-filter を利用するものが提案されているが，それが適用可能な範囲は限定されていた．本論文では，Join Pre-filter を利用する既存の効率化手法が前提とする仮定が成立しない場合であっても適用可能な効率化手法である Crowdsourced Join Pre-filter を提案し，その理論的・実験的評価を行う．特に，静的なデータベースだけでなく，タプルが継続的に追加される動的なデータベースを対象とした場合においても提案手法が効果的であることを示す．

キーワード クラウドソーシング

1. はじめに

計算機ネットワーク技術の発達に伴い，計算機だけではなく人も情報資源や処理装置とみなし，問題解決のためのシステムに含めることが可能になった．その結果，計算機には困難なデータ収集や処理を部分的に人手によって行う**クラウドソーシングシステム**が数多く出現している [1]．多くのクラウドソーシングシステムは，データの処理・管理などを目的とする**データ中心型**である．

データ中心型クラウドソーシングシステムには，選択演算や結合演算のような DB 操作に対応する人手の演算（**Human-powered 演算**）がしばしば含まれる [3]．Human-powered 演算のひとつである，人手による選択演算（Human-powered Selection）は，人手による主観的基準（美しい，楽しい等）に基づいて画像の選定を行うような演算である [2]．また，人手による結合演算（Human-powered Join）は，写真共有 SNS において顔写真に人名のタグ付けを行うような演算である．これら Human-powered 演算の処理コストは高いので，効率化に関する議論が活発に行われている [3] [4] [5] [6]．

本論文では，Human-powered 演算のうち，データ統合等の場面において重要な演算である Human-powered Join の処理効率化について議論する．Human-powered Join とは，人手によって結合条件の判定を行う結合演算であり，一般に計算機による結合条件の判定が難しい場合に利用される．Human-powered Join を含む問合せの別の例として，2 つの顔写真の集合から同一人物の顔写真のペアだけを得るような問合せが考えられる．計算機による被写体の同一性の判定が難しい場合，図 1 のようなタスクを人手によって行う必要がある．

Human-powered Join に対する効率化手法のひとつとして，

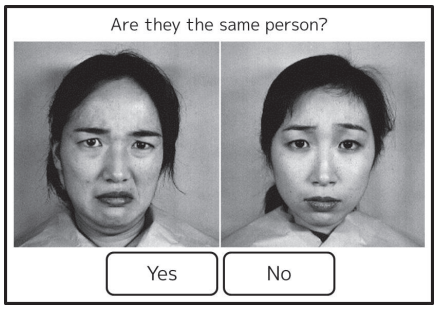

図 1 Human-powered Join のタスク：顔写真の被写体の同一性判定（画像は JAFFE Database [11] を利用）

論文 [3] では，**Join Pre-filter** を利用した効率化手法を提案している．この手法のポイントは，クラウドソーシングのプログラマが与える条件から生成される Join Pre-filter によって，結合条件を満たさないタプルのペアを事前に除去する前処理を行うという点である．例えば，同一人物の顔写真のペアだけを得る問合せに対しては，性別の異なる 2 人は同一人物ではないという Join Pre-filter が考えられる．プログラマがその Join Pre-filter を指定し，クラウドが各タプルに対して性別を入力するタスクを追加で行うことによって，追加タスクの増加分以上に，人手による結合条件判定タスクの数を削減することができる．

しかし，プログラマが常に適切な Join Pre-filter を考えるために必要なドメイン知識を持っているとは限らない．例えば，同一人物の顔写真のペアだけを得る問合せに対して，性別の情報を用いる Join Pre-filter は誰でも思いつくかもしれないが，筑波大学の部屋番号とゼミ名の結合を行うための適切な Join Pre-filter がなんであるかは誰でも思いつくとは限らない．

我々はプログラマが適切な Join Pre-filter を生成することができない場合にも適用可能な処理効率化手法 **Crowdsourced Join Pre-filter（CSJP）**を提案している [7] [8] [9]．CSJP では，適切な Join Pre-filter の生成をクラウドソーシングによって行う．本論文では，CSJP に関する理論的・実験的な評価結果を示す．CSJP では，タプルが継続的に追加されるような**動的なデータベース**（例：名簿に新しい人が追加される）においては，効率化のための追加タスクも継続的に増加する．本論文では，特にこのような動的なデータベースにおいても提案手法が有効であることを，理論的・実験的に示す．

本論文の貢献は次の通りである．

（1）クラウドソーシングによる処理効率化．多くのデータ中心型クラウドソーシングシステムは，データ収集や処理をクラウドソーシングするが，その処理効率化はプログラマが指定する必要がある．一方，提案手法は，データ収集や処理をクラウドソーシングするだけでなく，その処理効率化自体もクラウドソーシングする手法である．この意味において，提案手法は高階のクラウドソーシングであるといえる．

（2）提案手法の理論的な分析と実験による評価．提案手法を実現するタスクが適切であることを理論的に評価する．また，提案手法の有効性をクラウドソーシングプラットフォームである Crowd4U [10] 上で実際に実験を行った結果を示す．本論文では，特に動的なデータベースにおいても提案手法が有効であることを示す．

本論文の構成は次の通りである．2 節では，関連研究について述べる．3 節では，Human-powered Join と，Join Pre-filter を利用した効率化手法について述べる．4 節では，提案手法 Crowdsourced Join Pre-filter と，その理論的な分析結果について述べる．5 節では，特に，タプルが継続的に追加される動的なデータベースを対象とした場合における，提案手法の理論的な分析結果について述べる．6 節では，提案手法の実験的評価について述べる．7 節は，まとめと今後の課題である．

2. 関連研究

我々の知る限り，提案手法はデータ指向クラウドソーシングの処理効率化をクラウドソーシングする唯一の手法である．一方，データ指向クラウドソーシングの処理効率化をプログラマが行う手法はいくつか提案されている．例えば，タスクの大きさや，ユーザインタフェースを変更する手法が提案されている [3] [4] [5] [6] ．我々の提案手法とこれらの効率化手法は，組み合わせて利用することができる．

データ指向クラウドソーシングにおける処理効率化以外の重要な問題はデータ品質である．データ品質を向上させる手法も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採用されている多数決 [3] はデータ品質を向上させる手法のひとつである．ただし，データの品質は大数の法則に依存する．別のアプローチとしては，同調ゲーム [12] [13] がある．ただし，合理的なワーカが適切な値を入力することを前提としている．論文 [14] 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も，組み合わせて利用することができる．

```
SELECT R.img, S.img
FROM R JOIN S
ON samePerson(R.img, S.img)
```

図 2　Human-powered Join を含む問合せ

3. Human-powered Join とその処理効率化

本節では，Human-powered 演算のうち，データ統合等の場面において重要な演算である Human-powered Join と，その効率化のための Join Pre-filter を利用した効率化手法について述べる．

3.1 Human-powered Join

Human-powered Join とは，人手によって結合条件の判定を行う結合演算であり，一般に計算機による結合条件の判定が難しい場合に利用される．Human-powered Join を含む問合せの例として，2 つの顔写真の集合（それぞれ，`R(img)`，`S(img)` とする）から同一人物の顔写真のペアだけを得るような問合せが考えられる．計算機による被写体の同一性の判定が難しい場合，図 1 のようなタスクを人手によって行う必要がある．この問合せの結合条件を「`R.img` と `S.img` の被写体が同一である」とすると，論文 [4] で提案されている拡張 SQL を用いれば，この問合せを図 2 のように記述することができる．

結合条件判定タスク．Human-powered Join を含む問合せを処理するためには，結合条件判定タスクをワーカが行う必要がある．具体的には，ワーカは演算対象の 2 つのリレーションの直積の全てのタプルに対して「Yes」または「No」を入力する．例えば，図 2 の問合せを処理するためには，ワーカは `R` と `S` の直積に含まれる全ての顔写真のペアに対して，図 1 のようなフォームから「Yes」または「No」を入力する．問合せの処理結果は，ワーカによって「Yes」が入力されたタプルである．結合条件判定タスクの数は $|\mathtt{R} \times \mathtt{S}|$ である．

3.2 Join Pre-filter を利用した効率化手法

Human-powered Join に対する効率化手法として，Join Pre-filter を利用した効率化手法が提案されている．この手法のポイントは，Join Pre-filter を利用して，結合条件を満たさないタプルのペアを事前に除去する前処理を行うという点である．例えば，図 2 の問合せに対するタスクの数は，Join Pre-filter を生成するための次の 2 つのタスクを生成することで削減することができると考えられる．

（1） `R` と `S` の全ての顔写真に対する性別の入力タスク

（2） 同性同士の顔写真のペアの被写体同一性判定タスク

論文 [3] では，この Join Pre-filter を利用した問合せは図 3 のように記述する．図 2 との違いは，4 行目に `Possibly` 句を追加することで，Join Pre-filter の生成に性別の情報が必要であり，性別の情報を Join Pre-filter として利用することを指定していることである．具体的には，`Possibly` 句で，Join Pre-filter を生成するための属性 `gender` を指定している．

```
SELECT R.img, S.img
FROM R JOIN S
ON samePerson(R.img, S.img)
AND POSSIBLY gender(R.img) = gender(S.img)
```
図 3　Join Pre-filter を利用した図 2 の問合せの処理効率化

Join Pre-filter を利用した問合せの処理．図 3 のような，Join Pre-filter を利用した問合せの処理は次のように行われる．

Step 1. まず，プログラマが Join Pre-filter を生成するための属性（例：属性 gender）を考えて，図 3 のような問合せをクエリプロセッサに入力する．クエリプロセッサは，演算対象のリレーション R と S に属性 gender 追加したリレーション R'(img, gender) と S'(img, gender) を生成する．

Step 2. クエリプロセッサは，リレーション R' と S' の全ての顔写真に対して，属性 gender の属性値を入力するタスクを生成する．このタスクは，Join Pre-filter を利用する効率化のために，ワーカが追加で行うタスクである．

Step 3. ワーカが全ての顔写真に属性値を入力すると，クエリプロセッサは R' $\bowtie_{JP}$ S' を計算する．ここで，JP は，Join Pre-filter を利用した効率化を行うための結合条件 R'.gender=S'.gender である．

Step 4. クエリプロセッサは，R' $\bowtie_{JP}$ S' の全てのタプルに対して，結合条件判定タスクを生成する．ワーカが全ての結合条件判定タスクを完了すると，クエリプロセッサは Human-powered Join の処理結果を出力する．

Join Pre-filter による結合条件判定タスクの削減数．Join Pre-filter は，Human-powered Join のための結合条件判定タスクを数削する．仮に，R と S の顔写真の男女比が 1 : 1 である場合，結合条件判定タスク数は $|\mathtt{R}| \cdot |\mathtt{S}|$ から $\frac{|\mathtt{R}| \cdot |\mathtt{S}|}{2} + |\mathtt{R}| + |\mathtt{S}|$ に削減される．Join Pre-filter を利用した効率化の本質は，演算対象を**同値類**（例：[male], [female]）に分割することである．

3.3 Perfect Join Pre-filter モデル

Join Pre-filter は次のようにモデル化することができる [3]．N を問合せに追加される Possibly 句の数とする．$F = \{F_1, F_2, \ldots F_N\}$ を Join Pre-filter を生成するために追加する属性の集合とする．ここで，F_i は i 番目の Possibly 句で指定されている属性の属性値を含む集合である．例えば，i 番目で指定されている属性が gender であるとすると，$F_i = \{\mathtt{male}, \mathtt{female}\}$ である．L を演算対象の各タプルに付与される付与されるラベルの集合 $F_1 \times \ldots \times F_N$ とする．つまり，ラベル $l \in L$ は F の各属性の属性値を持つ N つ組である．演算対象のリレーション R と S の全てのタプルは，付与されるラベル l によって表されるいずれかの同値類に属する．例えば，全ての顔写真は，[male], [female] という 2 つのいずれかの同値類に属する．したがって，同じ同値類に属するタプルの組に対してのみ人手による結合条件判定を行えばよい．このモデルを，*perfect join pre-filter* モデルと呼ぶ．

4. Crowdsourced Join Pre-filter

Join Pre-filter を利用した効率化手法では，演算対象を適切な同値類に分割するための属性をプログラマが考える必要がある（3.2 節 Step 1 参照）．しかし，プログラマが必ずしも適切な属性を考えることができるとは限らない．例えば，同一人物の顔写真のペアだけを得る問合せに対して，同値類に分割するための属性として gender を考えることは誰でもできるかもしれないが，筑波大学の部屋番号と研究室の結合を行う問合せに対して，どのような属性が適切であるかは必ずしも自明ではない．

CSJP では，演算対象の適切な同値類への分割をクラウドソーシングによって行う．したがって，プログラマが適切な同値類への分割ができない場合も，タスクを行う各ワーカが演算対象に関する部分的なドメイン知識を持つ場合は Join Pre-filter を利用した効率化が適用可能である．提案手法のチャレンジは，演算対象の適切な同値類への分割という知的に高度な処理を，小さなタスクの集まりとして実現する点である．我々は，CSJP で 3 つのタイプのタスクを利用する．

本節では，次の問合せ Q1 を利用して 3 タイプのタスクを説明する．

問合せ Q1：筑波大学の春日エリアには複数の建物があり，各建物内の部屋は一意に識別可能な部屋番号（7D202 等）を持つ．また，部屋のうちの一部は，筑波大学図書館情報メディア研究科に所属する教員のゼミ（森嶋研等）によって利用されているが，どの部屋がどのゼミによって利用されているかは，ドメイン固有の知識であり，誰もが知っているわけではない．ここで，春日エリアの部屋番号の一覧を表すリレーションを R（スキーマは R(部屋番号)）とし，当該研究科のゼミ名の一覧を表すリレーションを S（スキーマは S(ゼミ名)）とする．R と S のタプル間を明示的に対応づける属性は存在しないため，これらの対応付けには人手による判断が必要である．このとき，部屋番号とゼミ名を結びつける Human-powered Join $R \bowtie_p S$（ただし，$p =$ 利用 (部屋番号, ゼミ名)）を**問合せ Q1** と呼ぶ．　□

4.1 CSJP のタスク設計

簡潔に言うと，CSJP で利用するタスクは，タプルに付与する属性値（例：male, female）の入力をワーカに求めるものと，属性値間の非両立性（例：male, female は非両立）の入力をワーカに求めるものである．

問題は，一般に Join Pre-filter を生成するために追加する属性（例：gender）がプログラマは事前に分からないことである．そこで，提案手法では，R(...) および S(...) に個別の属性を追加せず，prop と no_prop という属性を追加して，R'(..., prop, no_prop) および S'(..., prop, no_prop) を作成する．属性 prop は，各タプルに付与される属性値の和集合を表す属性である．一方，属性 no_prop は，各タプルに付与される非属性値の和集合を表す属性である．例えば，prop 属性の値として male を持ち，no_prop 属性の値として female を持つタプルは，male と female が同時には成立しないという非両立性から得られる．

タプルに付与する属性値とそれらの非両立性を記録するために，Name_table(name, incompatible_names)（図 4）を利用する．各タプル (n, in) において，n は属性値（例：male）で

Name_table

name	incompatible_names
春日エリア	{}
A棟	{B棟，C棟，D棟}
…	…
D棟	{A棟，B棟，C棟}

図 4　リレーション Name_table

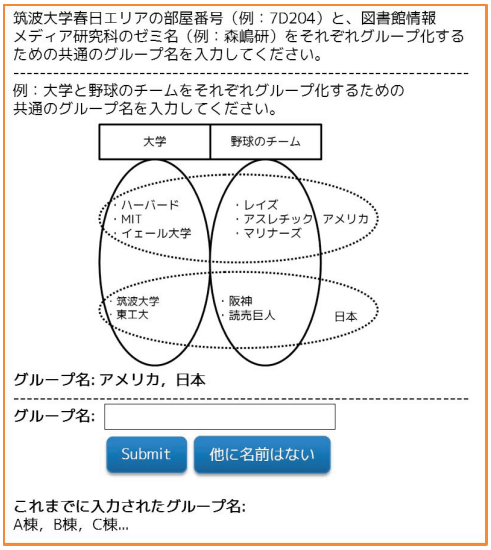

図 5　Type 1 のタスク

あり，*in* は，非両立な属性値の集合（例：{female}）である．また，Name_table リレーションに含まれる全ての属性値を記録するために Names リレーションを利用する．

具体的な設計．CSJP では，Join Pre-filter を利用した問合せの処理（3.2 節）の Step 1 を除去する．その代わりに，ワーカは次の Type 1, Type 2 のタスクを行う．さらに，Step 2 のタスクを修正し，ワーカは prop と no_prop 属性の値を入力する Type 3 のタスクを行う．

（1） Type 1: タプルに付与する属性値の入力
（2） Type 2: 属性値間の非両立性の入力
（3） Type 3: prop と no_prop 属性の値の入力

以下では，CSJP のための各タスクについて順に説明する．以下では，処理を行うワーカの集合をリレーション Crowd と表す．

Type 1. タプルに付与する属性値の入力．Type 1 のタスクでは，各 $c \in$ Crowd が，タプルに付与する属性値を図 5 のようなフォームを通じて入力する．フォームのページには，タスクに関する説明と，既に入力された属性値（図 5 では，ワーカへの分かりやすさを考慮して「グループ名」と表記），属性値を入力するための HTML フォームが表示されている．さらに，ワーカが，他に属性値がないことを宣言するためのボタンも配置されている．

入力された属性値はリレーション Name_table の name 属性に追加される（図 4）タスク数は，$|\text{Crowd}| \times k + M$ である．ここで，k は，各 $c \in$ Crowd の平均の入力回数，M は終了宣言のための特別なタスクの入力回数である．

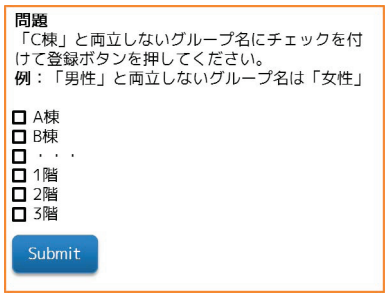

図 6　Type 2 のタスク

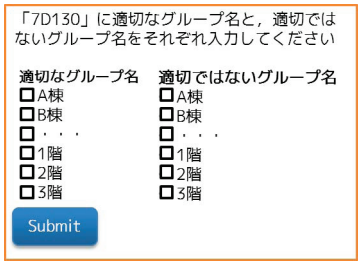

図 7　Type 3 のタスク

R'

roomno	prop	no_prop
7D130	{春日エリア}	{A棟，B棟}
7D131	{D棟}	{A棟，B棟}
7D140	{D棟}	{C棟}
…		
7D541	{}	{B棟，C棟}
7D542	{春日エリア}	{}

S'

fucultymember	prop	no_prop
三津石研	{}	{春日エリア}
森嶋研	{D棟}	{}
…		
品川研	{B棟}	{}
青木研	{}	{D棟}

図 8　CSJP で生成されるリレーションの例

Type 2. 非両立性入力タスク　Type 2 では各 $n \in$ Names に対してタスクが作成される（したがって，タスク数は |Names| である）．各タスクは，図 6 のようなフォームを通じて n と両立しない全ての $n' \in$ Names $- \{n\}$ にチェックを入力するものである．チェックされた全ての n' を Names リレーションの incompatible_names 属性の値として追加する．例えば，図 4 では「A 棟」と両立しない属性値の集合として「B 棟」「C 棟」「D 棟」が入力されている．

Type 3. 属性値付与タスク　R' と S' の各タプルに対して Type 3 のタスクが作成される．このタスクでは，ワーカが各タプルに適切な属性値を付与する．タスク数は |R| + |S| である．図 7 のようなフォームを通じて prop の属性値および no_prop の属性値を入力する．例えば，図 8 では R' の部屋番号が「7D130」である属性 prop の値として {春日エリア}，属性 no_prop の値として {A 棟，B 棟} が入力されている．

CSJP の手続き．CSJP の手続きは次の通りである．

（1） R(...) および S(...) に，prop と no_prop 属性を追加して，R'(..., prop, no_prop)，S'(..., prop, no_prop) を作成する．

（2） Type 1 のタスクをワーカに割り当て，ワーカが入力した属性値を Name_table リレーションの name 属性に追加する．他に属性値がないことを宣言するためのボタン M 回押されると，Type 2 のタスクをワーカに割り当て，ワーカが

incompatible_names の値を入力する．全ての Type 2 のタスクが完了すると，Type 3 のタスクがワーカに割り当てられ，ワーカが R' と S' リレーションに prop と no_prop の値を入力する．Type 3 のタスクの処理が行われる間，Type 2 のタスクの結果は no_prop の値のいくつかを自動的に計算するのに使われる．

（3） R' $\bowtie_{JP}$ S' を次の Join Pre-filter で計算する．JP は次の通りである．

$$\neg(\texttt{R'.prop} \cap \texttt{S'.no\_prop} \neq \phi \vee \texttt{R'.no\_prop} \cap \texttt{S'.prop} \neq \phi)$$

すなわち，R' の属性 prop の値が，S' の属性 no_prop に含まれる場合，タプルの結合を行わない．S' に対しても同様である．例えば，図 8 では R' の部屋番号が「7D130」である属性 prop の値として { 春日エリア }，S' のゼミ名が「三津石研」である属性 no_prop の値として { 春日エリア } が入力されている．したがって，これらのタプルの結合は行わない．

4.2 定　　理

本節では，CSJP が，Join Pre-filter を利用した効率化手法の一般化であることを示す次の 2 つの定理について述べる．(1) CSJP は，ワーカによった必要な情報が全て入力されれば，perfect join pre-filter の結果と同等である．(2) perfect join pre-filter で削減されないタプルは，CSJP でも削減されない．これらの定理は，Type 1 と Type 2 のタスクのみに関係する．なぜなら，Type 3 のタスクは通常の結合条件判定タスクと本質的に同じだからである．

[**Theorem** 1]　CSJP の手続きは perfect join pre-filter をシミュレート可能である．□

証明．CSJP の手続きによって生成される Join Pre-filter は，次の場合に perfect join pre-filter の手続きによって生成される Join Pre-filter と等価である．

- Type 1 のタスクによって入力された属性値の集合が $F_1 \cup \ldots \cup F_N$ であり, かつ
- Type 2 のタスクによって，各 F_i に属する f_j, $f_k \in F_i$ に関する全ての非両立性が入力されている．□

次の定理は，各タプルに付与されるラベル $l \in L$（3.3 節参照）によって表されるいかなる同値類も CSJP によってそれ以上に分割されないということをいう．すなわち，perfect join pre-filter で削減されないタプルは，CSJP でも削減されないということを保証する定理である．

[**Theorem** 2]　G を CSJP の手続きによって生成される同値類の集合とする．また，L を perfect join pre-filter において，演算対象の各タプルに付与されるラベルの集合とする．このとき，$l \in L$ によって表されるいかなる同値類も G の部分集合である．□

証明．Join Pre-filter を生成するために追加する属性を F_1 のみであると仮定しても一般性を失わない．なぜなら，perfect join pre-filter では，各 F_i が互いに独立していることを仮定しているからである（すなわち，$L = F_1 \times \ldots \times F_N$ である）．次に，$F_1 = \{A, B, C\}$ とする．同様に，この仮定も一般性を失わない．この時，perfect join pre-filter は，3 つの異なる同値類 $[A], [B], [C]$ を生成する．この時，CSJP の手続きで perfect join pre-filter と同じ同値類が生成できない次の 2 つのケースが存在する．

Case A:　CSJP の手続きが Type 2 のタスクより前で終了した場合．このケースでは，CSJP が F_1 を分割することはない．全ての $[A]$, $[B]$, $[C]$ が F_1 の部分集合であることは自明である．

Case B:　CSJP の手続きが F_1 に関する全ての非両立性が入力されるより前で終了した場合．C とそれ以外の属性値の間の非両立性が入力されなかった（すなわち '$A \neq B$' だけが入力された）と仮定する．この時，CSJP の手続きは，$[A] \cup [C]$ と $[B] \cup [C]$ という 2 つの同値類を生成する．具体的には，$G = \{[A] \cup [C], [B] \cup [C]\}$ である．したがって，$[A]$, $[B]$, $[C]$ は，G の部分集合である．一般的に，CSJP の手続きは,「perfect join pre-filter が生成する同値類は，G に含まれる全ての非両立でない同値類の部分集合」という条件を満たすように同値類の集合を生成する．したがって，perfect join pre-filter が生成する全ての同値類は，CSJP によって生成される同値類の部分集合である．□

4.3 総タスク数

Type 1 の総タスク数は，$|\texttt{Crowd}| \times k + M$ である．ここで，k は，各 $c \in$ Crowd の平均の入力回数，M は終了宣言のための特別なタスクの入力回数である．また，Type 2 のタスク数は $|\texttt{Names}|$，Type 3 のタスクにおいて $|\texttt{R}| + |\texttt{S}|$ である．したがって，総タスク数は次のようになる．

$$(|\texttt{Crowd}| \times k + M) + |\texttt{Names}| + (|\texttt{R}| + |\texttt{S}|).$$

5. 動的なデータベースにおける CSJP

CSJP では，タプルが継続的に追加されるような動的なデータベースにおいては，効率化のためのタスクも継続的に増加する．本節では，動的なデータベースにおける CSJP について説明する．

5.1 動的なデータベースの分類

動的なデータベースにおいて追加されるタプルは，既存のタプルと同じような性質を持っていることもあれば，全く異なる性質を持っていることもある．この観点から動的なデータベースを分類すると，表 1 に示す 4 つのケースに分類することができる．

表 1　動的なデータベースの分類

		既存の属性 F_i の属性値	
		増えない	増える
新たな属性 F_j の属性値	増えない	Case 1	Case 2
	増える	Case 3	Case 4

Case 1: 新しく追加されるタプルが既存のタプルと同じ性質の場合である．具体例を鉄道車両の写真とその路線名（例：山手線，京急線）の結合を行う問合せで考える．この例で，車両

の色で同値類に分割するための既存の属性 color の属性値として，red，blue，green が入力されているとする．Case 1 は，既存の路線の鉄道車両の写真が追加される場合である．この場合，既存のタプルに付与されている属性値以外は現れない．したがって，追加で Type 1 と Type 2 のタスクを行う必要はない．

一方，Case 2, Case 3, Case 4 は，新しく追加されたタプルが既存のタプルと異なる性質の場合である．

Case 2: 既存のタプルに付与されている属性値以外に，既存の属性の属性値が新たに現れる．同じ問合せの例では，新たに路線の鉄道車両の写真とその路線名が追加される場合である．この場合，新しい色の鉄道車両が追加されているとすると，追加で Type 1 と Type 2 のタスクを行う必要がある．例えば，既存の属性 color の属性値として，orange が必要になるとすると，追加で Type 1 と Type 2 のタスクを行う必要がある．

Case 3: 新たな属性の属性値が現れる．同じ問合せの例では，新たに路線バスの写真とその路線名が追加される場合である．この場合，例えば，新たな属性 vehicle の属性値として，bus と train が必要になるので，追加で Type 1 と Type 2 のタスクを行う必要がある．

Case 4: 既存の属性の属性値が新たに現れ，かつ，新たな属性の属性値が現れる．この場合も，追加で Type 1 と Type 2 のタスクを行う必要がある．

5.2 定　　理

本節では，CSJP が，動的なデータベースに対しても適用可能であることを示す定理について述べる．この定理は，動的なデータベースにおいても，perfect join pre-filter で削減されないタプルは，CSJP でも削減されないことをことを保証する定理である．

［**Theorem** 3］ G を CSJP の手続きによって生成される同値類の集合とする．その後，タプルが新たに追加された後に，CSJP の手続きによって生成される同値類の集合を G' とする．また，L を perfect join pre-filter において，演算対象の各タプルに付与されるラベルの集合とする．このとき，$l \in L$ によって表されるいかなる同値類も G' の部分集合である．　　□.

証明. $F_1 = \{A, B, C\}$ とする．また，$G = \{[A], [B], [C]\}$ とする．この時，動的なデータベースには，次の 4 つのケースが存在する（5.1 参照）.

Case 1: 既存の属性 F_1 の属性値以外が現れない場合．このケースでは，新たに追加されたタプルに対して，prop と no_prop 属性の値を入力する Type 3 のタスクのみを行うので，F_1 は変化しない．この時，$G' = \{[A], [B], [C]\}$ である．したがって，Theorem 2 より，$l \in L$ によって表されるいかなる同値類も G' の部分集合である．

Case 2: 既存の属性 F_1 の新たな属性値が現れる場合．新しい属性値 D が現れて，$F_1 = \{A, B, C, D\}$ となったとする．perfect join pre-filter は，$[A], [B], [C], [D]\}$ という同値類を生成する．この時，Theorem 2 より，$l \in L$ によって表されるいかなる同値類も G' の部分集合である．

Case 3: 新たな属性 F_2 の属性値が現れる場合．新たな属性の属性値 α, β が現れて，$F_1 = \{A, B, C\}, F_2 = \{\alpha, \beta\}$ となったとする．この時，perfect join pre-filter においては，F_1 と F_2 は互いに独立していることを仮定しているので，このケースは Case 1 と同様である．したがって，Theorem 2 より，$l \in L$ によって表されるいかなる同値類も G' の部分集合である．

Case 4: 既存の属性 F_1 の新たな属性値と，新たな属性 F_2 の属性値が現れる場合．$F_1 = \{A, B, C, D\}, F_2 = \{\alpha, \beta\}$ となったとする．この時，perfect join pre-filter においては，F_1 と F_2 は互いに独立していることを仮定しているので，このケースは Case 2 と同様である．したがって，Theorem 2 より，$l \in L$ によって表されるいかなる同値類も G' の部分集合である．　□

5.3 総タスク数

本節では，動的なデータベースにおける総タスク数について議論する．リレーション R と S が存在し，R に x タプルが追加されたとする．CSJP を適用しない場合には，結合条件判定タスクが $x \times |S|$ 増加する．

t_1, t_2, t_3 がそれぞれ，Type 1, Type 2, Type 3 で増加するタスク数とすると，CSJP を適用する場合の総タスク数は，$t_1 + t_2 + t_3$ 増加する．t_1, t_2, t_3 については 5.1 節で述べた Case 1 の場合と，それ以外のケースの場合の 2 通りで次のようにまとめることができる．

Case 1: この場合，既存のタプルに付与されている属性値以外は現れないので，追加で Type 1 と Type 2 のタスクを行う必要はない．したがって，$t_1 = 0, t_2 = 0, t_3 = x$ である．

Case 2, Case 3, Case 4: この場合，追加で Type 1 と Type 2 と Type 3 のタスクを行う必要がある．Type 1 で追加される属性値の数を y とすると，$t_1 = (|\texttt{Crowd}| \times k + M), t_2 = y, t_3 = x$ となる．

6. 実験と評価

本節では，CSJP を評価する実験とその結果について述べる．本実験の目的は，CSJP が perfect join pre-filter と比較してどの程度の効率化を実現できるか調査することである．また，動的なデータベースにおける CSJP の効率を検証するため，5.3 節で理論的に示した動的なデータベースにおける総タスクが，実際にどのような値になるかシミュレーションで調査する．

本実験では，学術利用を目的として構築中のクラウドソーシングプラットフォーム Crowd4U [10] を利用した．Crowd4U は，閉世界仮説を前提としない Datalog 風の言語 CyLog [10] を用いて，データ中心型クラウドソーシングを実現するためのプラットフォームである．Crowd4U では，登録されたワーカの属性情報がプログラマに見えており，プログラマはドメイン知識を持っていそうなワーカへタスクを依頼することが可能である．本実験ではこの機能を利用して，タスクの依頼を行った．

6.1 実 験 環 境

問合せ．本実験では，Human-powered Join を含み，タスクを行う際にドメイン知識が必要であると考えられる問合せ Q1 を Crowd4U 上で実行する．筑波大学に詳しくないプログラマが事前に Join Pre-filter のためのヒントを与えることは困難であると考えられる．

ワーカ．本実験では Crowd4U を通じて，問合せ Q1 に対する部分的な知識を持つと思われる学生 5 人を用意し，タスクを行うことを依頼した．

データ．実験に利用するデータとして次の 2 つのリレーションを利用した．各リレーションのタプル数は，R が 52，S が 62 である．

- R：筑波大学春日エリアの部屋番号一覧
- S：筑波大学図書館情報メディア研究科のゼミ名一覧

パラメータ．本実験における唯一の変数は，M である．これは，ワーカが，他に属性値がないことを宣言するためのボタンを Type 1 のタスクで押した回数である（参照 4.1 節）．本実験では，$M = 5$ としている．すなわち，全てのワーカがボタンを押す必要がある．

比較対象．CSJP の比較対象として，一定のドメイン知識を持つプログラマによる perfect join pre-filter を適用した場合を設定した．具体的には，次の 2 つの異なるレベルのドメイン知識を用いて行われる理想的な効率化を想定し，どれだけのタスクの削減効果があるか計算した．**（Perfect 1）** building_name 属性を R と S に追加する．これは，プログラマが建物名に関するドメイン知識を持っているということを想定している．**（Perfect 2）** building_name 属性と floor_number 属性を R と S に追加する．これは，プログラマが建物名と階数に関するドメイン知識を持っていることを想定している．

CSJP によって入力される属性値は必ずしも全て正しい値とは限らない．タスク数の削減効果を計算するために，次の 2 つのケースを設定した．**（CSJP 1）** building_name 属性と floor_number 属性に対応するような属性値（例：7D 棟，1 階など）を CSJP の計算に利用する．**（CSJP 2）** 入力された全ての属性値を CSJP の計算に利用する．

6.2 実験結果と考察

静的なデータベースに対する CSJP の効率．図 9 は，Join Pre-filter を適用しない場合の総タスク数（JP なし），一定のドメイン知識を持つプログラマによる perfect join pre-filter である Perfect 1 と Perfect 2 を利用した場合の総タスク数，CSJP である CSJP 1 と CSJP 2 を利用した場合の総タスク数を比較したグラフである．CSJP 1 では総タスク数が 1617 であり，十分に効率的な Perfect 1 と Perfect 2 の中間のタスク数である．また，CSJP 2 では総タスク数が 734 であり，Perfect 2 より多くのタスクが削減されている．これは，building_name 属性と floor_number 属性以外にも，Join Pre-filter に利用可能な属性値が数多く入力されたからである．これらの結果は，各 $c \in Crowd$ が問合せ Q1 に対する部分的な知識を持つときには，CSJP による効率化が有効である可能性を示唆している．

CSJP による各タスクの実行結果の詳細は次の通りであった．まず，Type1 のタスクでは，29 の分割ラベルが付与された．次に，Type2 のタスクでは，分割ラベルの組合せ ${}_{29}C_2$ のうち，40 の非両立ラベルのペアが発見された．また，Type3 の属性値付与タスクでは，部屋番号リレーション R（タプル数 52）に関して，全てタプルに対して属性 prop の値が付与され，51 のタプルに対して属性 no_prop の値が付与された．ゼミ名リレーション S（タプル数 62）に関しては，全てのタプルに対して属性 prop および，属性 no_prop の値が付与された．

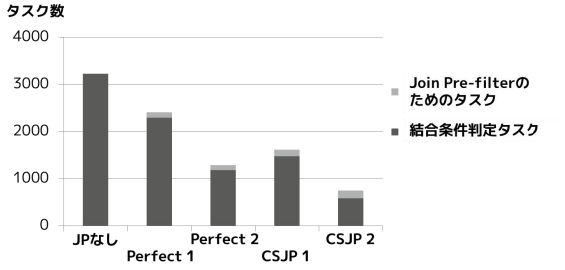

図 9　CSJP によるタスク数の削減効果

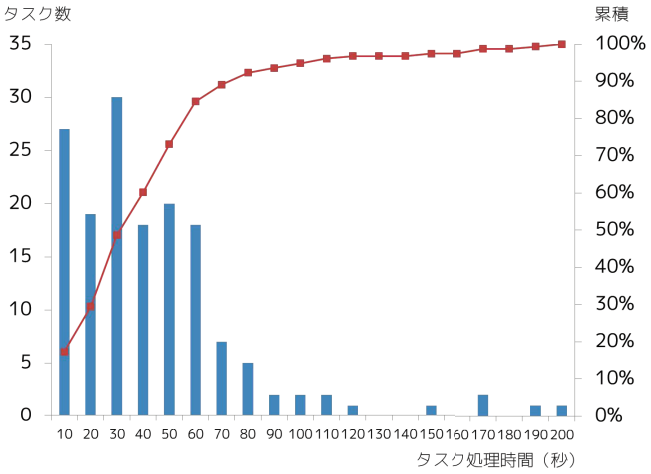

図 10　タスク処理時間分布

タスク処理時間分布と総タスク処理時間．本実験では，ワーカが各タスクを選択してから終了ボタンを押すまでの時間をタスクの処理時間として計測した．タスク処理時間分布図を図 10 に示す．これらのタスクの処理時間の平均は約 40 秒であり，約 84%のタスクが 60 秒以下で完了した．この結果から，Crowdsourced Join Pre-filter で利用するタスクは，ワーカにとって短時間で比較的容易に実行可能なタスクであったと考えられる．このことは，本論文で提案するタスク設計が適切であることを示唆している．

総タスク処理時間は，1 時間 42 分 26 秒であった．

6.3 動的なデータベースのシミュレーション

動的なデータベースにおける CSJP の効率を検証するため，5.3 節で理論的に示した動的なデータベースにおける総タスクが，実際にどのような値になるかシミュレーションで調査する．部屋番号リレーション R にいくつかのタプルが追加されることをシミュレーションするために，次のようなシナリオを設定する．なお，このシナリオは，5.1 節で述べた Case 2 に対応する．

シミュレーションのシナリオ．筑波大学の春日エリアには講義棟・研究棟・ユニオン棟という 3 つの建物がある．このうち，ユニオン棟という建物が新しく建造され，いくつかの部屋が新たに増えるという状況を仮定する．具体的には，部屋番号リレーション R が 52 のうち，はじめから存在するタプル数が 40（約 77%），新たに追加されるタプル数が 12（約 33%）であること

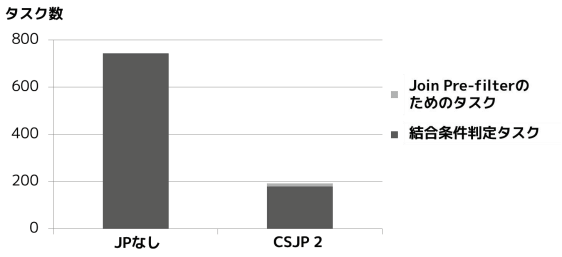

図 11　動的なデータベースにおける CSJP によるタスク数の削減効果

を仮定する．ゼミ名リレーション S にはタプルは追加されない．ユニオン棟が建造されるまでは，Join Pre-filter を生成するために追加する属性 building_name の属性値は { 講義棟, 研究棟 } であり，属性値 floor_number の属性値は {1F, 2F, 3F, 4F, 5F} であるとする．ユニオン棟は，4 階建ての建物であるので，タプルが追加されることによって追加される属性値は「ユニオン棟」のみを仮定する．

総タスク数のシミュレーション．このようなシナリオで，動的なデータベースにおける総タスク数を計算する．Type 1 のタスクでは，新たにユニオン棟という建物名が属性値として入力されるので，1 タスク増える．Type 2 のタスクでは，ユニオン棟とそれ以外の既存の属性値との非両立性が入力されるので，1 タスク増える．ここでは,「講義棟 ╪ ユニオン棟」「研究棟 ╪ ユニオン棟」の非両立性が入力されたとする．Type 3 のタスクでは，新たに追加される 12 のタプルに属性値を付与するために，12 タスク増える．したがって，Join Pre-filter のためのタスク数は 14 である

次に，削減されるタスク数を計算する．新たなタプルが 12 追加されたとき，Join Pre-filter を適用しない場合のタスク数は 744 である．一方，Type 1, Type 2, Type 3 を 14 タスクを行うことによって，タスク数を 179 まで削減できる．このシナリオにおいては，動的なデータベースにおいても提案手法が有効であることが分かった．

7. まとめと今後の課題

本論文では，Human-powered Join に対する効率化手法である Crowdsourced Join Pre-filter に対する理論的な分析および，実験による評価を行った．具体的には，Crowdsourced Join Pre-filter のためのタスク設計が適切であることを理論的に示し，実環境での効果を実験によって示した．また，静的なデータベースだけでなく，タプルが継続的に追加される動的なデータベースに対しても提案手法が有効であることを示した．今後の課題としては，他の効率化手法やデータ品質管理手法等と組み合わせた場合の分析を行うことがあげられる．

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「情報環境と人」による．

文　　献

[1] Doan AnHai, Ramakrishnan Raghu, Halevy Alon Y.. “Crowdsourcing systems on the World-Wide Web. Commun.ACM. vol. 54, no. 4, pp. 86-96, 2011
[2] gwap.com. “Matchin”. http://www.gwap.com/gwap/gamesPreview/matchin/
[3] Marcus Adam, Wu Eugene, Karger David R., Madden Samuel, Miller Robert C.. “Human-powered Sorts and Joins”. PVLDB 2011, vol. 5, no. 1, pp. 13-24, 2011.
[4] Marcus Adam, Wu Eugene, Madden Samuel, Miller Robert C.. “Crowdsourced Databases: Query Processing with People”. CIDR 2011, pp. 211-214, 2011.
[5] Franklin Michael J., Kossmann Donald, Kraska Tim, Ramesh Sukriti, Xin Reynold.. “CrowdDB: answering queries with crowdsourcing”. SIGMOD 2011, pp. 61-72, 2011.
[6] Parameswaran Aditya G., Polyzotis Neoklis.. “Answering Queries using Humans, Algorithms and Databases”. CIDR 2011, pp. 160-166, 2011.
[7] 三津石智巳，森嶋厚行，品川徳秀，青木秀人．“クラウドソーシングによる Human-powered Join の効率化”．DEIM 2012, 2012.
[8] 三津石智巳，森嶋厚行，品川徳秀，青木秀人．“Human-powered Join 処理に対するクラウドソーシングに効率化手法の評価”．SoC 2012, pp. 103-108，2012.
[9] Tomomi Mitsuishi, Atsuyuki Morishima, Norihide Shinagawa, Hideto Aoki. “Efficient Evaluation of Human-powered Joins with Crowdsourced Join Pre-filters”. ICUIMC 2013, 2013 .
[10] Atsuyuki Morishima, Norihide Shinagawa, Tomomi Mitsuishi, Hideto Aoki, Shun Fukusumi. “CyLog/Crowd4U: A Declarative Platform for Complex Data-centric Crowdsourcing”. VLDB 2012, August 2012 (demo, to appear).
[11] Michael J. Lyons, Akamatsu Shigeru, Kamachi Miyuki, Gyoba Jiro. “Coding Facial Expressions with Gabor Wavelets”. Third IEEE International Conference on Automatic Face and Gesture Recognition. pp. 200-205, 1998.
[12] Atsuyuki Morishima, Norihide Shinagawa, Shoji Mochizuki. “The Power of Integrated Abstraction for Data-Centric Human/Machine Computations”. VLDS 2011: 7-10
[13] gwap.com. “ESP Game”. http://www.gwap.com/gwap/gamesPreview/espgame/
[14] Shaili Jain, David C. Parkes: A Game-Theoretic Analysis of Games with a Purpose. WINE 2008: 342-350