時間経過を考慮したワークフローにおけるタスク割当て手法

山口　大河†　　鈴木　伸崇††

† 筑波大学人間総合科学学術院人間総合科学研究科　〒 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　〒 305-8550 茨城県つくば市春日 1-2
E-mail: †s2121659@s.tsukuba.ac.jp, ††nsuzuki@slis.tsukuba.ac.jp

あらまし　多様なワーカを取り入れるためにはワーカ側の視点・リクエスタ側の視点双方が重要である．本研究ではワークフローへの割当てを扱うが，ワークフローには時間的制約が存在する．1 つはサブタスク間の順序関係であり，また 1 つはサブタスクの処理に時間が必要な点である．これらの制約を明示的に扱うことは，ワーカがタスクに取り組むべき時間を局所化 することにつながり，ワーカの効率なタスク受注の実現が期待できる．以上を踏まえ，本研究では，多くのワーカを差別なく組み込み，それによる効率の低下を防ぎ．かつワークフローが含む時間制約を明示的に取り扱い，ワーカがタスクを行うべき期間を局所化する割り当て手法の構築を目指す．そのために，本研究ではワーカにスケジュールの概念を取り入れ，それに基づいた割当てを行う．その手法として，既存手法を拡張した手法および，各期間の割当ワーカ数を最大化する手法を提案する．これら 2 手法を比較する実験を行った．その結果，最大流問題を利用した手法が多くの場合において優位であ ること，および，時間を明示的に取り扱うことにより，割当ての効率性という面において優位な結果となることを示した．また，それらの結果を併せ，より実践的な割当てに応用できる可能性を示した．

キーワード　クラウドソーシング，タスク割当て，ボランティア，スケジューリング

1　はじめに

クラウドソーシングにはプロジェクト型，コンペティション型，マイクロタスク型など複数の形式が存在するが，中でもマイクロタスク型は通常 1 回当たりの作業量が数分程度で終了する分量（シンプルタスク）に設定され，また必要とされる能力も少ないという特徴を持っており，この特徴により，従来は様々な要因（労働に割ける時間が少ない，能力が低いなど）から労働が困難であった層を労働人口として取り入れるための手段としても有用であると考えられている [13] が，一方で課題も多い．

現在は Amazon Mechanical Turk やクラウドワークスに代表されるようにマイクロタスク型のクラウドソーシングではシンプルタスクが主流である．しかし，マイクロタスク型クラウドソーシングの今後の活用のためにはコンプレックスタスクへの対応が望ましい．コンプレックスタスクとはシンプルタスクに比べ複雑な課題を解決するようなタスク（ホームページを作成するなど）のことを指す．コンプレックスタスクを処理するための一般的なアプローチとして，主に 2 通りの手法が提案されている．1 つはコンプレックスタスクを複数のより簡単なサブタスク（シンプルタスク）に分割し，各サブタスクに対してワーカを割り当てるという手法 [5] [6] である．2 つめは複数のワーカでタスクを処理するためのグループを構築し，そのグループに対してコンプレックスタスクを割り当てるという手法 [14] [15] である．また，これら 2 通りの手法はいずれも 1 つの処理するべきタスク（コンプレックスタスク）に対してとりうるアプローチを挙げたものであるが，複数のタスクを縦断的に扱い，類似するタスクを括りそれを同一のワーカに割当てるバッチ割当ての手法 [7] [16] なども提案されている．本論文では，これらの手法のうちサブタスクへコンプレックスタスクを分割する手法の中でも，ワークフロー形式にサブタスクを分割する場合のタスク割当て手法を取り扱う．

ワーカへのタスク割当て手法の多くでは生産性やスループットなどタスクを依頼する側（リクエスタ）の事情に配慮して割当て手法を評価してきた．一方で，前述の通りマイクロタスク型のクラウドソーシングは，より幅広い層から労働人口を確保するという点からも期待されており，ワーカへの配慮を行うことも重要である [2]．加えて，クラウドソーシングはボランティア活動においての活用も活発になってきており [4]，そのようなボランティアベースの場面ではより顕著にワーカへの配慮が求められる．このようなワーカへの配慮の観点から提案された指標の 1 つがインクルージョン性である．インクルージョン性とは集まったワーカに対してどれだけのワーカが実際にタスクに関わることができているかどうかを測る指標である．ワークフローにおけるタスク割当てを扱った先行研究として生産性やスループットとインクルージョン性を両立することを目指した手法 [11] が提案されている．本論文もこの流れを汲み，ボランティアベースでの活用を想定し，効率性とインクルージョン性の両面から評価を試みる．

本論文ではワークフローで表現されるサブタスクの集合に対してワーカを割当てる課題を取り扱う．ワークフローはサブタスク処理の順番と経路を定義するものであり，そこには一定の時間的制約（タスクの先後，タスク処理に係る時間）と構造的制約（タスク処理の経路）が存在する．先行研究の多くでは構造的制約のみに着目し，時間的制約が明示的に示されることは

少なかった．一方で，時間的制約について言及した先行研究も存在し，サブタスクの処理期間が考慮されておらず全体としてのタスクが時に完了しない可能性があるという指摘 [7] [16] や，有向グラフ形式で表現されるサブタスクには順序関係が存在するため，これを考慮するべきであるという指摘 [8] という形で示されている．また加えて，[16] では，多くのワーカが同時に受注するタスクを１つに絞る傾向があり，なおかつ，タスクの期限は通常処理に必要な期間に対してやや冗長に設定される傾向があるため，ワーカの労働が効率的ではないことをも指摘した．以上のような指摘を踏まえ，本論文ではワークフローの時間的制約を明示的に取り扱うことでこれらの課題に対処することを試みる．

本論文の提案する手法の目的は以下の２つに設定されている．１つに多くのワーカを差別なく組み込み（インクルージョン性に配慮する），それによる効率の低下を防ぐこと．２つ目に，ワークフローが含む時間的制約を明示的に取り扱い，ワーカがタスクを行うべき期間を局所化することの２点である．本論文で取り扱う課題の一例として，期間Ｐにおいて，ワークフロー（Ａ⇒Ｂ⇒Ｃ）を実行する必要がある場合においてワーカ {a, b, c} にタスクを割当てるという課題を想定する．タスクＡにワーカ a を，タスクＢにワーカ b を，タスクＣにワーカ c を割り当てるといった割当てを行うのが時間的制約の存在しない従来手法である．しかし，この場合現実にはワーカ a がタスクＡを終えない限り，ワーカ b はタスクＢに取り掛かることはできず（ワーカ c も同様），期間Ｐのうち一部のみしかタスクの作業に使用されず，時間的なロスが発生すると考えられる．そこで本論文では期間Ｐを，{ 期間１，期間２，期間３ } の集合に分割して考え，[タスクＡ，期間１] にワーカ a を，[タスクＢ，期間２] にワーカ b を，[タスクＣ，期間３] にワーカ c を割り当てるといった形式で時間的制約を明示化し，タスクの割当てを考える．これによりワーカはタスクを行うべき期間が局所化されより効率的に時間を使うことが可能になる．

本論文では，以上を踏まえ，２つの手法（既存手法拡張型，最大流問題利用型）に対して，ワーカの能力数およびワークフローを変化させた複数の実験条件を準備し，インクルージョン性を評価する $inclusion$，同じ成果量に対して必要なワーカ数を評価する $influx$，完成しなかった成果の数を評価する $loss$ の３つの観点について評価実験を行った．評価実験は２手法の比較評価実験と，時間の明示化を導入した割当ての特性を評価する特性評価実験を行った．なお，後者では前述の２つの要素に加え，期間の分割数を実験条件に加えている．比較評価実験では，最大流問題利用型の手法が多くの場合優位であるということ，また，手法の優位性がワークフローの違いによる影響を受けることを示した．特性評価実験では，時間の明示化を導入した割当て手法が，主に $influx$ や $loss$ といった効率性に関係する指標に影響を及ぼすことを示した．

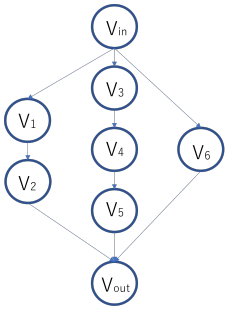

図 1: ワークフロー例

2 諸 定 義

2.1 タ ス ク

本研究では「タスク」というワードに対し，複数の定義が存在するため，それらを使い分けるために以下のように定義する．

１つ目はメインタスクである．本研究における「メインタスク」とは，コンプレックスタスクそのものの事を指す．例えば，T:「授業Ｌの録画動画に対して日本語の音声に英語字幕をつける」というタスクが存在した場合，T をメインタスクとする．

２つ目はサブタスクである．サブタスクはメインタスクに対してそれをより簡単なタスクの集合（本研究ではワークフロー）に構成し直した場合の，各タスク（ワークフローにおけるノード V_i）の事を指す．上述の例に則ると，例えばメインタスク T を３つのタスク，A:「動画を３分程度の動画に分割する」，B:「分割された動画に対して，日本語を文字として書き起こす」，C:「書き起こされた日本語を英語として書き起こす」に分割した場合，A，B，C のような各タスクをサブタスクと定義する．

３つ目はタスクインスタンスである．タスクインスタンスはサブタスクに含まれる１単位（１人が１単位時間中に処理するタスク）の事を指す．例に則ると，サブタスク B:「分割された動画に対して，日本語を文字として書き起こす」に対して，B_3「動画（06:00〜09:00）に対して，日本語を文字として書き起こす」のようなタスク B_3 をタスクインスタンスと定義する．

また，本論文では「パス」というワードを使用する．パスとは，メインタスクを処理可能なサブタスクの一連の処理経路の事を指す．例えば，P_1：「サブタスク A，サブタスク B，サブタスク C の順に処理」というような場合の P_1 を指す．これを，グラフ（図 1）を例にとると，次のように表記する．

$$P_1 = (V_{in}, V_1, V_2, V_{out})$$

また，この場合 $P_2 = (V_{in}, V_3, V_4, V_5, V_{out})$ や $P_3 = (V_{in}, V_6, V_{out})$ などの経路もパスである．

2.2 モ デ ル

本研究では以下のようなモデル（図 2）のもとワーカの割当てを考える．まず，各ワーカはメインタスクに対して申込を行い，各ワーカをメインタスクで受け入れるかどうかを判断する．この際，基本的には全てのワーカを受け入れることを想定しているが，ワーカの有する能力（A_w）とタスクに必要とされる能

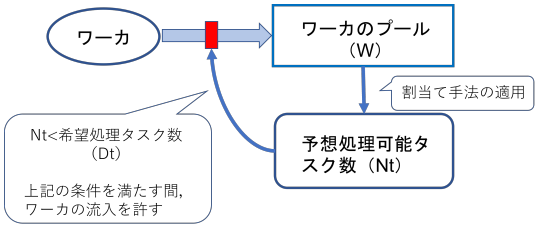

図 2: 本研究の割当てモデル

力（A_t）とを比較し，いずれのタスクに対しても必要な能力を有さないと判断される場合にのみ受け入れを拒否する．そうして受け入れられたワーカでワーカのプール（集合 W）を作成し，形成された W に対して割当て手法を適用，想定される処理可能タスクインスタンス数 N_t を算出する．これをリクエスタ側が提示する希望処理タスク（タスクインスタンス）数 D_t と比較し，$N_t \geqq D_t$ となった時点でワーカの流入を停止し，募集を打ち切る．

この際，リクエスタおよび，ワーカは以下の情報を与えるものとする．

- リクエスタ
  - 希望処理タスク数（D_t）：必要な処理済みタスクインスタンスの数を設定する
  - 希望処理期間（N）：単位期間（標準的な 1 タスクインスタンスを処理するために要する期間）に対してどの程度の期間内に処理が完了することを求めるか，期間の長さを設定する
  - ワークフロー：サブタスクに分解済みのワークフローを提供する
  - サブタスクに要する能力（A_t）：各サブタスクを処理する為に必要な能力を設定する
- ワーカ
  - 能力（A_w）：自身が有する能力（サブタスクの処理に必要な能力）を申告する
  - スケジュール（S_w）：メインタスクの処理期間内におけるタスク割当ての希望を単位期間ごとに提出する

なお，割当て手法の適用については，第 3 章において詳細に記述する．

2.3 インクルージョン性

インクルージョン性とは集まったワーカに対して，そのうちどの程度のワーカに実際にタスクが割り当てられたかどうかを評価する指標である．本論文では，これを以下のように定義する．

$$inclusion = \frac{\sum_{w \in W} allocation(w)}{|W|}$$

ここで W は集まった全てのワーカの集合，$allocation(w)$ はワーカ w に対して，そのワーカに対してタスクが割当てられていれば 1 をそうでなければ 0 を返す関数である．本研究においては，ワーカはスケジュール上の複数期間に対してタスク割当ての希望を提出することを想定しており，またそれら複数に対して実際にタスクが割当てられることをも想定している．$allocation(w)$ はこのようにワーカ側が希望する期間が複数存在する場合，いずれか 1 つの期間に対してタスクが実際に割当てられている状態をタスクが割当てられているとみなす．

3 提 案 手 法

本章では，本論文の提案手法について解説する．まず，本論文で取り扱う時間の明示化について例を交えながら概説し，その後，手法に共通する処理，既存手法を拡張した手法について述べ，最後に最大流問題を活用した手法について解説する．

3.1 時間の明示化

本研究で取り扱うモデルについては第 2 章で述べた．ここでは，第 2 章で取り上げたモデル（図 2）における割当て手法の適用について詳細に記述する．例として以下のような割当てを考える．

- $D_t = 3$
- $N = 3$
- $Workflow = (V, E)$
- $V = \{V_{in}, V_1, V_2, V_3, V_4, V_5, V_6, V_{out}\}$
- $E = \{(V_{in}, V_1), (V_{in}, V_3), (V_{in}, V_6), (V_1, V_2), (V_2, V_{out}), (V_3, V_4), (V_4, V_5), (V_5, V_{out}), (V_6, V_{out})\}$
- $Ability = \{a_1, a_2, a_3\}$
- $A_t = \{(V_1, [a_1, a_2]), (V_2, [a_3]), (V_3, [a_1]), (V_4, [a_2]), (V_5, [a_3]), (V_6, [a_1, a_2, a_3])\}$
- $W = \{w_1, w_2, w_3, w_4, w_5\}$
- $A_w = \{(w_1, [a_1]), (w_2, [a_2]), (w_3, [a_3]), (w_4, [a_1, a_2]), (w_5, [a_1, a_2, a_3])\}$
- $S_w = \{(w_1, [1, 0, 1]), (w_2, [0, 1, 0]), (w_3, [0, 1, 1]), (w_4, [1, 0, 0]), (w_5, [0, 0, 1])\}$

ここで各要素について順に改めて解説する．D_t はリクエスタ側が提示した必要なタスクインスタンス数を示すものである．N は割当てを考える期間の長さを示し，上記の例では 3 つの単位期間でタスクを終了することを示す．$Workflow$，V，E はいずれもワークフローを定義するものである．これについては第 2 章で提示したワークフローの例（図 1）を用いている．$Ability$ は使用するワークフローにおいて考慮するべき能力として提示される能力の集合である．A_t は各サブタスクにおいて必要とされる能力をサブタスクごとに定義したものであり，例えば $(V_1, [a_1, a_2])$ はサブタスク V_1 に対して必要な能力は a_1 と a_2 であることを示している．また，V_{in} および V_{out} は始点と終端のサブタスクを定義するものであり，実際に処理するべきタスクインスタンスは存在せず，これらのサブタスクに能力は設定されない．W はプールされたワーカの集合である．A_w は各ワーカの有する能力を定義するものであり，例中の $(w_4, [a_1, a_2])$ はワーカ w_4 が能力 a_1 と a_2 を有していることを示す．最後に，S_w は各ワーカの各期間におけるメインタスクへの参加の可否（希望）を示すものである．上記の例では $N = 3$ となっていることから，各ワーカが提示するスケジュール長は 3 となる．例えば，$(w_1, [1, 0, 1])$ はワーカ w_1 が提示したスケジュールが $[1, 0, 1]$ であることを示し，$[1, 0, 1]$ は 3 つの

表 1: ワーカのスケジュール例

worker	期間 1	期間 2	期間 3
w_1	○	−	○
w_2	−	○	−
w_3	−	○	○
w_4	○	−	−
w_5	−	−	○

表 2: 割当て例

サブタスク名	期間 1	期間 2	期間 3
V_1	w_4	−	−
V_2	−	w_3	−
V_3	w_1	−	−
V_4	−	w_2	−
V_5	−	−	w_3
V_6	−	−	w_5

期間において，１番目と３番目の期間にメインタスクへの参加が可能（割当て可能）であることを示している．また，このスケジュールを表に示すと以下のようになる（表 1）．本研究では，割当て上の時間的制約を以下のように設定する．

（1） 各単位期間に１人のワーカは１種類・１つのタスクインスタンスを実行する
（2） 先行タスクが存在する場合，それより前の期間において先行タスクのインスタンスが実行されていることを必要とする
（3） 各ワーカへはスケジュール上参加可能としている期間のみタスク割当て可能である

以上を踏まえ，このような例に対して適当な割当ての一例は上表（表 2）のようになる．なお，ここで行：V_1，列：期間１に記された w_4 はサブタスク V_1 のタスクインスタンスを期間１においてワーカ w_4 に割当てることを表す．以下，割当てが適当であることを確認する．まず，期間１において割当て可能なワーカは表１より $w1$ および w_4 であり，各ワーカは V_3，V_1 の各１種類のサブタスクに対してのみ割当てられている．よって，期間１において時間的制約１と３を満たしている．これは期間２，期間３についても同様である．また，時間的制約２について確認すると，V_2 はサブタスク V_1 を先行タスクにもつが，期間２における w_3 への割当てについて，期間１において既にサブタスク V_1 のインスタンスを実行済みである．同様に，V_4，V_5 についても先行タスクが実行済みであることが確認できる．次に能力について確認する．割当ての際には，ワーカがサブタスクに必要とされる能力全てを有している必要がある．この点について，V_6 を例にとると，V_6 は a_1，a_2，a_3 全ての能力を必要とするサブタスクであるが，期間３において割当てられたワーカ w_5 は全ての能力を有しており，条件を満たしていることが確認できる．これもまた全てのサブタスクについて同様に確認できる．

また，第２章で提示したモデル（図 1）に拠ると，割当てをしたのち予想処理可能タスク数 N_t を算出する必要がある．これは V_{in} から V_{out} までの完成したパスの数と同義である．例のような割当ての場合，$(V_{in}, V_1, V_2, V_{out})$，$(V_{in}, V_3, V_4, V_5, V_{out})$，$(V_{in}, V_6, V_{out})$ の３つのパスが完成していることを確認できる．よって，$N_t = 3$ である．最後に，これを $D_t = 3$ と比較し，$N_t \geqq D_t$ が成立するため，このような割当てが考えられる場合にはワーカの募集を打ち切ることになる．

以上が，時間の明示化とそれを考慮した割当て，およびそれを利用したモデル中での活用の例である．

3.2 ワーカのグループ化

前節では本研究で取り扱うタスク割当ての適用について述べた．本節以降では，実際に本研究が提案する割当て手法について解説する．本論文では２種類の割当て手法を提案する．１つは既存手法拡張型，もう一方は最大流問題利用型である．それぞれ第３．３節，第３．４節で詳細な手法については記述する．本論文では，両手法に共通で行っている処理について解説する．

本研究では，割当ての簡略化のため，ワーカのグループ化を行っている．ワーカのグループ化の手順は以下の通りである．

以下，ワーカのグループ化について概説する．

１～４行目では，各値の初期化を行っている．G_w はワーカのグループの集合，G_m は各グループに含まれるワーカの集合，G_s は各グループでグループ内全体を統合したスケジュール，G_t は各グループが実行可能なタスクの集合を登録する変数である．

５～１３行目では，各ワーカについて，実行可能なタスクを抽出し，ワーカをグループ化している．６行目では各ワーカが実行可能なタスクの集合を保存する *tasks* を初期化する．７行目では各サブタスクについてワーカが有する能力とサブタスクに必要とされる能力とを比較し，ワーカが必要な能力を有する場合に *tasks* に該当サブタスクを追加する．１２行目では，各ワーカの *tasks* を参照し，現状登録されているワーカのグループに同様のグループが存在するかどうかを確認し，存在しない場合，新規のグループを作成し，存在する場合，該当グループのワーカ集合に追加する．

１４～２０行目では，グループごとにグループ内のワーカが提出したスケジュールを統合している．１５行目では各グループの統合後スケジュールを長さ N，初期値 0 で初期化している．１６行目から１８行目ではグループに含まれる各ワーカのスケジュールを参照し，ワーカがメインタスクに参加可能と提示している期間の値を 1 加算する．１９行目は作成された各グループの統合スケジュールを保存している．

このような処理を行うことで，ワーカは実行可能なタスクを特徴量としてグループ化されることになる．上記のアルゴリズム中で使用している関数は以下のような機能をもつ．なお，G_w にはワーカのグループの集合が，G_m には各グループに含まれるワーカの集合が，G_s には各グループでグループ内全体を統合したスケジュールが，G_t には各グループが実行可能なタスクの集合が登録されている．

- *update-group*$(G_w, G_m, G_t, tasks, w_i)$

– G_t 中の全ての $value$（集合）を参照し，$tasks$ と一致する集合が存在するかどうかを確認する

– 存在した場合，該当 $value$ の key を参照し，当該 $key = name$ に結びついた $G_m(name)$ に w_i を追加する

– 存在しなかった場合，G_w にユニークなグループ名 $name$ を生成し，$G_m(name)$ に w_i を追加する

- $update\text{-}schedule(schedule, w_j, S_w)$

– $S_w(w_j)$ を参照し，w_j のスケジュールを取り出す

– $schedule$ に取り出したワーカ w_j のスケジュールを期間ごとに足し合わせる

Algorithm 1 Grouping Algorithm

Input: N,W,$Workflow$,A_t,A_w,S_w
Output: G_w,G_m,G_s,G_t

Initialization :

$G_w = \emptyset$
$G_m = \emptyset$
$G_s = \emptyset$
$G_t = \emptyset$
for $i = 1$ to $|W|$ **do**
  $tasks = \emptyset$
  for $j = 1$ to $|V|$ **do**
    if $A_w(w_i) \supseteq A_t(t_j)$ **then**
      $tasks.push(t_j)$
    end if
  end for
  $update\text{-}group(G_w, G_m, G_t, tasks, w_i)$
end for
for $i = 1$ to $|G_w|$ **do**
  $schedule = \emptyset(length = N, initial\text{-}value = 0)$
  for $j = 1$ to $|G_m(g_i)|$ **do**
    $update\text{-}schedule(schedule, w_j, S_w)$
  end for
  $G_s(g_i) = schedule$
end for
return G_w,G_m,G_s,G_t

3.3 既存手法拡張型

本節で取り扱う既存手法拡張型の手法は，先行研究 [11] で提案された手法を前節で取り上げた割当てに対応できるように拡張するものである．先行研究で提案された手法は「できるタスクが少ないワーカを優先的にタスクに割当て，そのタスク結果が無駄にならないようにできるタスクが多いワーカを割り当てていく手法」である．本研究では時間制約を考慮するため，これを各期間において適用することを考える．

まず，簡単な例を用いて概要を説明する．例として，期間長：$N = 2$，ワークフロー：図 3，ワーカグループ：$\{G_1, G_2, G_3\}$ に対して割当てを行うことを考える．また，実行可能タスクは G_1：$\{V_1, V_2, V_3\}$，G_2：$\{V_1\}$，G_3：$\{V_2\}$ であり，期間 1 では G_1 と G_2 に，期間 2 では G_1 と G_3 に各 1 ずつのワーカを割当て可能であるとする．この時，本手法での割当て手順は以下の通り．

まず，ワークフローに対して可能なパスを探索すると，$P_1 = (V_{in}, V_1, V_2, V_{out})$，$P_2 = (V_{in}, V_3, V_{out})$ の 2 つが見つかる．これらをひな形に，各パスの先頭サブタスク $[P_1, 1]$ と $[P_2, 1]$ を処理待ちタスクに追加する．この時，例えば $[P_1, 1]$ はパス P_1 の 1 番目のサブタスクを表している．なお，V_{in} および V_{out} はパスの始点と終点を定義するものであり，実際のタスクを持たないため順番には含めない．次いで，処理待ちタスクをパスの完成までに必要な残りタスク数が昇順になるようにソートする．この場合，$[P_2, 1]$，$[P_1, 1]$ の順になる．また，ワーカのグループについても，グループが実行可能なタスク数の昇順になるようにソートする．この場合，$\{G_2, G_3, G_1\}$ である．これにより，実行可能なサブタスクが少ないワーカのグループに優先的にタスクが割当てられる．以上が初期化の処理である．

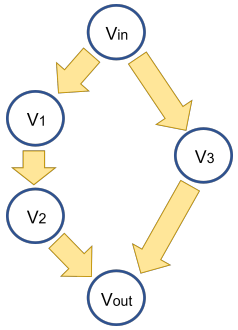

図 3: 割当て例用ワークフロー

次に，期間ごとの割当てを行う．期間ごとの割当てでは期間内で割当て可能であるワーカのグループのうち，ソートされたグループの先頭から順に割当てを行う．例の場合，期間 1 では G_2，G_1 の順である．グループへの割当てでは処理待ちタスクを先頭から参照する．例の場合まず，先頭 $[P_2, 1]$ を参照するが，該当するタスクは V_3 であり，G_2 はこれを実行できない．よって，次のタスク $[P_1, 1]$ を参照する．すると，該当するタスクは V_1 であり，G_2 はこれを実行できるため，G_2 に P_1 中の V_1 を割当て，割当て済タスクに $[P_1, 1]$ を追加する．また，これは P_1 中の先頭サブタスクであり，他のサブタスクの処理結果を必要としない．そのため，処理待ちタスク中から $[P_1, 1]$ は削除しない．次いで，G_1 の割当てを考えるが，G_1 には処理待ちタスクの先頭 $[P_2, 1]$ について該当するタスク V_3 を実行可能であるため，P_2 中の V_3 に割当て，割当て済タスクに $[P_2, 1]$ を追加する．これで期間 1 での処理は終了し，更新作業に移る．

更新作業では割当て済タスクを次の期間のために処理待ちタスクへ移行する．例の場合，$[P_1, 1]$ は次のサブタスクを表す $[P_1, 2]$ とし，該当するタスク V_2 が存在するため，これを処理待ちタスクに移行する．この際，P_1 は残り 1 つのタスクでパスが完成することから同様に残り 1 つのタスクでパスが完成するもののうち先頭に追加する．この場合，処理待ちタスクは $\{[P_1, 2], [P_2, 1], [P_1, 1]\}$ となる．次に，$[P_2, 2]$ を考えるが，これに該当するタスクは存在しない．よって，処理待ちタスクへは移行せず，パスが完成したものとし，処理済みタスクインスタンスをカウントする N_t を 1 加算する．以上で更新作業は終了である．

期間 2 では，G_3，G_1 の順にタスクを割り当てる．G_3 については処理待ちタスクの先頭 $[P_1, 2]$ に該当するタスクは V_2 で

あり，これを実行可能である．よって，P_1 中の V_2 に割当て，$[P_1, 2]$ を割当て済タスクに追加する．また，$[P_1, 2]$ はパス中先頭タスクではなく，先行するサブタスクの処理結果を使用・消費するため，これを処理待ちタスクから削除し，処理待ちタスクは $\{[P_2, 1], [P_1, 1]\}$ となる．以下同様にして，G_3 に P_2 中の V_3 が割当てられる．

最後に更新作業を行う．この際，さらに 2 つのパスが完成するため，$N_t = 1 + 2 = 3$ となり，割当ては期間 1 で G_1 に V_3，G_2 に V_1，期間 2 で G_1 に V_3，G_3 に V_2 が最終的な割当てとなる．以上が手法の概要である．続いて，アルゴリズムの詳細を以下に示す．まず，アルゴリズム中で使用している表記と関

Algorithm 2 Allocation Algorithm 1

Input: N, *Workflow*, A_t, G_w, G_s, G_t
Output: *Allocation*, N_t
Initialisation :
1: $Allocation = \emptyset$
2: $N_t = 0$
3: $processing = \emptyset$
4: $P = all\text{-}passes(Workflow)$
5: $P.sort!$
6: $G_w = sort\text{-}ability(G_w, G_t)$
7: **for** $i = 1$ to $|P|$ **do**
8: $processing.push([P_i, 1])$
9: **end for**
10: **for** $i = 1$ to N **do**
11: $allocated = \emptyset$
12: **for** $j = 1$ to $|G_w|$ **do**
13: $k = 1$
14: **while** k ¡ $G_s(i, g_j)$ **do**
15: **for** $m = 1$ to $|processing|$ **do**
16: $task = extract\text{-}task(processing_m)$
17: **if** $task \subseteq G_t(g_j)$ **then**
18: $update\text{-}pass(processing_m, allocated)$
19: $Allocation.push([i, g_j, task])$
20: $k = k + 1$
21: *break*
22: **end if**
23: **end for**
24: **end while**
25: **end for**
26: $processing = adjust(processing, N, i)$
27: $processing = merge(processing, allocated, N_t)$
28: **end for**
29: **return** *Allocation*, N_t

数について解説する．

- *all-passes(Workflow)*
  - *Workflow* 中の全てのパスを探索する
- $sort\text{-}ability(G_w, G_t)$
  - G_w 中のワーカグループについて，G_t を参照する
  - 各ワーカグループが実行可能なタスク数により G_w を昇順にソートする
- g_j
  - G_w 中に含まれる j 番目のワーカグループ
- $G_s(i, g_j)$
  - G_s からワーカグループ g_j のスケジュール i 番目の期間に登録されている数値（人数）を参照する
- $extract\text{-}task(processing_m)$
  - $processing_m$（$[P_s, t]$）について，パス P_s 中の t 番目のタスクを参照する
- $update\text{-}pass(processing_m, allocated)$
  - $processing_m$（$[P_s, t]$）について，*allocated* に $[P_s, t+1]$ を保存する
  - $t \neq 1$ の時，*processing* 中から $processing_m$ を取り除く
- $processing = adjust(processing, N, i)$
  - *processing* 中の完成しないパスを削除する関数である
  - *allocated* に含まれる個々のデータ $[P_u, v]$ を先頭から順に処理する
  - $|P_u| - v \geq N - i$ である場合には，u 番目以降のパスを全て P から削除する
- $merge(processing, allocated, N_t)$
  - *allocated* に含まれる個々のデータ $[P_u, v]$ を順に処理する
  - $v > |P_u|$ である場合には N_t に 1 を加算する
  - $v \leqq |P_u|$ である場合には *processing* の適切な位置に $[P_u, v]$ を挿入する
  - 適切な位置とは *processing* 中において，各データが $|P_u| - v$ の昇順に並ぶ状態をいう
  - $|P_u| - v$ が同値の場合には同値中の先頭に挿入する
  - *processing* 中において $|P_u| - v$

以下，アルゴリズムについて解説する．１～３行目では，各値を初期化している．*Allocation* は最終的な全割当てを [(割当て期間), (割当て先グループ名), (割当てサブタスク)] の形式で保存する．また，N_t は完成したパスの数を，*processing* は [(パス), (パス先頭からの順番)] の形式で処理途中（割当て可能）のパスを保存する．

４行目では，ワークフローに対して，全パスを探索し，P に保存している．

５，６行目では，P および，G_w をソートしている．P はパスを構成するサブタスクの数により，G_w はグループが実行可能なサブタスクの種類数により，いずれも昇順となるようにソートしている．

７～９行目では，*processing* に P に含まれる全パスについて先頭タスクを処理途中のパスとして登録している．

１０～２７行目では実際に割当てを行っている．１１～２６行目では各期間に行う処理を記述している．１１行目では該当期間に割当てが行われたパスを保存する *allocated* を初期化している．１２～２５行目ではワーカグループごとに先頭から行う処理を記述しており，１４～２４行目は該当期間にメインタスクへ参加可能としたワーカ１人ごとに処理を行うことを示している．１５～２３行目では，*processing*（処理中パスの集合）について先頭から該当ワーカグループに対してタスクを割当て可能であるかどうかを確認し割当てを行っている．より詳細には，１６行目で，パス中で次に処理されるべきタスクを抽出し，１７行目でワーカグループに当タスクを割当て可能であるかどうかを確認し，割当て可能な場合には１８～２１行目の処理を

実行する．１８行目では，*processing* から *allocated* に該当パスを移動し，重複して同一のパスにワーカを割り当てることを防ぐ．ただし，８行目で登録したパスについては重複して割当て可能（先行タスクが存在しないため）であるため，*allcoated* にコピーを作成する．１９行目では *Allocation* に割当てを保存する．２０～２１行目ではワーカ１人分の割当てを終了し，次のワーカの割当てに移行するための処理を行っている．最後に２６，２７行目で次の期間の割当てへと移行するための処理を行っている．まず，２６行目で残りの期間では完成しないパスを *processing* 中から削除する．２７行目では *allcoated* 中のパスについてパスが完成していれば，N_t を加算し，そうでなければ *processing* に「パス先頭からの順番」を１加算し，パスの完成までに必要なタスクの数が *processing* 中で昇順となる位置に挿入する．

以上の処理において，G_w がグループの実行可能なサブタスクの種類数により，また *processing* がパスの完成までに必要なタスク数の種類により昇順にソートされている．これにより，ワーカは実行可能なタスクの少ないワーカを優先的に割当て，また割当てたワーカが無駄にならないように完成しやすいパスに対して優先的に割当てを行うことができる．

3.4 最大流問題利用型

本節では最大流問題利用型の割当て手法について記す．最大流問題利用型の割当ては，各期間におけるワーカへのタスクインスタンスの割当て数を最大化する割当てを導出している．まず，３．２節と同様の例を用いて本手法の概要を説明する．ただし，本手法では割当てを求める際に G_t を利用し，ここでは $G_t = 3$ であるとする．また以下，$stock(V)$ の表現を利用する．これはサブタスク V について V までが終了しているタスクインスタンスの数を表し，初期値として $stock(V_{in}) = G_t$，その他のタスク V_n に対して $stock(V_n) = 0$ を設定する．また，終端を形成するサブタスク（V_{out} へと接続するタスク）の集合を E とする．この例では $E = \{V_2, V_3\}$ となる．

本手法では，各期間の割当てに重み付きグラフを作成する．期間１において作成されるグラフは図 4 である．なお，重みが 0 となるエッジは省略している．グラフ作成の詳細は後述するが，概要は図 6 に示す通りである．$Weight\_w$ はその期間に割当てることが可能なグループの人数を，$Weight\_t$ は集合２中のタスク T_2 について，$stock(T_2)$ で算出される．本手法ではこの作成した重み付きグラフに対してフォードファルカーソン法を用いて最大流を求めることで割当てを行う．図 4 のような場合，考えうるフローの 1 つは $(input, G_1, V_1, V_{in}, output)$ である．これは，ワーカグループ G_1 に対して，タスク V_1 をタスク V_{in} の処理結果を利用し，割り当てると解釈する．なお，このとき V_{in} の処理結果を利用すると述べたが，V_{in} は例外として処理するべきタスクインスタンスの最大数を管理するための処置であり，実際には処理するべきタスクが存在するわけではない．このようにして，1 つのフローを 1 つの割当てとみなし，最大流を求めることで期間内に可能な限り多くのワーカグループに割当てる手法が最大流問題利用型の手法である．期間１

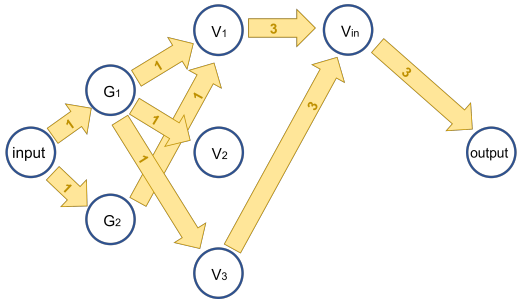

図 4: 期間１割当て用重み付きグラフ

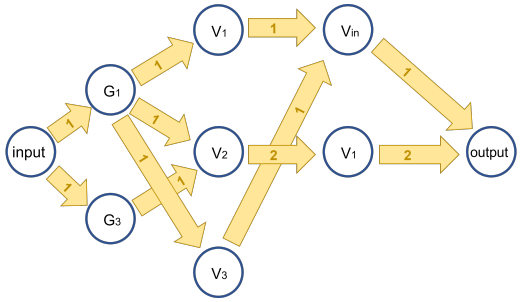

図 5: 期間２割当て用重み付きグラフ

では，考えうる最大流の 1 つは $(input, G_1, V_1, V_{in}, output)$ と $(input, G_2, V_1, V_{in}, output)$ の 2 つである．これより，G_1 に V_1 を，G_2 に V_1 を割当て，いずれも V_{in} の処理結果を利用する．次に，更新作業を行う．この時，V_{in} の結果を利用していることから，$stock(V_{in}) = 3 - 2 = 1$ また，V_1 に 2 人のワーカが割当てられていることから，$stock(V_1) = 0 + 2 = 2$ とする．以上で，更新作業を終了し，期間２の割当てに移る．

期間 2 で作成されるグラフは図 5 の通りである．ここから考えうる最大流の 1 つは $(input, G_1, V_1, V_{in}, output)$ と $(input, G_3, V_2, V_1, output)$ の 2 フローである．これより，更新作業を行い，$stock(V_{in}) = 1 - 1 = 0$，$stock(V_1) = 2 - 1 + 1 = 2$，$stock(V_2) = 0 + 1 = 1$ である．最後に，N_t の算出を行う．ここで，$N_t = \sum_{v \in E} stock(v)$ である．よって，$N_t = stock(V_2) + stock(V_3) = 1 + 0 = 1$ となる．以上より，$N_t = 1$ であり，割当てとして，期間１で G_1 と G_2 に V_1，期間２で G_1 に V_1，G_3 に V_2 の割当てが得られる．

以下，アルゴリズムの詳細について解説する．まず，アルゴリズム中で使用している関数について解説する．

- *terminal?*(E_i)
  - エッジ E_i が終端を表すノード V_{out} に向かうエッジであるかどうかを判定する
  - V_{out} に向かうエッジであれば *true* を返す
- *extract-terminal*(E_i)
  - V_{out} に向かうエッジから実際の終端ノードを抽出する
  - (V_n, V_{out}) のようなエッジの場合，V_n を抽出する
- *make-graff*$(stock, Workflow, A_t, G_w, G_s, G_t)$
  - 以下のような割当て用グラフ（図 6）を作成する
  - ワーカグループ集合には $|G_w|$ 個のノード（$g_i (\in G_w)$）が含まれる

– サブタスク集合１には V_{in} と V_{out} を除く V 中の各ノード（$v_j(\in V)$）が含まれる

– サブタスク集合２には V_{out} を除く V 中の各ノード（$v_k(\in V)$）が含まれる

– $Weight_w$ はエッジの始点もしくは終点となっているワーカグループ集合中のノード g_i について，$G_s(g_i)$ である

– $Weight_t$ はエッジの始点もしくは終点となっているサブタスク集合２中のノード v_k について，$stock(v_k)$ である

– $input$ からワーカグループ集合中の各ノード，およびサブタスク集合２中の各ノードから $output$ へはそれぞれ１本ずつのエッジを伸ばす

– ワーカグループ集合中の各ノード（g_i）からサブタスク集合１中の各ノード（v_j）へは $v_j \in G_t(g_i)$ である場合にエッジを伸ばす

– サブタスク集合１中の各ノード（v_j）からサブタスク集合２中の各ノード（v_k）へは $(v_k, v_j) \in E$ である場合にエッジを伸ばす

- $ford(graph)$

– $graph$ に対してフォードファルカーソン法を適用する

– 返り値を保存する $passes$ を初期化する

– 増大路を探索し，みつかった順に $passes$ に保存する

– 返り値として保存した増大路の集合 $passes$ を返す

- $simplyfy(simplified, roads_j)$

– 増大路 $roads_j$ を単純化する関数である

– 例えば，$(input, g_1, v_1, v_{in}, output)$ のような５つのノードから構成される増大路が最も単純な増大路である

– 一方で，フォードファルカーソン法の適用では増大路（$roads_j$）が

$p_a = (input, g_1, v_1, g_1, v_2, v_1, output)$ のような形で逆向きのエッジ（このパスの場合，v_1, g_1 の部分）を含むことがあり，このような場合，以下のような手順で単純な増大路へ変換する

– 逆向きのエッジに対して正順のエッジ（上記の例の場合 g_1, v_1）を考え，これを含む増大路 (この例のような場合 $p_b = (input, g_1, v_1, v_{in}, output)$ など) を $simplified$ 中から探索する（フォードファルカーソン法の性質上必ず存在する）

– 単純化されていない増大路（例：p_a）を当該逆向きエッジ（例：v_1, g_1）で２つのパスに分割する（例：$a_1 = (input, g_1, v_1)$ と $a_2 = (g_1, v_2, v_1, output)$）

– 次に，既に単純化されている増大路（例：p_b）を当該正順エッジを削除し，２つのパスに分割する（例：$b_1 = (input)$ と $b_2 = (v_{in}, output)$）

– それぞれの増大路から作成された４つのパスをクロスするように（例：a_1 と b_2，および b_1 と a_2）連結する（例：$(input, g_1, v_1, v_{in}, output)$ と $(input, g_1, v_2, v_1, output)$）

– 以上の手順により作成された２つのパスが両者ともに単純化されていれば，両者を $simplified$ に保存する，そうでなければ単純化されていないパスについて上述の手順を繰り返す

- $extract\text{-}allocation(i, simplified_j)$

– $simplified_j$ から割当てを取り出す関数である

– $simplified_j$ は５つのノードから構成される（例：

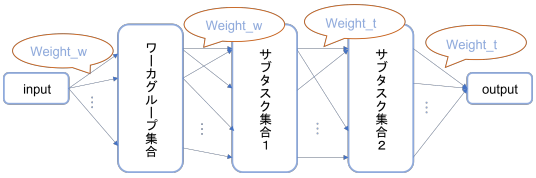

図 6: 割当て用グラフの作成

$(input, g_1, v_1, v_{in}, output)$）

– ５つのノードは順に，$input$，ワーカグループ集合中のノード（例：g_1），サブタスク集合１中のノード（例：v_1），サブタスク集合２中のノード（例：v_{in}），$output$ である

– これらのうち，ワーカグループ集合中のノード（例：g_1）が割当先ワーカグループ，サブタスク集合１中のノード（例：v_1）が割当てるサブタスク，サブタスク集合２中のノード（例：v_{in}）が必要となる先行サブタスクのインスタンスの供給元を表す

– 以上から割当てとして，割当て期間，割当先グループ，割当てるサブタスクのデータを取り出し，返す（例：$[2, g_1, v_1]$(ただし，この例では i=2)）

- $update\text{-}stock(simplified_j, stock)$

– $simplified_j$ により $stock$ の値を更新する関数である

– $simplified_j$ の構成は上述の通りであり，この関数ではサブタスク集合１中のノード v_m とサブタスク集合２中のノード v_n を使用する

– $stock(v_m) = stock(v_m) + 1$ および，$stock(v_n) = stock(v_n) - 1$ の処理を行い，タスクインスタンスの処理状況を記録する $stock$ を更新する

– $stock$ を返す

以下，アルゴリズムについて概説する．１～１６行目では，必要な変数の初期化を行っている．$Allocation$ は割当てを保存する変数，N_t は完成したパスの数をカウントする変数，$terminal$ はパスの終端となるサブタスクの集合を保存する変数である．４～８行目では，$terminal$ に該当するサブタスクを登録している．９～１６行目では，各サブタスクまでの処理済み（処理待ち）タスクインスタンスの数を保存する変数 $stock$ の初期化を行っている．なお，$stock(V_n)$ はサブタスク V_n について，そのタスクまで処理が終了している（次のサブタスクは終了していない）タスクインスタンスの数を表す．$stock(V_{in})$ には処理するべきタスクインスタンスの最大数を入れ，その他のサブタスク V_n（V_{out} のぞく）に対応する $stock(V_n)$ には０を入れる．

１７～２９行目では，実際に割当てを行っている．１８～２８行目は各期間ごとに行う処理である．１８行目では，フォードファルカーソン法の適用により探索された増大路の集合を保存する $roads$ を初期化している．１９行目では，当期間における割当て用グラフを作成し，$graph$ に保存している．２０行目では，作成した $graph$ に対してフォードファルカーソン法を適用し，$roads$ に順に増大路を登録している．２１～２３行目では得られた増大路の集合 $roads$ に対して，増大路を単純化することで最大流問題の解となるフローの集合 $simplified$ を求め

る．２４～２７行目では *simplified* に対して，当期間の割当てを保存するとともに，次の期間の割当てのために *stock* を更新している．２５行目では，各フローから割当てを導出し，期間と合わせて整形し *extracted* に保存している．２６行目では *extracted* を *Allocation* に保存している．２７行目では，各フローから *stock* の更新を行っている．

３０～３２行目では，N_t のカウントを行っている．各パスの終端となるサブタスクの集合に対して，各サブタスクの *stock* を求め，それらの合計を N_t とする．

Algorithm 3 Allocation Algorithm 2

Input: N, D_t, $Workflow$, A_t, G_w, G_s, G_t
Output: $Allocation$, N_t

Initialisation :
1: $Allocation = \emptyset$
2: $N_t = 0$
3: $terminal = \emptyset$
4: **for** $i = 1$ to $|E|$ **do**
5: **if** $terminal?(E_i) == true$ **then**
6: $terminal.push(extract\text{-}terminal(E_i))$
7: **end if**
8: **end for**
9: $stock = \emptyset$
10: **for** $i = 1$ to $|V|$ **do**
11: **if** $V_i == V_{in}$ **then**
12: $stock(V_{in}) = D_t$
13: **else**
14: $stock(V_i) = 0$
15: **end if**
16: **end for**
17: **for** $i = 1$ to N **do**
18: $roads = \emptyset$
19: $graph = make\text{-}graph(stock, Workflow, A_t, G_w, G_s, G_t)$
20: $roads = ford(graph)$
21: **for** $j = 1$ to $|roads|$ **do**
22: $simplified = simplyfy(simplyfied, roads_j)$
23: **end for**
24: **for** $j = 1$ to $|simplified|$ **do**
25: $extracted = extract\text{-}allocation(i, simplified_j)$
26: $Allocation.push(extracted)$
27: $stock = update\text{-}stock(simplified_j, stock)$
28: **end for**
29: **end for**
30: **for** $i = 1$ to $|terminal|$ **do**
31: $N_t = N_t + stock(terminal_i)$
32: **end for**
33: **return** $Allocation$, N_t

4 評価実験

本章では，評価実験について述べる．本研究では，シミュレーションにより評価実験を行った．実験ではワーカ生成確率 P，ワークフロー $Workflow$，手法 $Method$ の 3 つの要素を変化させ，実験条件を作成した．また，各条件につき 20 回の試行を行い平均化を行った．本研究では，2 種類の評価実験を行った．以下概説で試行 1 回の処理について解説し，その後，2 種類の評価実験について述べる．

4.1 概　　説

試行 1 回における実験の手順は以下の通り（図 2 参照）である．

（1） ワークフロー $Workflow$ およびワーカ生成確率 P に基づき 1 人のワーカを生成し，プールされたワーカ集合 W に追加する

（2） ワークフロー $Workflow$ を考慮し，W に対して割当て手法 $Method$ を適用する

（3） 予想処理可能タスク数 N_t を算出し，N_t と希望処理タスク数 D_t を比較する

（4） $N_t < D_t$ であれば 1 から 3 を繰り返す

（5） $N_t \geqq D_t$ であれば，結果を記録しシミュレーションを終了する

また，実験条件を構成する要素の詳細は以下の通りである．

- ワーカ生成確率 P

ワーカの能力数について以下の 3 通りの確率 P のもと，能力数 a を決定し，それに基づきワーカを生成する．ただし，ワーカの能力数についてサブタスクに必要な能力の種類の総数を a_{max}，サブタスクに必要な能力の最小数を a_{min} としたときに，$a_{half} = (a_{max} + a_{min})/2$ と定義し，a について,「能力が多い」状態を $a \geqq a_{half}$,「能力が少ない」状態を $a \leqq a_{half}$ とする．この時，$(30, 70)$ のような確率は「能力が多い」状態にあるワーカを生成する確率が 30%,「能力が少ない」状態にあるワーカを生成する確率が 70% であることを表している．

- $P(basic) = (50, 50)$
- $P(high) = (70, 30)$
- $P(low) = (30, 70)$

また，$a_{min} \leqq a \leqq a_{half}$ および，$a_{half} \leqq a \leqq a_{max}$ を満たすような整数 a が複数存在する場合には a の値は等確率で決定されるが，a_{half} が整数である場合にはその確率を他の値と比べて $\frac{1}{2}$ とする．これは，能力数 a_{half} であるワーカが過剰に生成されることを防ぐための処理である．その後，サブタスクに必要な能力全ての集合から，等確率で a 個の能力を生成するワーカに与える．ただし，与えられたタスクの集合ではいずれのサブタスクも実行不可能であるような場合には，a を変化させずに再度，タスクを a 個与える処理を繰り返す．また，ワーカ生成では，P に依らず以下の処理を行う．等確率で $1 \leqq n \leqq 5$ を満たす整数 n を決定し，期間長 N の範囲内で n 箇所をメインタスクへ参加可能であると表示し，ワーカのスケジュールとして付与する．

- ワークフロー $Workflow$

3 種類のワークフローを使用した．以下にその構造を示す．ただし，ここでいう $Workflow$ には前述のサブタスクに必要な能力 A_t も含まれるものとする．

- $Workflow(A)$ シンプルな構造をもつ仮想のワークフローである．構造は図 7 の通り．また，サブタスクに必要な能力は以下の通り．
  - $Ability = \{a_1, a_2, a_3, a_4\}$
  - $A_t = \{(V_1, [a_1]), (V_2, [a_2]), (V_3, [a_3)], (V_4, [a_4]),$

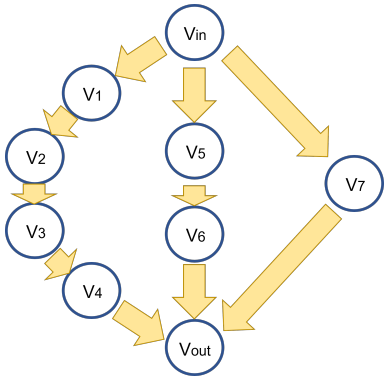

図 7: Workflow(A)

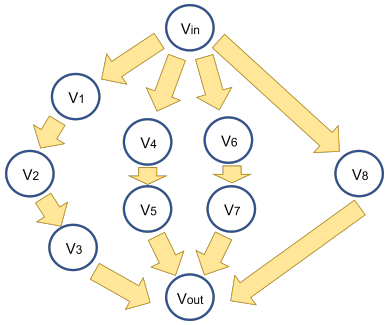

図 8: Workflow(B)

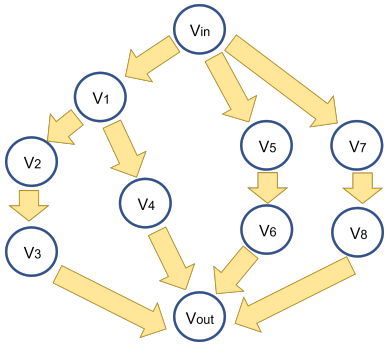

図 9: Workflow(C)

$(V_5, [a_1, a_2]), (V_6, [a_3, a_4]), (V_7, [a_1, a_2, a_3, a_4])\}$

– $Workflow(B)$ 日本手話を英語の手話に翻訳するというメインタスクに対して，実際に使用されたワークフローである [11]．構造は図 8 の通り．また，サブタスクに必要な能力は以下の通り．

＊ $Ability = \{JSL, J, E, ASL\}$

＊ $A_t = \{(V_1, [JSL, J]), (V_2, [J, E]), (V_3, [E, ASL]), (V_4, [JSL, J, E]), (V_5, [E, ASL]), (V_6, [JSL, J]), (V_7, [J, E, ASL]), (V_8, [JSL, J, E, ASL])\}$

– $Workflow(C)$ 音声データを文字データに書き起こすというメインタスクに対して考えられるワークフローである [11]．ただし，本研究で使用可能なワークフローの構造に合わせるため，引用元と比較して統合のサブタスクを削除している．また，音声認識のサブタスクは引用元では記されているが，実際には機械により自動化された作業であるため，本論文では省略している．構造は図 9 の通り．また，サブタスクに必要な能力は以下の通り．

＊ $Ability = \{a_1, a_2, a_3, a_4\}$

＊ $A_t = \{(V_1, [a_1]), (V_2, [a_2]), (V_3, [a_1]), (V_4, [a_1, a_3]), (V_5, [a_1, a_3]), (V_6, [a_4]), (V_7, [a_4]), (V_8, [a_1])$

- 手法 $Method$

3 章で示した 2 つの提案手法のいずれかを指定する．

– $Method(m_1)$ 既存手法拡張型の手法を適用する．なお，手法の比較実験では，こちらの手法をベースラインとして，最大流問題利用型の手法を評価する．

– $Method(m_2)$ 最大流問題利用型の手法を適用する．手法の比較実験では主にこちらの手法について評価することを目的とする．

また，全ての実験条件で共通して使用する値（要素）は以下の通り．

- 期間長 N

$N = 60$

- 希望処理タスク数 D_t

$D_t = 100$

次に，評価実験において取得するデータについて解説する．評価実験の結果として取得する指標は以下の通りである．

- $inclusion$ インクルージョン性を評価する指標である．定義は 2 章に示した通りである．
- $influx$ $N_t \geq D_t$ を満たすために要したワーカの数であり，$influx = |W|$ で定義する．割当ての効率性を評価するための指標である．値が小さいほど，より少ないワーカにより目標数のインスタンスを処理できたことを表す．
- $loss$ ワーカを割り当てたが，パス内に存在する全てのサブタスクにワーカを割り当てることができずインスタンスの処理が終了しなかったもの，もしくは割当てを行ったが $N_t > D_t$ となり余剰となったインスタンスの割合である．このようなパスの数を n_l として，$\frac{n_l}{D_t}$ と定義する．値が小さいほど，より無駄の少ない割当てであることを表す．

5 手法の比較評価

本節では 2 手法を比較評価する評価実験について述べる．ベースラインとして既存手法拡張型の手法を設定し，最大流問題利用型の手法の評価を行うことが目的である．詳細は以下に述べる通り．

実験条件はワーカ生成確率 P を 3 通り，ワークフロー $Workflow$ を 3 通りから作成した計 9 通りのワーカのフローとワークフローの組に対して，2 つの手法 $Method$ を適用した計 18 の実験条件を使用した．この際，各条件に対して，20 回の試行を行い平均化を行っている．また，使用したワーカのフローの違いによる影響を軽減するため，ワーカのフローについてはあらかじめ十分な数のワーカのリストを条件に基づき生成した．このリストについて，両手法とも同一のリストを 1 試行分として先頭から読み込むことで疑似的なワーカのフローを再現し，なおかつワーカのフローの違いによる影響を軽減している．これらの実験条件について，以降 $basicCm_1$ のよう

な形式で表現し，この例のような場合，実験条件が $P(basic)$，$Workflow(C)$，$Method(m_1)$ であることを示す．また，指標には *inclusion*，*influx*，*loss* を使用する．

5.1 手法の特性評価

本節では手法の特性を検証する評価実験について述べる．本節で述べる評価実験の目的は，割当てに時間経過（スケジュール）の考えを導入することによる利点を検証する点にある．これは以下のような想定のもと行う実験である．

本研究では，時間経過（スケジュール）を取り入れることにより，$30(min) \times 12(set) \times 5(day)$ のような形式で長期間にわたるメインタスクの募集に対応可能であり，かつ 1 回あたりの拘束時間は 1 セット分 (30 分) に抑えることが可能である．一方，従来の割当て手法では時間経過が考慮されないため，例えば 5 日間で行うメインタスクへの参加は期間中全ての拘束時間（上記例と同様の規模の場合 30 時間）を生じる．しかし，このような長期の拘束は現実的とはいえず，実際には期間の分割等により対応することが想定される．一方で，期間を分割する場合においても課題は存在する．従来手法には著者の知る限り処理途中のインスタンスを考慮し，次の期間に引き継ぐといった期間をまたぐような割当て手法は存在しない．それゆえ，全ての期間を一括して割当てを行う場合と比較するとロスが生じることが想定される．これにより，従来手法の適用により募集を大規模化（長期化）する場合には，このようなロスを考える必要がある．そこで，本研究では，疑似的に期間の分割を再現することで，こうしたロスを生じない本研究の手法の有用性を検証する．具体的な実験の詳細は以下の通りである．

前述の実験条件に加え，実験条件を構成する要素に分割数 *Sep* を追加する．*Sep* は期間長を *Sep* の数に等分割することを示しており，例えば $Sep = 3, N = 60$ の時，$N = 20$ である期間 3 つに分割することを示す．本評価実験では，*Sep* を $1 \leqq Sep \leqq 6 (Sep \in \mathbb{N})$ の範囲で変化させる．割当て手法の適用については，分割された各期間 $m (1 \leqq m \leqq Sep)$ について独立に割当て手法を適用し，期間 m で想定される処理可能タスク数 $N_t(m)$ を算出，$N_t = \sum_{m=1}^{Sep} N_t(m)$ であるとする．また，ワーカのフローの生成等については概ね上述と同様の手順を踏むが，ワーカのスケジュールについては，先頭から *Sep* に分割したものを順に期間 m のワーカのスケジュールであるとする．例えば，$N = 6, Sep = 2$ である時に $S_w(w_1) = [1, 0, 1, 1, 0, 0]$ であるワーカ w_1 が生成されている場合，期間 $m = 1$ において $S_w(w_1) = [1, 0, 1]$，$m = 2$ において $S_w(w_1) = [1, 0, 0]$ であるとみなす．つまり，ワーカ w_1 が W に追加されるということは，分割された各期間全てについてワーカ w_1 が分割されたスケジュールを以って参加することを示す．

本実験ではワーカ生成確率 P を $P(basic)$ の 1 通り，ワークフロー $Workflow$ を 3 通り，手法 *Method* を 2 通り，分割数 *Sep* を $1 \leqq Sep \leqq 6 (Sep \in \mathbb{N})$ の 6 通り，計 36 通りの実験条件を使用した．また，指標には *inclusion*，*influx*，*loss* を使用する．

表 3: Workflow(A)

条件	inclusion	influx	loss
$highAm_1$	94.0	70.1	11.6
$highAm_2$	96.9	63.2	0
$basicAm_1$	93.7	84.0	17.3
$basicAm_2$	96.7	77.4	0
$lowAm_1$	94.7	103.0	16.5
$lowAm_2$	97.1	97.0	0

表 4: Workflow(B)

条件	inclusion	influx	loss
$highBm_1$	75.2	67.7	5.5
$highBm_2$	85.3	58.4	0
$basicBm_1$	68.4	88.7	5.5
$basicBm_2$	79.5	78.6	0
$lowBm_1$	60.5	123.7	25.2
$lowBm_2$	72.7	108.0	0

表 5: Workflow(C)

条件	inclusion	influx	loss
$highCm_1$	97.2	68.5	5.9
$highCm_2$	98.6	68.7	0
$basicCm_1$	94.5	71.7	5.4
$basicCm_2$	98.1	72.2	0
$lowCm_1$	93.4	73.0	5.1
$lowCm_2$	98.2	73.6	0

6 実験結果と考察

6.1 手法の比較評価

手法の比較評価実験の結果を以下に示す．表 3，表 4，表 5 にワークフローごとの結果を表す． 以下，ワークフローごとに結果を確認する．

まず，$Workflow(A)$ について，*inclusion*，$influx$，*loss* のいずれについても最大流問題利用型の手法が優位であった．また，ワーカのもつ能力数による影響は，*inclusion* に関してはいずれの手法を取ったとしても大きな違いは生じていない．一方，$influx$ については，いずれの手法についても，ワーカの能力数が少ないほど低い多くのワーカを必要とする傾向が存在することがわかる．*loss* については手法の性質上，最大流問題利用型の手法についてはいずれにおいても 0 を維持している．既存手法拡張型の手法については能力数の多いワーカが過半数となる条件と，平均的な分布の条件とで違いが生じているが，一方で，能力数が少ないワーカが過半数となる条件と平均的な分布との間では大きな違いは生じていない．

次いで，$Workflow(B)$ についてであるが，こちらも *inclusion*，$influx$，*loss* のいずれについても，最大流問題利用型の手法が優位である．また，ワーカのもつ能力数による影響は *inclusion* および $influx$ において見られ，手法の違いを問わず能力数が多いワーカの割合が高いほど良い結果を示す

ことが見て取れる．また，$loss$ について，最大流問題利用型の手法については $Workflow(A)$ と同様に常に 0 を示している．一方，既存手法拡張型の手法については，$Workflow(A)$ の条件とは異なり，ワーカの分布が平均的である条件と，能力数の低いワーカの割合が高い条件との間で大きな違いができている．

$Workflow(C)$ については，他の 2 つの条件と異なり，$inclusion$ において，既存手法拡張型の手法が優位を示している．また，$influx$ については手法間に大きな違いは見られず，$loss$ については，引き続き最大流問題利用型の手法が優位である．ワーカの能力数による影響については，$inclusion$ において手法間で違いが見られ，既存手法拡張型の手法はほとんど影響を受けないのに対して，最大流問題利用型の手法では能力数が少ないワーカの割合が高くなるにつれ低下する傾向が見られる．$influx$ については，影響は小さいものの，両手法とも能力数が少ないワーカが増えるにつれ $influx$ が増加する傾向が見て取れる．$loss$ については，最大流問題利用型の手法では他のワークフローと同様である．一方，既存手法拡張型の手法では，能力数の少ないワーカの割合が高いほど $loss$ が減少するという通常想定されるものとは逆の結果が見て取れる．しかし，他のワークフローの条件と異なり，その差は 1 未満であり，誤差であると考えられる．

以上から，最大流問題利用型の手法は必要なワーカの数を抑える，未完成のパスを減らすという評価観点において既存手法拡張型の手法に比べ優れているといえる．また，インクルージョン性についても，多くの場合において高い値を取る．以上から，最大流問題利用型の手法は概ね既存手法拡張型の手法よりも優れた手法であると判断できる．一方で，グラフの形状・条件により，結果に影響があるということも示されており，この点についてはワークフローの多様性への対応のため，さらなる追究が必要である．

6.2 手法の特性評価

手法の特性評価実験の結果，分割数の増加による影響は主に $influx$，$loss$ の割当の効率性に関係する指標に表れることが分かった．なお，詳細な結果については紙幅の都合上割愛する．

7 まとめ

本論文では，時間の明示化という概念をクラウドソーシングのタスク割当てに導入し，それに対する割当て手法を 2 種類提案した．1 つは既存手法に時間制約を取り入れ拡張したものであり，もう一方は最大流問題を活用し，割当てを行う手法である．これらの手法に対して，評価実験ではワーカの能力数の分布，ワークフローの条件を変化させ 2 手法間の比較および，手法の特性についてワーカのインクルージョン性，必要なワーカ数，割当てのロスの 3 つの観点から検証した．手法間の比較により，最大流問題を利用した手法が多くの条件で優位であること，また，時間の明示化という概念を導入することによる手法の特性として，主に割当ての効率性（必要なワーカ数，割当てのロス）という面で効果的であることが示せた．

文　献

[1] Ho, Chien-Ju, Vaughan, Jennifer. Proceedings of the ... AAAI Conference on Artificial Intelligence: Adaptive Task Assignment for Crowdsourced Classification. 2021, vol. 26, no. 1, p. 45–51.

[2] Sihem Amer-Yahia, Senjuti Basu Roy. Toward Worker-Centric Crowdsourcing. Bulletin of the Technical Committee on Data Engineering. 2016, vol. 39, no. 4, p. 3-13.

[3] Kulkarni, Anand, Can, Matthew, Hartmann, Björn. Collaboratively crowdsourcing workflows with turkomatic. ACM, 2012, 1003–1012p., (CSCW).

[4] 西 記代子. MLA 機関によるクラウドソーシングの活用について : 欧米の事例と日本における導入の可能性 (小特集 アーカイブズ・ノート). 国文学研究資料館紀要. アーカイブズ研究篇 = The bulletin of the National Institute of Japanese Literature. 人間文化研究機構国文学研究資料館 編. 2018, no. 14, p. 103-112.

[5] Ambati, Vamshi, Vogel, Stephan, Carbonell, Jaime. Collaborative workflow for crowdsourcing translation. ACM, 2012, 1191–1194p., (CSCW).

[6] Tran-Thanh, Long, Huynh, Trung Dong, Rosenfeld, Avi, Ramchurn, Sarvapali, Jennings, Nicholas. Proceedings of the ... AAAI Conference on Artificial Intelligence: Crowdsourcing Complex Workflows under Budget Constraints. 2015, vol. 29, no. 1.

[7] Jiang, Jiuchuan, An, Bo, Jiang, Yichuan, Shi, Peng, Bu, Zhan, Cao, Jie. IEEE transactions on parallel and distributed systems: Batch Allocation for Tasks with Overlapping Skill Requirements in Crowdsourcing. 2019, vol. 30, no. 8, p. 1722–1737. https://ieeexplore.ieee.org/document/8620370.

[8] Xiao, Lei, Gao, Zhigang, Xu, Ruichao, Wu, Bo, Zheng, Leilei, Cen, Weipeng, He, Xuanzhang, Zhao, Wei. An allocation method of crowdsourcing tasks for protecting fairness of participants.

[9] Goto, Shinsuke, Ishida, Toru, Lin, Donghui. Understanding Crowdsourcing Workflow: Modeling and Optimizing Iterative and Parallel Processes. 2016.

[10] Ho, Chien-Ju, Vaughan, Jennifer. Proceedings of the ... AAAI Conference on Artificial Intelligence: Online Task Assignment in Crowdsourcing Markets. 2021, vol. 26, no. 1, p. 45–51.

[11] A Task Assignment Method Considering Inclusiveness and Activity Degree, Hashimoto,H.; Matsubara,M.; Shiraishi,Y.; Wakatsuki,D.; Zhang,J.; Morishima,A. IEEE, Dec 2018, p. 3498-3503.

[12] U. Pferschy and J. Schauer, The maximum flow problem with disjunctive constraints, Journal of Comb. Optim. (2013), 26:109-119

[13] 総務省. "ICT による多様な人材の労働参加促進: クラウドソーシングの広がり". 情報通信白書. https://www.soumu.go.jp/johotsusintokei/whitepaper/ja/h30/html/nd16 (accessed 2020-7-15).

[14] Rokicki, Markus, Zerr, Sergej, Siersdorfer, Stefan. Groupsourcing. ACM, 2015, 906–915p.

[15] Wang, Wanyuan, Jiang, Jiuchuan, An, Bo, Jiang, Yichuan, Chen, Bing. IEEE transactions on cybernetics: Toward Efficient Team Formation for Crowdsourcing in Noncooperative Social Networks. 2017, vol. 47, no. 12, p. 4208–4222.

[16] Jiang, Jiuchuan, Zhou, Yifeng, Jiang, Yichuan, Bu, Zhan, Cao, Jie. Knowledge-based systems: Batch allocation for decomposition-based complex task crowdsourcing e-markets in social networks. 2020, vol. 194.