複数レシピで並行調理する際の調理環境に応じた最適調理手順作成法と評価

中部 仁[1] 水本 旭洋[2] 諏訪 博彦[1,3] 安本 慶一[1]

概要：近年，自分で料理を作る人が増加しており，複数の料理を含む献立を効率よく調理できるような最適調理手順を提示することが求められている．一方で，調理者ごとに調理環境が異なり，最適調理手順は調理者によって多種多様になるため，最適調理手順を調理者自身が考えることは困難である．そこで本研究では，各レシピのタスクグラフを作成し，複数レシピの並行調理をタスクスケジューリング問題として捉えた最適調理手順の作成法を提案する．タスクスケジューリング問題の最適化アルゴリズム DF/IHS 法をベースとし，前処理部と枝刈り部に料理特有の洗い物を考慮した拡張を行うことで，効率よく最適調理手順を探索する．評価実験において，提案アルゴリズムを，実際のレシピから作成した献立に対して実行したところ，手動で作成した調理手順と比較し，洗い物も含め合計調理時間を 15%短縮できる最適調理手順を探索できていることを確認した．

Minimizing Cooking Time for Multiple Parallel Recipes Under Arbitrary Cooking Environment and Investigating Its Effect

JIN NAKABE[1] TERUHIRO MIZUMOTO[2] HIROHIKO SUWA[1,3] KEIICHI YASUMOTO[1]

1. はじめに

食は，人間の健康な生活を支える重要な要素である．「人生 100 年時代」[1] と言われるように人々が 100 年近く健康な生活を送る上で，健康管理や生活習慣病の予防のために，塩分や油分の量を自身で調整・把握できる手作り料理の重要性は高い．また，共働き世帯数が増加している [2] 現代では，調理に長時間を費やすことなく短時間で一汁三菜のようなバランスの良い料理を作ることが望ましい．しかしながら，調理に不慣れな初心者は，調理時間を効率化する方法を知らないため，調理に長時間を要する．

調理に不慣れな初心者をサポートする手段として，cookpad[3] や allrecipes.com[4] のような手作り料理の知恵を検索・相互交換可能な Web サイトが注目されている．これらの Web サイトでは，自分以外の調理者が投稿したレシピの閲覧や調理時間の短縮の工夫を閲覧でき，時間短縮の知識を容易に得ることができる．調理手順についても文字ベースに加えて，写真や動画など詳しい手順を把握することができるため，これらの Web サイトからその日に調理する料理を決定する人が増えている．

一方で，料理を複数作る際に，調理時間短縮のためにどの調理作業同士が並行調理できるかを把握することは難しい．並行調理できるかは，調理器具の数などのユーザの調理環境や，各調理作業に必要とする時間に依存しているためユーザによって異なる．よって複数の料理を作る際に，複数のレシピの依存関係，ユーザの調理環境，ユーザが各調理作業に必要とする時間を考慮した最適調理手順を提示することが求められる．

本研究では，複数レシピで並行調理する際の調理環境に応じた最適調理手順の自動作成法の提案と評価を行う．ここで最適調理手順とは，調理開始から終了までの時間が最短であることを指す．作成する最適調理手順には，レシピには存在していないが調理中に発生する洗い物も調理作業に含める．ここで，ユーザごとに，調理器具の個数などの調理環境は違うことから，調理環境はリソースとして扱う．必要とされるリソースは，調理作業ごとに決められており，リソースを割り当てることにより，その調理作業は実行できる．また各調理作業に必要とする時間は，ユーザごとの各調理作業に必要とする時間の違いを考慮するために，ユーザが過去にその調理作業を実行するのに実際に必要とした時間を用いる．

最適調理手順を作成するためには，効率的な並行調理を

[1] 奈良先端科学技術大学院大学，Nara Institute of Science and Technology
[2] 大阪大学大学院情報科学研究科, Graduate school of Information Science and Technology, Osaka University
[3] 理化学研究所 革新知能統合研究センター（AIP），RIKEN Center for Advanced Intelligence Project (AIP)

含む調理手順を探索する必要がある．しかしながら，複数レシピの調理作業へのリソースの割り当ては膨大であり，すべての割り当てを探索するのは現実的ではない．そこで，最適調理手順の探索をタスクスケジューリング問題として捉え，既存の最適化アルゴリズム DF/IHS 法 [5] で用いられている前処理部と枝刈り法を，料理で発生する洗い物も含めることでより料理の特性を活かせるように拡張し，最適調理手順の探索問題へ適用する．これによりユーザの調理作業時間・調理環境にあわせた調理手順を自動的に実時間で作成する．

評価実験において，提案アルゴリズムを，実際のレシピから作成した献立に対して実行したところ，手動で作成した調理手順と比較し，洗い物も含め合計調理時間を 15% 短縮できる最適調理手順を探索できていることを確認した．また，探索数の増加に対して，実行時間の増加をおよそ半分に抑えられていることを確認した．

本稿の章構成は以下の通りである．2 章では，提案手法に関連した既存研究を概説する．その後，3 章では，タスクスケジューリング問題と最適調理手順探索問題の関連付けを行う．4 章では，本研究で対象とする最適調理手順作成問題を定式化し，5 章で，最適調理手順アルゴリズムの作成方法について述べる．6 章では，作成した最適調理手順探索アルゴリズムの評価を行う．最後に，7 章で本論文のまとめを述べる．

2. 関連研究

本章では，本研究に関連する研究である，調理手順のスケジューリングに関する研究，タスクスケジューリングに関する研究について述べる．

調理手順のスケジューリングに関する研究として，松島ら [6] は，ある調理モデルに対する手順の決まった複数の料理の最短時間調理手順を SA 法を用いて求めている．この研究では，全ての調理作業を「切る」,「混ぜる」など 6 種類に分類しており，食材や調理者の熟練度などによって決定した各調理作業時間を用いてスケジューリングを行なっている．この時，調理器具の個数などの調理環境の変化にも対応している．一方で，調理時に発生する洗い物や，同一レシピ内での作業順序の入れ替えは考慮されていない．これは，作成した調理手順通りに調理作業を実施することができないこと，また複数レシピで並列調理する際に発生する空き時間に実行可能な調理作業があったとしても割り当てが行われないことが考えられ，最適調理手順とはならない．

また，山吹ら [7] は，複数の料理を複数人で調理を行う場合の調理手順スケジューリングを行なっている．この研究では，調理作業をジョブ，料理人を機械と対応させることにより資源制約付きプロジェクトスケジューリング問題（Resource constrained project scheduling problem, RCPSP）を拡張したものとして捉え，SA 法を用いて調理手順を求めている．一方で，この研究も洗い物を考慮していない．また調理作業を RCPSP と捉える際に，料理人のみを機械としているため，各調理作業は料理人を割り当てられた時に実行可能としているが，調理器具の個数などの制約は考慮されていないため実行不可な場合が考えられる．

最短時間の調理手順をスケジューリングする問題は，調理作業をタスク，調理作業で使用するリソースをプロセッサとするタスクスケジューリング問題として捉えることができる．タスクスケジューリング問題の解を求めるタスクスケジューリングアルゴリズムは，短時間で近似解を得るヒューリスティックアルゴリズムと，最適解を得る最適化スケジューリングアルゴリズムに分類される．

近似解を得るヒューリスティックアルゴリズムとして，実行可能となったタスクを随時実行するリストスケジューリング法 [8]，タスクグラフの出口ノードから各タスクまでの最長パス長 (以下 CP 長) の大きいタスクから割り当てていく CP 法 [9]，CP 長が大きいタスクの割り当てを優先するが，同一の CP 長を持つ複数のタスクがある場合は後続のタスク数が多いものを優先する CP/MISF 法 [5] などが提案されている．

最適解を得る最適化スケジューリングアルゴリズムとして，笠原ら [5][10] は，CP/MISF 法により初期解を得，そのタスク割り当て優先度を用いて探索順を決定するとともに，シャープな下界 [11] により限定操作を行う深さ優先分枝限定法である逐次最適化アルゴリズム DF/IHS 法と，それを並列処理用に拡張した並列アルゴリズム PDF/IHS 法を提案している．また DF/IHS 法の改良として，中村ら [12] は，散策中のノードと探索済みノードを比較し DF/IHS 法の探索ノード数の削減を行い，松瀬ら [13] は，その比較にハッシュテーブルを用いた手法を提案している．

関連して DF/IHS 法にデータ転送オーバヘッドを考慮した最適化アルゴリズムとして，澁谷ら [14] は，各タスクから限定された範囲の後続タスクまでの通信遅延を考慮した下限値を求める REDIC 法 [15] を加えたアルゴリズムを提案している．またこのアルゴリズムの改良として，下限計算においてグループ化可能な部分タスクグラフの最適化やによってより精度の高い下限計算を行うアルゴリズム [16] や，部分タスクグラフと全体タスクグラフを複数回スケジューリングする階層的スケジューリングによる高速化を図るアルゴリズム [17] が提案されている．

調理手順のスケジューリングに関する研究では，調理中に本来発生する洗い物は考慮されておらず，出力されたスケジューリングは，最適調理手順でなかったり実行不可能なものであることが考えられる．また既存の最適化スケジューリングアルゴリズムをそのまま用いるだけでは，料理の特徴を捉えることができず，効率的な探索を行うことはできない．そこで本研究では，複数レシピに対する最適

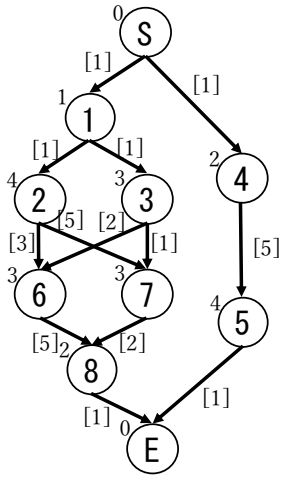
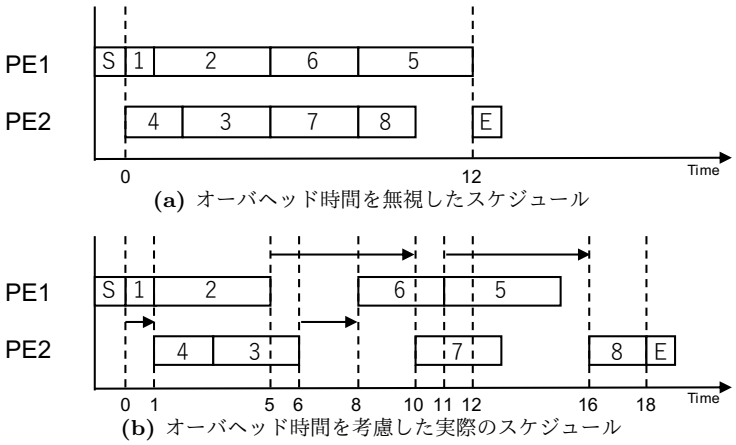


図 1: タスクグラフの例



(a) オーバヘッド時間を無視したスケジュール

(b) オーバヘッド時間を考慮した実際のスケジュール

図 2: オーバヘッド時間によるスケジュールの変化

調理手順を探索する際に，洗い物を含めた上で調理時間が最短となる最適調理手順の作成を目指す．また調理作業ごとに必要とされるリソースが存在することから，調理作業をタスク，各リソースをプロセッサとしたタスクスケジューリング問題として捉え，前処理と探索どちらにも料理の特徴を含むように改良することができる DF/IHS 法をベースとしたアルゴリズムを提案する．

3. タスクスケジューリング問題の最適調理手順探索問題への拡張

本研究では，最適調理手順の探索問題をタスクスケジューリング問題として捉え，既存のタスクスケジューリングアルゴリズムを拡張し，最適調理手順の探索へ適用することで探索時間の短縮を試みる．本章では，タスクスケジューリング問題と最適調理手順探索問題の関係を述べる．

3.1 タスクスケジューリング問題

タスクスケジューリング問題は，任意の処理時間，任意の先行制約を持つ n 個のタスク集合 $T = \{t_1, t_2, \ldots, t_n\}$ を能力の等しい m 台のプロセッサで並列処理する際に，その実行時間（スケジュール長）を最小にするスケジュールを求める問題であり，強 NP 困難な問題であることが知られている [18]．このタスク集合 T はタスクグラフ $G(T, E)$（E は有向エッジの集合）として DAG（有向非巡回グラフ）で記述され，1 つの開始ノード S と 1 つの終了ノード E を持ち，全てのノードは入口ノードから到達可能であるとする．タスクグラフの例を図 1 に示す．図 1 は，8 つのタスクからなるタスクグラフの例である．図中のノードは 1 つのタスク，ノード内の数字はタスク番号，ノード左上の数字はタスクの処理時間，エッジ上の数字はオーバヘッド時間を表している．また図中の各有向エッジはタスク同士の半順序制約を表す．

3.2 タスクスケジューリング問題のオーバヘッド時間

実際の並列環境では，先行タスクと後続タスクが異なるプロセッサで処理される場合，後続タスクが必要な先行タスクの情報をプロセッサ間で送受信する通信が発生する．この通信は，タスクスケジューリング問題においてオーバヘッドとして定義され，オーバヘッド時間分の通信時間が発生する．オーバヘッドがスケジュールにどのような影響を与えるかを示すために，図 1 のタスクグラフの 2 プロセッサへの割り当てを例として，オーバヘッド時間を無視したスケジュールとオーバヘッド時間を考慮した実際のスケジュールを図 2 に示す．図は横軸が時間，縦軸がプロセッサ名であり，横棒が記入されているタスク番号を実行している時間を表している．また，図中の矢印は，オーバヘッド時間が発生することによって，タスクの開始可能な時間が遅れていることを表している．

図 2 の (a)，(b) を比較すると，実際にはオーバヘッド時間により各タスクの開始可能時間が遅れることで実行時間に差が出てくることがわかる．よってオーバヘッドの存在する問題に対するタスクスケジューリング問題では，このオーバヘッド時間を考慮した上で実行時間を最小とするスケジュールを求めることが必要である．

3.3 最適調理手順探索における各種作業の扱い

最適調理手順の探索は各レシピにおける調理作業をタスク，調理作業で使用できるリソース (調理者，包丁，まな板など) をプロセッサ，洗い物をオーバヘッド時間として捉えることによりタスクスケジューリング問題として扱うことができる．また，調理環境はユーザによって異なるため，ユーザの調理環境を考慮した上で，調理作業をどのリソースを用いて行えば効率よく調理時間を短縮できるかを考えなくてはならない．さらに，本研究の最適調理手順の探索において，既存のタスクスケジューリング問題との大きな相違点は，洗い物の扱いである．タスクスケジューリ

ング問題のオーバヘッドは，前述したように前後関係のある先行タスクと後続タスクが，異なるプロセッサで処理される場合に発生する．一方で，料理における洗い物は，使用するリソースが異なるときではなく，使用するリソースを最後に使用した調理作業によって決定する．例えば，野菜を切ろうとした際に，使用する包丁で肉を切る調理作業を行っていた場合には，調理作業間で前後関係がなくても洗い物をする必要がある．またタスクスケジューリング問題では，オーバヘッド時間は前後関係のあるタスク間ごとに事前に全て定義することが可能であるが，最適調理手順の探索においては事前に定義できない可能性がある．例えば，ある調理工程で使用するリソースのうち，包丁とまな板は洗い物が必要であるが，ボウルは洗い物が必要ないなどその時のリソースの状態によって洗い物の必要なリソースが変化するためである．この相違点を考慮したアルゴリズムを作成しなければならない．

4. 問題設定

本章では，本研究で対象とする最適調理手順作成問題を定式化する．

4.1 レシピ情報

調理者が調理したい料理のレシピ集合を $RR = \{R_1, R_2, \ldots, R_k\}$ とする．ここで k は調理者が選択したレシピ数である．各レシピ $R_i (1 \leq i \leq k)$ に含まれる調理作業の集合を $R_i = \{r_{i_1}, r_{i_2}, \ldots, r_{i_{l_i}}\}$ とする．ここで l_i は，レシピ R_i の調理作業番号でありレシピによって l_i の値は異なる．また本研究で扱う調理作業は，「1. タマネギとキャベツを切る」ではなく，「1. タマネギを切る」「2. キャベツを切る」のように細分化されているものとする．

各レシピの作業には，先に挙げた「1. 玉ねぎを切る」，「2. キャベツを切る」のようにレシピ内で順序の入れ替えが可能なものがある．各レシピ R_i において，順序の入れ替えが可能な作業群を 1 つの集合にまとめ，それら集合を作業順に並べた列を $P_i = \langle P_{i_1}, P_{i_2}, \ldots, P_{i_{h_i}} \rangle$ とする．ここで h_i は，入替可能な作業集合の数でありレシピによって h_i の値は異なる．

レシピに含まれる調理作業 r は，{ 食材，分量，調理方法，使用リソース，調理時間，種別，開始条件，直前調理作業 } を要素として持つ．食材は，その調理作業に使用する食材名であり，複数の食材を使用するまたはどの食材も使用しない場合は $NULL$ とする．分量では，その調理作業で使用する食材の分量であり食材に合わせた単位となる．例えば，豚肉であれば (g)，きゅうりであれば（本）となる．使用リソースは，その調理作業で使用する調理器具などのリソースの集合である．調理時間は，その調理作業を実行するのに必要な時間である．この調理時間は，レシピに調理時間が記載されている場合はその時間，記載されていない場合は，ユーザが過去にその調理作業を実行するのに必要とした実際の時間である．種別は，その調理作業で使用したリソースの状態を表し，洗い物の発生判断に使用される．開始条件は，その調理作業を実行する際に，すでに終了しておかなければならない調理作業の集合であり，これを調理作業の前後関係と呼ぶ．直前調理作業は，調理手順において，その調理作業の直前に実行される必要のある調理作業の調理作業である．ここで直前に実行されるとは，調理作業同士の終了時刻と開始時刻が一致することをいい，これを調理作業の連続性と呼ぶ．

本研究では，リソースとして，**調理者，包丁，まな板，ボウル，電子レンジ，フライパン，鍋，コンロ，グリル**などを想定する（要素は加減可能）．これらリソースのそれぞれの数（同時に利用可能な数）をベクトル $\boldsymbol{RS} = (RS_1, RS_2, \ldots, RS_g)$ で表す．ここで g はリソース種類の数である．また調理作業ごとに，必要とされるリソースは定義されているものとする．例えば「切る」という調理作業では，{ 調理者，包丁，まな板 } を必要とする．一方「茹でる」という調理作業では，{ 鍋，コンロ } を必要とするが調理者を必要としない．これらのリソースに対して，ある時間において異なるリソースは並列使用可能であり，同じリソースに対してはリソース数までの並列使用が可能である．

調理者ごとの各調理作業に必要とする時間の違いを考慮するために，調理者ごとの調理作業に必要とする時間を記録するデータベースを作成する．このデータベースには，（調理者・調理作業）の組み合わせごとに過去の調理履歴からかかった時間を記録していく．履歴がない場合には，初期値として統計データ（他の調理者）の平均値を使用する．この値は調理終了後に実際に計測した時間をフィードバックし更新する．本データベースの値は調理をすればするほど調理者の実態に合ったものとなる．各作業にかかる時間は，関数 D_t(調理者，調理作業) で求められるものとする．

4.2 調理手順

最適調理手順作成問題の出力である調理手順は，RR のレシピに含まれる全調理作業のうち並列作業可能な集合の列として作成する．作成する調理手順を $S = \langle Q_1, Q_2, \ldots, Q_n \rangle$ とする．ここで集合 Q_i は，並列処理可能な作業の集合であり，n は逐次実行される作業集合の数である．また調理手順 S には，S の実行中に発生する洗い物と，使用したリソースを最後に全て洗う片付けを含める．ただし調理開始直後は，全てのリソースがすでに使用可能である状態である．

ここで作成する調理手順 S は次に示す 3 つの制約条件を全て満たさなければならない．

1 つ目の制約として，調理手順 S によって選択した料理をすべて完成させるために，S は RR の全レシピの全調理作業を含んでいる必要がある．この制約を以下の式で表す．

$$\bigcup_{R \in RR} R \subseteq \bigcup_{i=1}^{n} Q_i \quad (1)$$

2 つ目の制約として，調理手順内の全ての時刻において，使用されるリソース数が，使用可能なリソース数以下である必要がある．S の作業集合 $Q_i(1 \leq i \leq n)$ が実行される際に必要なリソース数を $(rs_{i,1}, rs_{i,2}, \ldots, rs_{i,g})$ とすると，この制約は以下の式で表すことができる．

$$\forall i(1 \leq i \leq n)\ \forall j(1 \leq j \leq g), rs_{i,j} \leq RS_j \quad (2)$$

where

$$rs_{i,j} = \sum_{r \in Q_i \wedge r \text{ uses } j\text{th resource}} 1$$

3 つ目の制約として，調理作業の前後関係を持つ全ての調理作業の組み合わせは，調理手順 S 内でもその前後順序を満たす必要がある．すなわち，開始条件を持つ調理作業 r の開始条件である調理作業集合 r_p の全ての要素の終了時刻は，r_p の開始時刻より前でなければならないこの制約を以下に示す．

$$\forall i(1 \leq i \leq p), FinishTime(r_i) \leq StartTime(r) \quad (3)$$

特に調理作業の連続性を持つ調理作業同士であれば，(3) は等号が成立しなければならない．ここで $StartTime(r)$，$FinishTime(r)$ はそれぞれ調理手順内での調理作業 r の開始時刻，終了時刻を求める関数である．

4.3 洗い物の発生と定義

料理にはある調理作業開始時に，調理器具を洗う必要がある場合が存在する．例えば，野菜を切ろうとした際に，それまでにまな板で肉を扱っていた場合は，切る前に包丁とまな板を洗う必要がある．このように調理器具を使用する前に洗い物が必要か不必要かを，本研究では種別によって判断する．種別では，選択したレシピ集合 RR の中で相互に使用したリソースをそのまま使用できる調理作業には，同一の種別を割り当てる．一方で，そのままでは相互に使用できない調理作業同士には，独立した種別を割り当てる．例えば，肉を切る調理作業と野菜を切る調理作業は，連続して行うにはまな板と包丁を洗う必要があるため，それぞれの調理作業に対して，meat，vegetable の種別を割り当てるなどである．よって本研究における洗い物作業の発生は，実行したい調理作業の種別と，そのリソースが直前に使用された時の調理作業の種別が異なる時に発生するとし，洗い物作業を実行することによって利用可能になるものとする．ここでこの洗い物にかかる時間は，リソースごとに定義された時間ベクトル $\boldsymbol{WT} = (WT_1, WT_2, \ldots, WT_g)$ をそれぞれ要するものとする．またこの洗い物を実行するには洗い物対象のリソースと調理者を要するものとする．

4.4 目的関数

作成した調理手順の終了までにかかる時間を $time(S)$ とする．本研究では，最適調理手順を調理時間が最短であることとしているため，目的関数は以下となる．

$$Minimize\ (time(S))\ s.t.\ (1) - (3) \quad (4)$$

5. 最適調理手順探索アルゴリズム

本章では，本研究で定式化した最適調理手順作成問題の最適調理手順探索アルゴリズムについて述べる．

5.1 最適調理手順探索アルゴリズムのベースとなる最適化アルゴリズム

最適調理手順作成問題の出力である調理手順は，4.2 節で述べた制約条件を全て満たす必要がある．よってある調理手順のスケジュールを考える際に，制約条件を満たさないとわかった時点でそれ以降がどのような順序であっても最適調理手順ではないため探索する必要はない．また調理手順ごとに最初からスケジューリングを行うのは効率が悪く，それまでスケジューリングしたことのある調理手順と一致している箇所までは，以前のスケジューリング結果を再利用し，差分のみをスケジューリングし直すことで計算時間の削減を目指す必要がある．よって本研究では，これらを実現できる最適化アルゴリズムである DF/IHS をベースとし，前処理部と限定操作に料理特有の洗い物の性質を取り入れる工夫を加えることで最適調理手順探索アルゴリズムを作成する．

5.2 DF/IHS 法

最適化アルゴリズム DF/IHS 法は，主に前処理と分木限定法による探索から構成される．DF/IHS 法の前処理段階では，2 章で述べた CP/MISF で採用されている優先度の高い順にタスク番号を付け替え，探索木の左側に CP/MISF 的に良い解を集める．これにより探索段階では DF/FIFO 方式で探索木の左から探索を行うことでヒューリスティック的に良い解から探索することを可能としている．この結果，初期解が CP/MISF の解となるので限定操作で枝刈りを行えるため，探索時間の大幅な短縮に繋がる．DF/IHS 法がどのように探索を進めていくかを示すために，図 1 のタスクグラフを 2 プロセッサに割り当てる際に DF/IHS 法で探索した際の部分探索木を図 3 に示す．図 3 中のガントチャートはその時点までのプロセッサに対する割り当て結果であり，上がプロセッサ 1，下がプロセッサ 2 である．ただし，タスクスケジューリング問題において，各タスクの処理時間が異なる場合，プロセッサへの割り当てが可能となった時点で割り当てるリストスケジューリングでは最適解を得られない可能性がある [19]．そこで DF/IHS 法では，分技時にプロセッサを強制的にアイドル状態にするよ

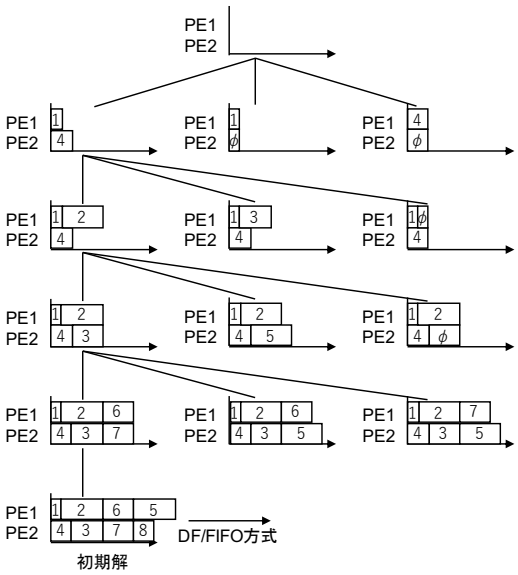


図 3: DF/IHS 法の部分探索木

うなタスク（図中の ϕ）も含めてノードを作成する．なお割り当てる際には，実際にはオーバヘッドが発生するが，簡単化のためここでは無視する．

5.3 最適調理手順探索アルゴリズム

最適調理手順探索アルゴリズムで使用する探索木を再帰的に作成する *Recursive* 関数と，調理作業の並列化と制約条件の確認を行う *Restruct* 関数の疑似コードをそれぞれ Algorithm1，Algorithm2 に示す．

Algorithm1 は，呼び出し後，最初の呼び出しに限り，O の初期化を行う．初期化する値は，どのような合計時間よりも必ず大きいような値である．続いて，現在作成途中のスケジュール P と最適調理手順とされている O を関数 *isBounded* を用いて枝刈りの判定を行う．関数 *isBounded* は，P に対して割り当てを続けることで O よりも良いスケジュールを得ることができるかを確認することにより，枝刈りをするかどうかを決定する関数である．枝刈りの基準については，次節で説明する．次に開始条件を満たしている調理作業集合 E の中から調理作業を 1 つ選択し，関数 *Restruct* の呼び出しを行う．ここで，F に含まれる調理作業のいずれかを直前調理作業とする調理作業が，E の中にある場合には，その調理作業の選択を優先する．それ以外の場合には FIFO 方式に従って選択する．これは，直前調理作業の関係のある調理作業同士は，連続して P に追加された方が並列化した後でも，調理作業のスケジュールが調理工程の連続性を満たす可能性が高く，最適調理手順の更新が期待できるからである．次に，選択された調理手順を P に追加し，並列化結果が，制約条件をすべて満たしているかを関数 *Restruct* によって確認する．その後，制約条件を満たすことが分かった時は，E，F の更新後に関数 *Recursive* を再度呼び出し，P への調理作業追加を

Algorithm 1: *Recursive* 関数

```
input : E : Executable Cooking task List
        F : Finished Cooking task List
        P : Provisional Schedule
output: O : Optimal Cooking Schedule
1  Initialize O only once ;
2  if isBounded (O,P) then
3      return
4  if E is not empty then
5      foreach element e in E do
6          isOk, P ← Restruct(e,P) ;
7          if isOk then
8              Remove(E, e) ;
9              Add(F,e) ;
10             AddExecutableTask(E,e) ;
11             Recursive(E, F, P) ;
12 else
13     AddCleanUpTask(P) ;
14     if time(P) < time(O) then
15         O ← P ;
```

Algorithm 2: *Recstruct* 関数

```
input : e : Cooking Process
        P : Provisional Allocation
output: isOk : Does P Satisfy Continuity
        P : Provisional Schedule
1  U ← AllocateResources(e) ;
2  W ← GetNeedWashingResources(U) ;
3  G ← GetNotUsingUserTime(P) ;
4  InsertWashingTask(P,W,G) ;
5  foreach element w in W do
6      AddWashingTask(P, w) ;
7  AddCookingTask(P,e) ;
8  isOk ← isSatisfyContinuity(P) ;
```

継続する．この時，P に追加した調理手順によって，開始条件を満たした調理手順は，関数 *AddExecutableTask* によって E に追加される．全ての調理作業が P に追加されたのち，P に洗われていないリソースを洗うタスクを関数 *AddCleanUpTask* を用いて追加する．ここで，洗い物の追加は，全ての調理作業の後に追加するのではなく，P 中に調理者と洗い物対象のリソースが使用されておらず，洗い物を行うことのできる隙間時間が存在する場合には，その隙間時間に追加する．最後に P と O を比較し，P が O より優れている場合には，P を O として更新する．

Algorithm2 では，まず P に追加する調理作業 e で使用するリソースを関数 *AllocationResources* によって割り当てる．割り当て候補となるリソースが複数ある場合は，以下の順で割り当てを行う．

(1) リソースが使用されておらず洗い物が不要

(2) リソースが使用されておらず洗い物が必要

(3) リソースの利用可能時刻が最も早い

次に，割り当てたリソースの中で洗い物が必要なリソースを関数 $GetNeedWashinResources$ によって取得し洗い物タスクを発生させる．発生した洗い物作業は，洗い物作業を行うことができる隙間時間 G がスケジュール中に存在する場合は，関数 $InsertWashingTask$ によって G に挿入し，挿入できない場合は関数 $AddWashingTask$ によって前詰で並列化中の調理手順スケジュールに追加する．全ての洗い物作業を追加後，関数 $AddCookingTask$ によって P に前詰で e の追加を行う．最後に，レシピの連続性を満たすかの判定を関数 $isSatisfyContinuity$ によって行う．

5.4 前処理手法

Algorithm1 の呼び出し前に，E に対して次のどちらかの前処理を行う．

5.4.1 手法 1：CP 長による並び替え

E に含まれる各調理作業の CP 長を求め，CP 長の大きい順に並べ替える．この手法は DF/IHS 法でも採用されている方法である．調理作業においてクリティカルパスが長い調理作業は，その調理作業後に多くの調理作業が存在するか，多くの時間を必要とする調理作業が控えている．多くの時間を必要とする調理作業が控えている場合，あるリソースを占有する時間が長くなり，そのリソースを使用するほかの調理作業を実行できなくなってしまう．一方で，占有されていないリソースを使用する調理作業とは並列実行可能であるため，多くの時間を必要とする調理作業を先に割り当てることは，並列実行できる調理作業の候補が多くなる．最適調理手順の探索おいてこの操作は，精度のよい初期解を得ることが期待できる．

5.4.2 手法 2：後続の調理作業による並び替え

与えられたレシピのタスクグラフにおいて，調理者を必要としない調理作業に到達可能な調理作業に，調理者を必要としない時間を付与する．その後，E を付与した値順に並び変える．調理作業において調理者を必要とする調理作業は必ず存在し，また洗い物にも調理者を必要とする．調理者を必要としない調理作業と調理者を必要とする調理作業を同時に実行することにより，調理時間の短縮が可能なことから，調理作業を必要としない調理作業を早い段階で P に割り当てることで並列実行できる調理作業の候補が多くなる．この操作も，CP 長による並び替え同様，精度のよい初期解を得ることが期待できる．

5.5 最適調理手順探索中の枝刈り

最適調理手順の探索の際には，以下の 2 つの手法を想定する．枝刈りは，Algorithm1 中の関数 $isBounded$ によって行われる．

5.5.1 手法 1：CP 長を用いた枝刈り

P に割り当てていない調理作業の CP 長は，洗い物を除き P に追加で必要とする時間となる．また P の割り当て時点で，片付けの際に洗い物が必要なリソースもわかっていることから，P の割り当てで既に必要としている時間に，洗い物時間と CP 長を加算した調理時間が O よりも優れていないときに枝刈りを行う．ただし，CP 長となるエッジ上に調理者を必要としない調理作業がある場合には，その隙間時間に洗い物を挿入できる場合があるので，隙間時間を考慮し洗い物時間は求める．

5.5.2 手法 2：リソースの残り使用時間を用いた枝刈り

調理作業中には，並列化できない調理作業が存在する．例えば，調理者が一人の場合，調理者を必要とする調理作業同士は，並列実行することができない．このようにリソースの制約によって並列実行できない調理工程の合計時間は，調理手順を作成する際にも最低でも必要となる。また手法 1 と同様，P の割り当て時点で，片付けの際に洗い物が必要なリソースもわかっていることから，P の割り当てで既に必要としている時間に，P に割り当てられていない調理作業に含まれている全てのリソースごとの合計時間の最大値と洗い物時間を加算した調理時間が O よりも優れていないときに枝刈りを行う．

6. アルゴリズムの評価

本章では，最適調理手順探索アルゴリズムの評価を行う．

6.1 実験概要

最適調理手順探索アルゴリズムの評価のために，実際のレシピをもとに複数のレシピを作成し，そのレシピを組み合わせた一般的な 1 食分の献立を作成する．作成した 2 つの献立の情報を表 1，2 に示す．

献立 1 は { 鯖の塩焼き，味噌汁，卵焼き }，献立 2 は { ハンバーグ，味噌汁，チョレギサラダ } を想定している．開始条件，直前調理作業は，同一レシピ内の作業番号を表しており，対象となる調理作業が存在しない場合は $NULL$ としている．

評価に用いる最適調理手順作成アルゴリズムは，Python 言語にて記述し，CPU：Intel Core i5 1.8GHz，メモリ：8GB，OS：MacOS10.14.6 のスペックを持つマシン上で実行した．

本稿では，次の 3 点について確認しアルゴリズムの有用性を示す．

1 点目は，与えられたレシピに対して，合計調理時間を短縮できるスケジュールを作成できているかについて確認する．最適調理手順作成アルゴリズムを実行した結果得られる最適調理手順スケジュールとの合計調理時間の比較対象

表 1: 献立 1 のレシピ情報

レシピ	調理作業	使用リソース	調理時間	種別	開始条件	直前調理作業
レシピ 1 鯖の塩焼き	1	grill	240	1	NULL	NULL
	2	user, grill	60	1	1	1
	3	grill	360	1	2	2
	4	user, grill	30	1	3	3
	5	grill	360	1	4	4
レシピ 2 味噌汁	1	bowl	300	NULL	NULL	NULL
	2	user, knife	60	2	NULL	NULL
	3	pot, stove	300	2	1, 2	NULL
	4	user, pot, stove	60	2	3	3
レシピ 3 卵焼き	1	user, bowl	120	3	NULL	NULL
	2	user, pan, stove	420	3	NULL	1

表 2: 献立 2 のレシピ情報

レシピ	調理作業	使用リソース	調理時間	種別	開始条件	直前調理作業
レシピ 1 ハンバーグ	1	user, knife, board	240	vegetable	NULL	NULL
	2	bowl, microwave	360	1	1	NULL
	3	bowl	600	1	2	2
	4	user, bowl	360	1	3	NULL
	5	user, pan, stove	240	1	4	NULL
	6	user, pan, stove	600	1	5	5
	7	user, pan, stove	120	1	6	6
レシピ 2 味噌汁	1	bowl	300	NULL	NULL	NULL
	2	user, knife	60	2	NULL	NULL
	3	pot, stove	300	2	1, 2	NULL
	4	user, pot, stove	60	2	3	3
レシピ 3 チョレギサラダ	1	user	60	NULL	NULL	NULL
	2	user, knife, board	60	vegetable	NULL	NULL
	3	user, bowl	120	3	NULL	NULL
	4	user, bowl	60	3	1, 2, 3	NULL

として，献立が与えられた時に，調理者自身が考えられるであろう調理手順スケジュールの 1 例を手動で作成する．手動で作成する調理手順スケジュールは，レシピの調理手順通りに行った際に，調理者を使用していない隙間時間を目視で確認し，隙間時間があった場合に，他のレシピの実行可能な調理手順を挿入することで作成する．本稿では，この 2 つのスケジュールと合計調理時間を比較することにより，最適調理手順作成アルゴリズムの実行によって，合計調理時間を短縮できているかどうかを確認する．

2 点目は，本研究で想定している前処理手法と枝刈り法それぞれ 2 つの各組み合わせのうち，どの組み合わせが最も最適調理手順探索アルゴリズムに適しているかを確認する．検証に用いる比較対象として，アルゴリズムを実行してから実行が終了するまでの実行時間と，アルゴリズムを実行してから暫定解が最適調理手順スケジュールの総調理時間となるまでの収束時間を定義し比較する．

3 点目は，本アルゴリズムが最適調理手順の探索の際に探索する必要がある必要探索数の増加に対して，効率よく探索を行うことができるかを実行時間・収束時間を用いて確認する．ここで必要探索数は，献立内の調理作業をレシピ内の制約条件を満たすように並び替えた場合の数であり，枝刈りせずに探索した場合の探索数である．

6.2 スケジュールと合計調理時間の比較

最適調理手順作成アルゴリズムの実行によって，合計調理時間を短縮できているかどうかを確認するために，献立 2 に対して手動で作成したスケジュールと，最適調理手順探索アルゴリズムを実行することによって得られる最適調理手順スケジュールをそれぞれ図 4，5 に示す．

与えられるリソースは，図中の縦軸のリソース数を想定している．スケジュールは，横軸が時間，縦軸がリソース名となっている．各調理作業は，{ レシピ番号-作業番号 } となっており，例えば献立 2 のハンバーグの調理作業 6 は，1 – 6 となっている．図内の横棒は，それぞれ各レシピに含まれている調理作業を表しており，同じ色の横棒は 1 つの調理作業を表している．先ほどの 1 – 6 の調理作業であれば，図 4，5 どちらの場合でも，1800 – 2400 の時間に {user, pan2, stove2} が割り当てられ実行するスケジュールとなっている．洗い物作業は，図中の Washing で表されており，調理者と洗い物の対象となるリソースを必要としていることがわかる．このスケジュールを見ることで，どの時間にどの調理作業をどのリソースを用いて実行しているかを確認できる．図 4 中には，調理者を使用していない隙間時間が存在するが，その隙間時間に別の調理作業を挿入すると，レシピの制約条件を満たすことができなくなるため挿入できない．また図 5 中にも隙間時間は存在するが，その隙間時間に別の調理手順を挿入すると，最適調理手順よりも調理時間が増加してしまうため挿入されていない．図 4，5 を比較すると，2 つのスケジュールでおよそ 480 秒ほど調理時間が差がある．これは，最適調理手順スケジュールでは手動で作成したスケジュールに比べて，洗い物回数が少なく，調理作業同士を効率的に並列化したスケジュールであるためである．よって最適調理手順探索アルゴリズムによって，調理者にとって効率よく合計調理時間を短縮できるシュケジュールを求められていることがわかる．

6.3 前処理と枝刈り法の組み合わせの比較

前処理手法，枝刈り手法それぞれ 2 つの各組み合わせによって解の探索がどのように変化するかを比較するために，献立 1，献立 2 の最適調理手順の探索における実行時間と収束時間を表 3 に示す．

表 3 の実行時間を比較すると，献立 1 に対しては combination2，3 が，献立 2 に対しては combination3 が最も優れているという結果になった．収束時間を比較すると，献立 1 に対しては combination3 が，献立 2 に対しては combination1 が最も優れているという結果になった．一方で，combination4 については，実行時間・収束時間共にあまり良い結果は得られなかった．前処理手法，枝刈り法についてそれぞれ比較すると，前処理手法については，若干ではあるが手法 2 を行った時に良い結果を得ることができ，枝刈り手法については，枝刈り手法 1 が優れていることがわかる．特に枝刈り手法では献立 2 において，手法 1 を用いている combination1，3 と手法 2 を用いている combination2，4 をそれぞれ比較すると，手法 1 が手法 2 よりも完全に優れていることがわかる．これは，献立で選

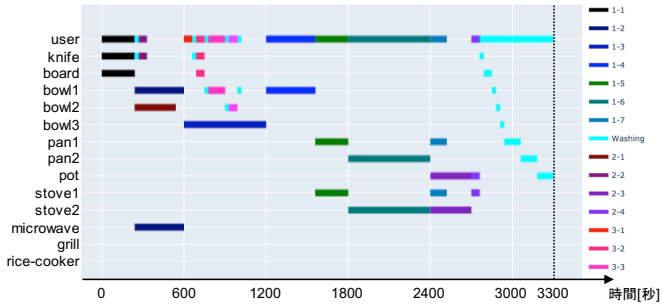


図 4: 手動で作成したスケジュール

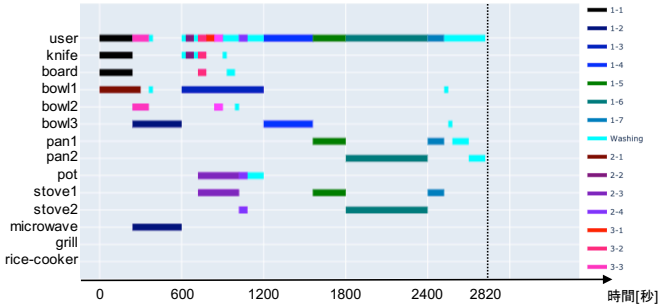


図 5: 最適調理手順スケジュール

表 3: 最適調理手順探索アルゴリズムの実行時間と収束時間

	combination1 前処理手法 1+枝刈り手法 1		combination2 前処理手法 1+枝刈り手法 2		combination3 前処理手法 2+枝刈り手法 1		combinaiton4 前処理手法 2+枝刈り手法 2	
	実行時間 [秒]	収束時間 [秒]	実行時間 [秒]	収束時間 [秒]	実行時間 [秒]	収束時間 [秒]	実行時間 [秒]	収束時間 [秒]
献立 1	6.09	1.61	5.93	1.61	5.93	1.33	6.93	1.52
献立 2	1.33×10^3	3.00×10	2.37×10^3	4.85×10	1.26×10^3	3.14×10	2.35×10^3	4.7×10

択したレシピの調理作業の中でリソースの被っている調理作業が少なかったため，枝刈り手法 2 では，枝切りを行うことが難しかったためであると考えられる．実行時間と収束時間について比較すると献立にかかわらず大きな差があることがわかる．これは，最適調理手順となるスケジュールが複数あるために，限定操作で調理手順の探索を停止することができないためであると考えられる．実際に献立 2 については，片付けで発生する洗い物を追加するまで限定操作の対象とならならず，最短調理時間と同じ調理時間となるスケジュールが最低でも 3777 通りあることを確認した．これらの結果から，最適調理手順探索アルゴリズムにおいて，前処理手法 2 と枝刈り手法 1 を組み合わせた combination3 が，最も適していることがわかる．

6.4 解の収束の様子

解の収束の様子を図 6，7 に示す．図 6，7 は，横軸が実行時間，縦軸がその実行時間において最適調理手順であるとされる暫定スケジュールの調理時間を表している．手動で作成したスケジュールにおける調理時間と，最適調理手順探索アルゴリズムの実行によって得られる初期解を比較すると，献立 1 では手動で作成したスケジュールの調理時間が，献立 2 では最適調理手順探索アルゴリズムの初期解の調理時間が優れていることがわかる．

献立 1 で最適調理手順探索アルゴリズムの初期解が手動よりも劣っているのは，献立 1 のレシピを目視で確認した際に，調理者を必要としていない調理作業時間が多くあり，またレシピの連続性が単純であったため，調理作業を挿入しやすい献立であり，手動で作成したスケジュールの精度がよかったためである．しかし最適調理手順探索アルゴリズムを実行後 0.1 秒未満で，手動でのスケジュールよりも優れているスケジュールは見つかっており，その後急速に収束に向かっている．

表 4: 献立の探索に関わる情報

	総調理作業数	必要探索数	手動作成調理手順 [分]	最適調理手順 [分]
献立 1	11	1.39×10^4	34.0	25.5
献立 2	16	5.41×10^6	55.0	47.0

献立 2 については，レシピ中に調理者を必要としない調理作業は存在するが，レシピ内の連続性が多く存在し複雑であったため，手動で考えたスケジュールよりも最適調理手順探索アルゴリズムの初期解がよくなったと考えられる．次に，必要探索数の増加に対して，効率よく探索できているかについて，各献立の探索に関わる情報を表 4 に示す．表 3，4 より，必要探索数が約 400 倍となった時に，枝刈り手法 1 を用いている組み合わせについては実行時間の増加をおよそ 200 倍に抑えられている．また献立 2 については，実行時間と収束時間に大きな差があるが，図 7 からもわかるように開始 10 秒程度で，手動で考えたものから 7 分短縮できるスケジュールを求めることができていることから，このアルゴリズムの有用性を確認できる．

7. 結論

本稿では，複数レシピで並行調理する際の調理環境に応じた最適調理手順を提示するために，調理手順をタスクスケジューリング問題として捉え，最適調理手順探索アルゴリズを提案した．評価実験の結果，提案した最適調理手順探索アルゴリズムによって，想定した献立に対する最適調理手順を求めることができることを確認した．また探索必要数が 400 倍に増加した時に，実行時間の増加を 200 倍に抑えられていることも確認した．さらに，初期解に対して 10 秒程度で急激に収束に向かっており，調理時間を短縮する調理手順を求める点については，レシピ数を増加させた場合にも本アルゴリズムは有効であることが考えられる．

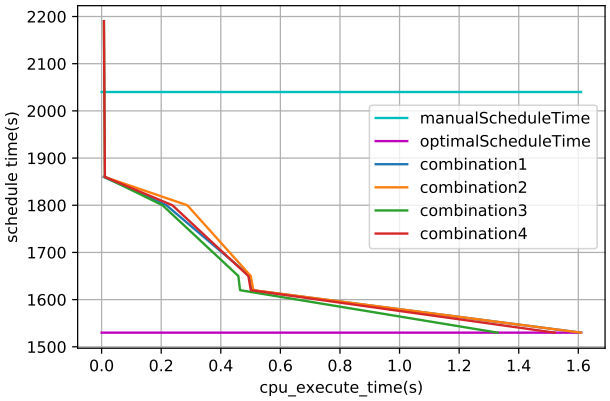


図 6: 解の収束の様子 (献立 1)

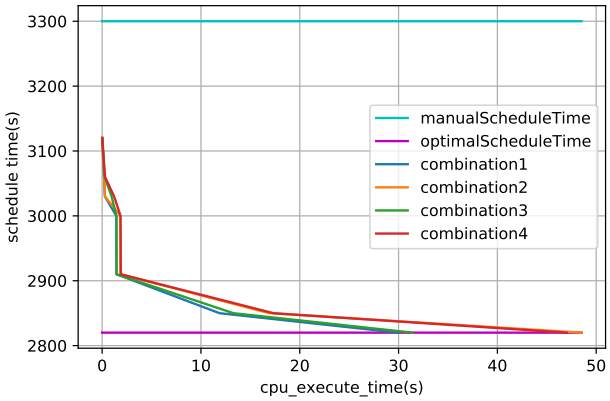


図 7: 解の収束の様子 (献立 2)

ただし実時間内に最適調理手順を求めるためには，今後実行時間を短縮させるための工夫がさらに必要であることが分かった．

今後は，手動で作成した調理手順の合計時間にに対して，ユーザがどの程度改善された調理手順を求めているのかや，求めた調理手順をどのようにユーザに対して提示するかを検討する．また最適調理手順通りに調理者が料理を行った場合，想定した調理手順とどのような変化が生じるのかについて検証を行う．

参考文献

[1] 首相官邸：人生 100 年時代構想会議. available from `http://www.kantei.go.jp/jp/singi/jinsei100nen/` (accessed 2020-03-18).

[2] 厚生労働省：平成 29 年版厚生労働白書 －社会保障と経済成長－. available from `https://www.mhlw.go.jp/wp/hakusyo/kousei/17/dl/all.pdf` (accessed 2020-03-18).

[3] クックパッド株式会社. available from `https://cookpad.com/` (accessed 2020-03-18).

[4] allrecipes.com，Inc. available from `https://www.allrecipes.com/` (accessed 2020-03-18).

[5] Hironori Kasahara and Seinosuke Narita. Practical multiprocessor scheduling algorithms for efficient parallel processing. *IEEE Transactions on Computers*, No. 11, pp. 1023–1029, 1984.

[6] Yukiko Matsushima and Nobuo Funabiki. Practices of cooking-step scheduling algorithm for homemade cooking. In *2015 IIAI 4th International Congress on Advanced Applied Informatics*, pp. 500–505. IEEE, 2015.

[7] 山吹卓矢, 小野典彦, 永田裕一. 同卓スケジューリング問題のモデル化とその動的スケジューリング. 計測自動制御学会論文集, Vol. 54, No. 3, pp. 346–356, 2018.

[8] Edward Grady Coffman and John L Bruno. *Computer and job-shop scheduling theory*. John Wiley & Sons, 1976.

[9] Te C Hu. Parallel sequencing and assembly line problems. *Operations research*, Vol. 9, No. 6, pp. 841–848, 1961.

[10] 笠原博徳, 伊藤敦, 田中久充, 伊藤敬介. 実行時間最小マルチプロセッサスケジューリング問題に対する並列最適化アルゴリズム. 電子情報通信学会論文誌 D, Vol. 74, No. 11, pp. 755–764, 1991.

[11] Eduardo B Fernandez and Bertram Bussell. Bounds on the number of processors and time for multiprocessor optimal schedules. *IEEE Transactions on Computers*, Vol. 100, No. 8, pp. 745–751, 1973.

[12] 中村あすか, 前川仁孝ほか. タスクスケジューリング問題の厳密解求解における探索ノード数削減アルゴリズム. 情報処理学会論文誌プログラミング (PRO), Vol. 7, No. 1, pp. 1–9, 2014.

[13] 松瀬弘明, 中村あすか, 富永浩文, 前川仁孝ほか. タスクスケジューリング問題における df/ihs 法のハッシュテーブルを用いた探索ノード数削減. 第 78 回全国大会講演論文集, Vol. 2016, No. 1, pp. 197–198, 2016.

[14] 澁谷智則, 栗田浩一, 甲斐宗徳ほか. B-017 通信を考慮したタスクスケジューリング問題のための並列分枝限定法とその評価 (b 分野: ソフトウェア, 一般論文). 情報科学技術フォーラム講演論文集, Vol. 13, No. 1, pp. 149–154, 2014.

[15] Masahiko Utsunomiya, Ryuji Shioda, and Munenori Kai. Heuristic search based on branch and bound method for task scheduling considering communication overhead. In *Proceedings of 2011 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*, pp. 256–261. IEEE, 2011.

[16] 澁谷知則, 甲斐宗徳. 通信遅延を考慮したタスクスケジューリング問題の並列解法: サブタスクグラフ最適スケジューリングによる探索効率の改善. 2016.

[17] 長谷川幹, 甲斐宗徳. タスクグラフの階層的マクロタスク化とそのタスクスケジューリング手法の提案. 2018.

[18] Michael R Garey. Computers and intractability: A guide to the theory of np-completeness. *Revista Da Escola De Enfermagem Da USP*, Vol. 44, No. 2, p. 340, 1979.

[19] Chittoor V Ramamoorthy, K Mani Chandy, and Mario J Gonzalez. Optimal scheduling strategies in a multiprocessor system. *IEEE Transactions on Computers*, Vol. 100, No. 2, pp. 137–146, 1972.