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

山 吹 卓 矢*・小 野 典 彦**・永 田 裕 一**

Modeling of the Simultaneous Table Scheduling Problem and Dynamic Scheduling for this Problem

Takuya YAMABUKI*, Norihiko ONO** and Yuichi NAGATA**

When a group of people orders dishes at a restaurant, in many cases it is desirable to realize “simultaneous offering of dishes” – to carry all dishes as simultaneously as possible. However, it is difficult to manage the cooking schedule so that the dishes ordered by the same group are completed at the same time in a situation where many other dishes are ordered by other groups. In this paper, we develop a scheduling system to generate a cooking schedule to realize simultaneous offering of dishes in accordance to the current situation on the orders from all groups. We also propose a dynamic scheduling method to generate a schedule in dynamic environments where orders come in one after another.

Key Words: cooking scheduling, modeling, dynamic scheduling

1. はじめに

ファミリーレストランなどの飲食店において複数人のグループが料理を注文するとき、大抵、それらの料理をできるだけ同時に提供する「同時同卓提供」が望まれる。しかし、多くの注文が殺到する昼時や夕方時には、料理人が注文内容や調理状況を判断して、同時同卓提供を実現するような調理順序を決定・管理することは困難である。そこで、本論文では飲食店において同時同卓提供を実現するための調理業務支援システムを開発することを目標とする。具体的には、すべての注文内容を元に、同時同卓提供を実現するための料理人ごとの調理スケジュールを生成する方法を開発する。

調理業務支援システムに関連する先行研究はあまり多くないが、一例として、ファミリーレストランにおける調理スケジュールを構築する研究¹⁾が挙げられる。この研究ではオープン調理を1人の料理人で行なう場合の調理スケジュールの最適化を数理計画問題として定式化している。スケジュールの評価は、料理の完了時間の総和、最終完了時刻、同時提供料理の提供時間差、連続作業の充足度合い、により行なわれている。また、野中ら²⁾は居酒屋などの飲食店において、顧客

が注文時に料理の提供順序の希望を出し、それをもとに調理スケジュールを立案するアルゴリズムを提案しているが、単純な例のみを扱っている。

本論文では注文された全料理を複数の料理人で調理を行なう場合の調理スケジュールを対象とする。また、調理工程における、並列調理（複数の調理工程の並列処理）や同時調理（同じ調理工程の一括処理）を考慮したスケジューリングを行なう。これにより、広いクラスの飲食店に対応可能な調理スケジュールのモデル化が可能である。本論文ではモデル化した問題を「同卓スケジューリング問題」と呼ぶことにする。また、注文が次々と入ってくるような動的環境において、料理の同時同卓提供を実現する調理スケジュールを作成するための動的スケジューリングを行なう手法も提案する。本研究では、同卓スケジューリング問題のモデル化に重点を置き、モデル化された問題の解法には比較的単純なシミュレーテッドアニーリング³⁾を用いた手法を提案する。

同卓スケジューリング問題は生産スケジューリング問題の一種であるが、その中でも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）を拡張した問題と捉えることができる。RCPSPは複数のジョブをいくつかの機械で処理する場合の最適スケジュールを見つける問題で、フローショップ問題やジョブショップ問題などをはじめとする多くのスケジューリング問題を定式化できる汎用性の高いモデルであることが知られている⁴⁾。RCPSPの基本形はつぎのように述べられる：(i) ジョブはいくつかの工程から成り、工程間には先行関係がある。(ii) 複数種類の機械があり、各機械は処理能力の上限値をもつ。各機械は同時に複数のジョブ工程を処理できるが、負荷の合計値が処理能力の上限値を越えてはならない。

* 徳島大学大学院先端技術科学教育部 徳島市南常三島町 2-1

** 徳島大学大学院社会産業理工学研究部 徳島市南常三島町 2-1

* Graduate School of Advanced Technology and Science, Tokushima University, 2-1 Minamijosanjima-cho, Tokushima

** Graduate School of Technology, Industrial and Social Sciences, Tokushima University, 2-1 Minamijosanjima-cho, Tokushima

(Received April 24, 2017)

(Revised September 3, 2017)

(iii) 各ジョブ工程は 1 つの機械 (ジョブを処理可能な機械から 1 つ選択) で処理されるが, その際決められた時間と負荷を要する (どの機械で処理するかで異なる値をもつ). (iv) 目的関数は最初の工程を開始してからすべての工程が完了するまでの時間である. 同卓スケジューリング問題は, 料理をジョブ, 料理人を機械と対応させることで, RCPSP と類似した問題となるが, 調理スケジューリングに内在する固有の制約条件を考慮する必要がある. 目的関数についても料理の同時同卓提供および料理スケジューリングに固有の評価項目を考慮する必要がある. RCPSP に対しては, より現実的で複雑な制約や目的関数を導入したさまざまな拡張モデルも提案されているが, 同卓スケジューリング問題もそのような拡張の一つと捉えることができる. RCPSP の拡張モデルについてはサーベイ文献⁵⁾などを参照されたい.

RCPSP は NP 困難問題として知られ, 問題サイズが大きくなると短時間で厳密解を発見することが困難となるため, メタヒューリスティクスを用いた近似解法が広く開発されている⁶⁾. 特に, RCPSP の特殊ケールであるジョブショップスケジューリング問題 (Job shop scheduling problem, JSP) に対しては, 多くの解法が考案されており, 問題の構造を利用した高度で高性能な近似解法が考案されている^{7), 8)}. 一方, 多様化する生産スケジューリング問題の個々の問題に対して専用ソルバーを作るのではなく, モデリング言語を用いて問題をモデル化し, 汎用ソルバーで問題を解く方法も開発されている. たとえば数理計画問題として問題を定式化できる場合は混合整数計画ソルバーなどを用いて厳密解を求めることができる. ただし, 生産スケジューリングのような工程間の順序を決定するような問題に対しては, 大きなサイズの問題は解けないことが知られている. そこで, 野々部ら⁹⁾は拡張 RCPSP に特化した汎用ソルバー (OptSeq) を開発している. このソルバーではなるべく広い範囲の生産スケジューリング問題を定式化できるように拡張 RCPSP のモデル化を行っており, モデリング言語で記述された問題の (近似) 解はタブー探索を中心とするメタヒューリスティクスを用いて求められる. しかし, 問題のモデル化能力には限界があり, 同卓スケジューリング問題に内在する固有の制約条件や目的関数を考慮するためには, 未だに別途専用ソルバーを開発する必要があるのが現状といえる.

本論文の構成は以下のとおりである. 2 章では同卓スケジューリング問題のモデル化を行なう. 3 章ではモデル化した問題の解法を提案する. 4 章では計算機実験を行ない, 同卓スケジューリング問題のモデル化の妥当性について検証する. 5 章はまとめである.

2. 同卓スケジューリング問題のモデル化

本章では同卓スケジューリング問題の定式化を行なう. 2.1 節では問題設定を述べる. 2.2~2.4 節では, 与えられた注文に対して調理スケジュールを作成する “静的” な同卓スケジューリング問題の定式化を行なう. ただし, これらの節で

定義される定数や変数の具体例は 2.5 節で一括して示すことにする. 2.6 節ではスケジュールを生成するために必要なアルゴリズムを説明する. 2.7 節ではスケジュールを評価する評価関数を定義する. 2.8 節では注文票が順次到着する場合の “動的” な同卓スケジューリング問題の定式化を行なう.

2.1 問題設定

同卓スケジューリング問題を構成する基本要素, 前提となる事項, 最適化の目的の概要を説明する. これらの事項はつぎのように述べられる.

- 客からの注文が注文票に記載され, 料理人は注文票を受けて料理を作る.
- 各料理の調理工程は複数の作業 (プロセス) に分解される.
- プロセスごとに固有の調理場や作業時間などの条件がある.
- 各プロセスには一人ずつ料理人が割り当てられる.
- 同じグループからの注文をなるべく早くかつ同時に提供するように, 各料理の各プロセスをどの料理人がどのタイミングで処理するかを決定するのが目的である.

2.2 定数

同卓スケジューリング問題を定式化するために必要な定数を定義する. 定式化において, 集合 A の要素数は n_A のように表現する. ただし, 表記が煩雑なる場合は, 特に断りなく $|A|$ と表わすことにする.

2.2.1 料理, 料理人, 調理場, 注文票

同卓スケジューリング問題を定式化するための定数として, 料理, 料理人, 調理場, 注文票に関するものを定義する.

料理 提供可能な料理に固有の ID 番号を付けるものとし, 料理 ID 集合を以下のように定義する.

$$M = \{1, 2, \dots, n_M\} \quad (1)$$

各料理 $m \in M$ にはその料理を調理するための手順が定義されている (後出 Table 2 参照, 詳細は 2.2.2 項で述べる).

料理人 レストランで働いている料理人に固有の ID 番号を付けるものとし, 料理人 ID 集合を以下のように定義する.

$$W = \{1, 2, \dots, n_W\} \quad (2)$$

各料理人 $w \in W$ にはスキルなどに応じた属性値が設定される (後出 Table 4 参照, 詳細は 2.2.3 項で述べる).

調理場 厨房は用途に応じた異なる調理場にわかれている. 調理場には固有の ID 番号を付けるものとし, 調理場 ID 集合を以下のように定義する.

$$Z = \{1, 2, \dots, n_Z\} \quad (3)$$

各調理場 $z \in Z$ には, 担当する料理人が決まっている (後出 Table 5 参照).

注文票 注文票は同じテーブル (または同一グループ) が注文した料理内容が記載される. 注文票には固有の ID 番号を付けるものとし, 調理場 ID 集合を以下のように定義する.

$$I = \{1, 2, \dots, n_I\} \quad (4)$$

各注文票 $i \in I$ には同じテーブルで注文された料理 ID が記入される．各注文票に並ぶ料理 ID の順番を注文明細番号と呼ぶことにする（後出 Table 1 参照）．ただし，同じ料理が複数注文されている場合でも，個々に異なる注文明細番号が割り振られるものとする．ここで， $\text{menuID}(i, j)$ を注文票 i の注文明細番号が j 番目の料理 ID を返す関数と定義する．

2.2.2 調理手順，プロセス，オペレーション

各料理 $m \in M$ の調理工程を作業内容に応じて細分化し，細分化された作業内容をプロセスと呼ぶことにする．全料理の調理工程に含まれるプロセスをまとめて，各プロセスに自然数で表わされる ID 番号を付けるものとする．プロセス ID 集合を以下のように定義する．

$$P = \{1, 2, \dots, n_P\} \quad (5)$$

プロセスはつぎの 5 つの属性をもち，プロセスごとに属性値が設定されているものとする（後出 Table 3 参照）．

並列可能性 プロセスには 1（並列可能），0（並列不可能）の属性値が設定される．並列可能プロセスは料理人が付きっきりで調理する必要がなく，1 人の料理人が他プロセスと並列に調理可能である．これを並列調理と呼ぶ．一方，並列不可プロセスは料理人が付きっきりで調理する必要があるため，1 人の料理人はほかの並列不可プロセスと並列調理することができない．すなわち，1 人の料理人は並列不可プロセスと複数の並列可能プロセスを並列に調理することができるが，並列不可プロセスは高々 1 つしか処理することができない．また，料理人ごとに最大可能な並列処理数が決められている（料理人の属性で指定）．

指定調理場 プロセスを処理するための調理場所．調理場 ID で指定．

最大同時調理数 同一プロセスを複数処理する場合，1 つのプロセスを処理するのと同じ時間でそれらのプロセスを同時に調理することが可能な場合がある．このような調理を同時調理と呼ぶ．最大同時調理数はプロセスごとに設定される同時調理可能なプロセス数の最大値である．

必要スキル プロセスを処理するために必要な料理人のスキル．スキル ID（1 つ）で指定．

調理時間 プロセスを処理するのに要する時間（秒）．段取り時間はプロセスごとに一定と仮定してここに含めるものとする．

各料理 $m \in M$ の調理工程はプロセスの遷移図で定義する．

Fig. 1 に料理「ミートソーススパゲティ」と「ナポリタンスパゲティ」それぞれの調理工程の遷移図を示す（ここでは点線の矢印は無視する）．遷移図はグラフ構造をもち，ノードはプロセス（ノード中の数字はプロセス ID）を表わす．また，ノード s と t は料理の開始と完成を表わすダミーノードを表わす．グラフ中の有向枝（実線）はプロセス間の先行関係を示している．すなわち，各プロセスはそこに入射するすべての有向枝の始点プロセスが完了した後でなければ，処理を開始することができない．また，いくつかのプロセス間には順

Fig. 1 Example of a disjunctive graph (solid arcs represents the precedence relation between the processes of the two dishes): numbers in the circles represent process IDs, and the numbers above the circles represent procedure IDs

序関係がなく，並列調理が可能である．便宜上，各料理ごとに，それを構成するプロセスに手順 ID を自然数で割り振るものとする（その順番で調理可能なように割り振る）．Fig. 1 の遷移図では，手順 ID はノードの左肩の数字で表わされている． $\text{processID}(m, k)$ を料理 ID = m と手順 ID = k を指定したときに，対応するプロセス p を返す関数と定義する．

調理順序を考える上で，あるプロセスがどの注文票のどの料理のプロセスなのか区別する必要がある．そこで，（注文票 ID，注文明細番号，手順 ID）の 3 つ組みの数字をオペレーション ID と呼ぶことにする．すなわちオペレーション ID である (i, j, k) の値が与えられ， $m = \text{menuID}(i, j)$ より料理 ID が得られ， $p = \text{processID}(m, k)$ より該当するプロセス ID が得られる．また，オペレーション ID (i, j, k) が指定するプロセスのことを，単にオペレーション (i, j, k) と呼ぶことにする．

2.2.3 料理人の属性設定

料理人ごとに店舗での経験が異なり，プロセスの調理時間が異なることが考えられる．また，料理人ごとに調理スキルが異なるため，調理可能なプロセスが限られることもある．これらの事情を考慮して，料理人に以下の属性を設定する（後出 Fig. 4 の例参照）．

所持スキル 料理人ごとに，各調理スキル（スキル ID で指定）を所持しているかどうかを指定．1（所持），0（非所持）の属性値で指定．

最大並列調理数 料理人ごとに，並列可能調理が可能なプロセス数の最大数を設定．

遅延倍率 料理人ごとに，プロセスの処理にかかる時間の遅延倍率（プロセスごとに設定された処理時間に乗ずる値）を設定．どのプロセスに対しても一律同じ値とする．

担当調理場 料理人ごとに設定される担当調理場．調理場 ID で指定（複数可能）．各料理人は指定された担当調理場で作業を行なうことを基本とするが，ヘルプとして別の担当調

理場で調理することも可能である。

2.3 調理順序の表現 (変数)

全料理人は注文された全料理を完成させるために、それらの料理の調理工程に含まれる全プロセスを実行しなければならない。同卓スケジューリング問題の解は、料理人がそれらのプロセスを処理する順番で表わされる。全料理人の調理順序 (変数) は次式で表わすものとする。

$$G = \{G_1, G_2, \dots, G_{n_W}\} \quad (6)$$

G_w ($w \in W$) は料理人 w の調理順序で、以下のように定義される (具体例は 2.5 節参照)。

$$G_w = \langle g_{w,1}, g_{w,2}, \dots, g_{w,|G_w|} \rangle \quad (7)$$

ここで、 $g_{w,s}$ ($s = 1, 2, \dots, |G_w|$) は料理人 w が s 番目に処理するオペレーション ID である。

調理順序 G が実行可能であるためにはつぎの制約条件を満たす必要がある。

- (1) 注文された全料理に割り当てられているすべてのオペレーション ID は G に 1 回だけ現れる。
- (2) 各料理人 w は、割り当てられた調理順序 G_w に含まれるすべてのオペレーションに対し、それを実行する調理スキルをもっていなければならない。
- (3) G で指定される調理順序が料理ごとに設定されたプロセスの先行関係に矛盾しない (詳しくは 2.4 節で説明する)。

2.4 調理スケジュール

実行可能な調理順序 G が与えられたとき、 G を構成するすべてのオペレーション $g_{w,s}$ ($w = 1, \dots, n_W, s = 1, \dots, |G_w|$) の開始時刻を決定したものを G のスケジュールと呼ぶ。スケジュールは調理順序 G と各料理の調理手順に矛盾しない範囲で、最も早く調理を進めるとして計算する。このようにして決定されたオペレーションの開始時間を最早開始時刻と呼ぶ。オペレーション o の最早開始時刻を $t(o)$ 、最早完了時刻を $t^e(o) (= t(o) + \text{オペレーション } o \text{ の処理時間})$ と表わすことにする。

調理順序 G からスケジュールを求めるには、 G に含まれる料理を構成するオペレーションの先行関係を Fig. 1 に示すようなグラフで表現するのが便利である。このグラフ表現は Job shop scheduling problem (JSP) のスケジュールを求める際に用いられる選択グラフ¹⁰⁾と本質的に同じものである。選択グラフは 2.2.2 項で説明した料理の調理工程を表わす遷移図に料理人ごとの調理順序を表わす有効枝 (点線) を追加したものである。図の例では 2 つの料理「ミートソーススパゲティ」と「ナポリタンスパゲティ」を 2 人の料理人 1, 2 で調理する場合のオペレーションの先行関係を示している。さらに、選択グラフの有向枝には枝の始点オペレーションの処理時間が重みとして付与されるものとする (ダミーオペレーション s と t の調理時間は 0 とする)。なお、選択グラフ中に閉路が存在する場合は調理順序 G に矛盾が生じており (2.3

節の制約条件 (3) に違反)、 G は実行可能でない。

最初に単純な場合として、同時調理と並列調理を考慮しない場合のスケジュールを考える。各オペレーション o は、選択グラフにおいてオペレーション o を終点とするすべての有効枝の始点オペレーションが完了していれば開始することができる。このことを考慮すると、各オペレーション o の最早開始時刻 $t(o)$ は s から o に至る最長経路の長さとなる。よって、調理順序 G に含まれるすべてのオペレーション $g_{w,s}$ ($w = 1, 2, \dots, s = 1, 2, \dots, |G_w|$) の最早開始時刻 $t(g_{w,s})$ は、選択グラフ上で s から各オペレーションへ至る最長経路の長さを動的計画法を用いて計算することができる。

同時調理と並列調理を考慮した場合、料理人 w が調理順序 G_w に従いオペレーション $g_{w,s}$ を処理する際、オペレーション $g_{w,1}, \dots, g_{w,s-1}$ のいずれかの完了を待たずに開始できる場合がある。Fig. 2 に具体例を示す。これが可能となる具体的な条件については 2.6 節で説明する。

2.5 具体例

問題定数として定義した注文票 ID 集合 I (Table 1)、料理 ID 集合 M (Table 2)、料理人 ID 集合 W (Table 4)、調理場 ID 集合 Z (Table 5)、プロセス ID 集合 P (Table 3)、およびそれらの指す内容の一例を一括して示す。また、この例において、2 人の料理人の調理順序 $G = (G_1, G_2)$ が与えられた際に構成される選択グラフ (Fig. 1) とスケジュール (Fig. 2) の例を示す。 G_1 と G_2 は以下で与えるものとする。

$$G_1 = \langle (1, 1, 4), (1, 2, 2), (1, 1, 2), (1, 1, 3), (1, 2, 4) \rangle$$

$$G_2 = \langle (1, 2, 5), (1, 2, 1), (1, 1, 1), (1, 2, 3) \rangle$$

Fig. 2 A schedule corresponding to the disjunctive graph given in Fig. 1: numbers in brackets represent operation IDs, and numbers in the circles represent process IDs

Table 1 A set of order sheets (only one order sheet)

Order sheet 1	
Order item number	Menu ID
1	5
2	7

Table 2 A set of dishes (excerpt)

Menu ID	Menu	Cooking procedure
5	Meat sauce spaghetti	See Fig. 1
7	Napolitan spaghetti	See Fig. 1

Table 3 A set of processes (excerpt)

Process ID	Details	Parallel processing	Designated kitchen area	Maximum no. of simultaneous cooking	Required skill	Cooking time
4	Make a salad	0	3	1	3	80
11	Boil the noodles	1	1	2	1	240
13	Cook meat sauce	1	2	2	1	70
14	Stir-fry ingredients	1	2	2	1	120
15	Boil down the sauce	1	2	2	1	60
17	Put meat sauce	0	2	1	1	30
18	Put napolitan sauce	0	2	1	1	30

Table 4 A set of cooks

Cook ID	Name	Skills			Maximum no. of parallel cooking	Delay coefficient	Assigned cooking place
		1	2	3			
1	Tanaka	1	1	1	2	1.0	1, 2
2	Suzuki	1	0	1	2	1.0	3

Table 5 A set of cooking areas

Cooking place ID	Cooking place	Assigned Cook ID
1	Boiler	1
2	Gas stove	1
3	Salad field	2

なお, Table 1～Table 5 において, 英語名称と日本語名称の対応は以下のとおりである: 料理 (Menu), 料理人 (Cook), 調理場 (Cooking place), 注文票 (Order sheet), 注文明細番号 (Order item number), プロセス (Process), 並列可能性 (Parallel processing), 指定調理場 (Assigned cooking place), 最大同時調理数 (Maximum no. of simultaneous cooking), 必要スキル (Required kill), 調理時間 (Cooking time), 指定調理場 (Assigned cooking place), 所持スキル (Skills), 最大並列調理数 (Maximum no. of parallel cooking), 遅延係数 (Delay coefficient), 担当調理場 (Assigned cooking place).

2.6 スケジュールの計算

調理順序 G のスケジュールを計算するアルゴリズムを以下に示す.

[スケジュールの計算アルゴリズム]

- すべての料理人 $w \in W$ に対し, $s_w = 1$ とする. 選択グラフにおいて, ダミーオペレーション s の最早完了時刻を $t^e(s) = 0$ とする (0 を料理の開始時刻とした場合).
- 選択グラフにおいて, オペレーション g_{w,s_w} ($w \in W$) のすべての直前オペレーションの最早完了時刻が計算されているような w を 1 つ選び, w^* とする. このような w が存在しない場合は, 選択グラフに閉路が存在しており, G は実行可能でない.
- $g_{w^*,s_{w^*}}$ の最早開始時刻 $t(g_{w^*,s_{w^*}})$ を計算する ($t^e(g_{w^*,s_{w^*}})$ も同時に計算される). 計算方法は別途解説する.
- $s_{w^*} = s_{w^*} + 1$ とする. すべての $w \in W$ に対して, $|G_w| < s_w$ であれば計算を終了 (スケジュールの完成). そうでなければ, ステップ 2 に戻る.

上記アルゴリズムのステップ 3 でオペレーション $g_{w^*,s_{w^*}}$ の最早開始時刻 $t(g_{w^*,s_{w^*}})$ を計算する手続きを説明する. まず, 必要な定義を行なう. 選択グラフ上, オペレーション o の料理手順に関して直前に位置するオペレーションの集合 (オペレーション o を終点にもつ実線枝の始点オペレーションの集合) を $\text{prev_m}(o)$ と表わすことにする. $t^m(o) = \max_{g \in \text{prev_m}(o)} t^e(g)$ と定義すると, $t^m(o)$ はオペレーション o の処理が料理の調理手順上開始可能となった時刻を表わす. また, $t^e(g_{w,0}) = 0$ ($w \in W$) とする.

同時調理と並列調理を考慮しない場合であれば, $t(g_{w^*,s_{w^*}})$ は次式で計算される.

$$t(g_{w^*,s_{w^*}}) = \max\{t^m(g_{w^*,s_{w^*}}), t^e(g_{w^*,s_{w^*}-1})\} \quad (8)$$

同時調理と並列調理を考慮する場合, $t(g_{w^*,s_{w^*}})$ は以下の手続きで計算される.

$[t(g_{w^*,s_{w^*}})$ の計算法]

- つぎの条件 (i)～(iii) をすべて満たす場合, $g_{w^*,s_{w^*}}$ は $g_{w^*,s_{w^*}-1}$ と同時調理が可能である: (i) $g_{w^*,s_{w^*}}$ と $g_{w^*,s_{w^*}-1}$ が同一プロセス, (ii) $t^m(g_{w^*,s_{w^*}}) \leq t(g_{w^*,s_{w^*}-1})$, (iii) $g_{w^*,s_{w^*}}$ を $g_{w^*,s_{w^*}-1}$ と同時に調理しても, そのプロセスに設定されている最大同時調理数を越えない ($g_{w^*,s_{w^*}-1}$ が $g_{w^*,s_{w^*}-2}$, $g_{w^*,s_{w^*}-3}$, ... と同時調理されている場合もある). 同時調理が可能な場合は, 次式で計算する.

$$t(g_{w^*,s_{w^*}}) = t(g_{w^*,s_{w^*}-1}) \quad (9)$$

同時調理が可能でなければ, 以下のどちらかで計算する.

- $g_{w^*,s_{w^*}}$ が並列可能オペレーションの場合: t_w^{mp} を料理人 w のすでに最早完了時刻が計算済みのオペレーション (すなわち, $g_{w,1}, \dots, g_{w,s_w-1}$) の中で a_w (料理人 w の最大並列調理数) 番目に最早完了時刻が遅いオペレーションの最早完了時刻を表わす変数とする. すなわち, 時刻が t_w^{mp} 以降であれば, 料理人 w は新たに 1 つ並列可能プロセスを追加で実行することができる. よって, $t(g_{w^*,s_{w^*}})$ は次式で計算される.

$$g_{w^*, s_{w^*}}^t = \max\{t^m(g_{w^*, s_{w^*}}), t_{w^*}^{mp}\} \quad (10)$$

• $g_{w^*, s_{w^*}}$ が並列可能オペレーションでない場合: $t_{w^*}^{np}$ を料理人 w のすでに最早完了時刻が計算済みで並列可能でないオペレーションの中で、最も最早完了時刻が遅いオペレーションの最早完了時刻を表わす。すなわち、時刻が $t_{w^*}^{np}$ 以降であれば、料理人 w は新たに 1 つ並列不可プロセスを追加で実行することができる。ただし、その結果、料理人 w が実行する並列プロセス数が、その料理人の最大並列調理数を越えてはならない。よって、 $g_{w^*, s_{w^*}}^t$ は次式で計算される。

$$t(g_{w^*, s_{w^*}}) = \max\{t^m(g_{w^*, s_{w^*}}), t_{w^*}^{mp}, t_{w^*}^{np}\} \quad (11)$$

2.7 評価関数

前節の計算により、調理順序 G に対してスケジュールが得られる。本研究では得られたスケジュールの満足度を評価するために、以下に説明する 4 つの評価項目を用いる。最初の 2 つの評価項目は同卓スケジューリング問題の元々の要請から来るものであるが、加えて料理人への負荷や料理の品質を考慮した 2 つの評価項目を定義する。

卓ごとの全料理完成時間 (Latest service time, LST) 料理の提供時間を短くすることは利用客の満足度の向上と共に店舗の回転率上昇に繋がるため、最も優先される評価基準である。一方、各卓では全員の料理が出そろって食事を始めない利用客が多く見受けられる。そこで、料理の提供時間を評価する場合には一品ごとの料理提供時間の総和ではなく、同卓内で最も提供が遅かった料理の提供時間を評価項目とする。

卓ごとの料理提供時間の差 (Difference of service time, DST) 卓ごとに全料理がどれくらい同時に提供されるかを評価する。具体的には、卓ごとの全料理の完成時間と平均完成時間の差分の絶対値の和を評価項目とする。

調理時間 (Time of cooking, TOC) 調理中に加熱する工程がある場合や、鮮度が重視されるような工程がある中で、料理の調理時間が長くなってしまうと、料理の品質に悪影響を及ぼす。そこで、各料理の最初にオペレーションが完了した時刻から調理終了までの経過時間を評価項目とする。

担当外調理プロセス数 (Number of time to help, NTH) 本モデル化では料理人ごとに担当調理場が設定されているが、担当ではない調理場に赴き調理を行なうスケジュールの生成を許している (プロセスごとに作業場が決まっている)。担当外の調理場での作業は、忙しい調理場をサポートするために必要であるが、料理人が作業場を変更することによる混乱が発生する可能性がある。そこで、料理人が担当外の調理場で調理したオペレーション回数を評価項目とする。

上記の 4 つの評価項目 LST , DST , TOC のそれぞれは次式のように定式化される。 NTH については数式で表現する

ためには必要な定義が多くなるため、言葉による説明にとどめる。定式化にあたり、 $t^e(i, j, k)$ をオペレーション (i, j, k) の最早完了時刻、 $t^e(i, j)$ を注文票 i の注文明細番号 j の料理の完成時刻と定義する。すなわち $t^e(i, j) = \max_k t^e(i, j, k)$ である。また、 h_i を注文票 i に含まれる料理数とする。

$$LST = \sum_{i=1}^{n_I} \max_{j \in \{1, \dots, h_i\}} t^e(i, j) \quad (12)$$

$$DST = \sum_{i=1}^{n_I} \sum_{j=1}^{h_i} \left| t^e(i, j) - \frac{1}{h_i} \sum_{l=1}^{h_i} t^e(i, l) \right| \quad (13)$$

$$TOC = \sum_{i=1}^{n_I} \sum_{j=1}^{h_i} \left(t^e(i, j) - \min_k t^e(i, j, k) \right) \quad (14)$$

調理順序 G の評価関数 $E(G)$ を次式のような 4 つの評価項目の重み付き和で与えるものとする。評価項目の重みの値については、計算機実験の節で述べる。

$$E(G) = \alpha LST + \beta DST + \gamma TOC + \delta NTH \quad (15)$$

2.8 動的スケジューリング

これまでの節では、固定的に与えられた注文票に対して、調理スケジュールを作成する“静的”な同卓スケジューリング問題のモデル化を行なった。実際には、調理をしている間にも次々に新しい注文票が届いてくるので、それらの注文に対応するための動的スケジューリングが必要となる。本節では“動的”な同卓スケジューリング問題のモデル化と動的スケジューリングの方法を述べる。

注文票には時系列に従って $1, 2, \dots$ のように注文票 ID が振られるものとする。現時点の時刻を $t_{current}$ とし、現状では注文票 n_I までの調理スケジュールが組まれているとする。ここで、注文票 n_I 以降の新規の注文票 $n_I + 1, \dots, n_I + h$ に対して、これらに含まれる料理の調理スケジュールを現状のスケジュールに追加することを考える。

最も単純な方法は、注文票 n_I までの調理スケジュール (現状の調理スケジュール) は固定して、新規の注文票の料理はすべて現状のスケジュールの後に追加することである。しかし、現時点 $t_{current}$ において、現状の調理スケジュール (まだ取り掛かっていない調理) を一旦破棄して、新規注文票の料理を含めて再スケジュールを行なったほうが、効率的な調理スケジュールを構成することが可能である。一方、料理人は組まれた調理スケジュールに従って調理を行なっているため、予定された調理スケジュールが変更されると混乱してしまうという問題がある。そのため、動的スケジューリングでは現状の調理スケジュールを踏襲しつつ、新規の注文票に含まれる料理の調理を効率的に組み込んでいくような工夫が必要である。

本論文で提案する動的スケジューリングは以下の方法で行なう。概念図を **Fig. 3** に示す。

[動的スケジューリングの手順]

1. 動的スケジュールを行なうと判断した時刻を $t_{current}$ とし、現状では注文票 n_I までの料理の調理スケジュールが組

Fig. 3 A schedule before the dynamic scheduling (left) and a schedule after the dynamic scheduling (right). The gray operations are fixed during the dynamic scheduling process. Newly added operations are (2, 1, 1), (2, 1, 2), (2, 1, 3)

まれているとする。新規注文の注文票を $n_I + 1, \dots, n_I + h$ とする。

2. d_{margin} をマージン時間とし、時刻 $t_{current} + d_{margin}$ 以前に開始が予定されているオペレーションの調理順序（およびスケジュール）は現状のまま固定する。それ以外のオペレーションは料理人の調理順序から外す。

3. ステップ 2 で調理順序から外されたオペレーションと新規注文票に含まれるオペレーションに対して、固定された調理順序に以降の調理順序を決定する。調理順序の決定方法は静的スケジューリングの場合と同様である。ただし、時刻 $t_{current} + d_{margin}$ 以降でないと、それらのオペレーションを開始することはできない。

上記のステップ 1 で、マージン時間 d_{margin} は最低でも動的スケジュールの探索に要する計算時間以上に取る必要があるが、加えてスケジュールの変更により料理人が混乱しないだけのマージン時間を確保する必要がある。

3. 探索アルゴリズム

本研究では、同卓スケジューリング問題の解法としてシミュレーテッドアニーリング (Simulated annealing, SA) を用いる。本章では使用する SA アルゴリズムを説明する。

SA の探索において、解候補は調理順序 G で表わす。SA で用いる近傍構造には Relocation 近傍を用いる。Relocation 近傍は調理順序 G に対して 1 つのオペレーションを別の場所に移動することで得られる解集合（すべてのオペレーションに対し、すべての別の場所への移動を試す）である。ただし、移動後の解候補 G は実行可能であるための条件 (2.3 節参照) を満たしていなければならない。このようにして定義される近傍を $N(G)$ と表わすことにする。また、動的スケジューリングでは、現在の調理順序 G の固定部分は固定して、それ以降の部分でのみオペレーションの移動を行なうものとする。SA アルゴリズムを以下に示す。

[SA アルゴリズム]

1. 初期解を生成して、現在の解 G とする。暫定ベスト解 G^* を G で初期化する。 $iter = 0$ とする。温度パラメータを $T = T_{init}$ に設定。暫定ベスト解の発見温度を $T_b = T$ に設定。
2. $G' \in N(G)$ をランダムに 1 つ選択。

Table 6 Weight values for the evaluation function $E(G) = \alpha LST + \beta DST + \gamma TOC + \delta NTH$

$E(G)$	α	β	γ	δ
E_1	1	0	0	0
E_2	0	1	0	0
E_3	1	1	0	0
E_4	1	1	1	0
E_5	1	1	1	50
E_6	1	1	1	200
E_7	1	1	1	1000

3. $E(G') \leq E(G)$ であれば G' に遷移する。そうでなければ確率 $\exp(-\frac{E(G') - E(G)}{T})$ で G' に遷移する。

4. 解が遷移しなかった場合は $iter = iter + 1$ とする。現在の解 G が暫定ベスト解 G^* の評価値を更新していれば、 $G^* = G$, $T_b = T$, $iter = 0$ とする。

5. $iter = I_{max}$ であれば、 $T = \gamma T$ として、温度を減衰させる。 $T < \eta T_b$ であれば、 T を初期温度に戻す（再加熱）。 T の値が変更された場合には $iter = 0$ とする。

6. 探索時間が一定時間 $time_{max}$ に達していれば、 G^* を出力して探索終了。そうでなければ、ステップ 2 へ戻る。SA アルゴリズムのパラメータ値は予備実験により適切な値を設定した。実験ではそれぞれのつぎの値を用いる： $T_{init} = 60$, $I_{max} = 300$, $\gamma = 0.975$, $\eta = 0.15$, $time_{max} = 20$ (秒)。

ステップ 1 の初期解生成は、すべてのオペレーションを ID の若い順（辞書式）に、調理順序 $g_{w,s}$ の若い順（辞書式）に割り付けていくことで生成する。ただし、この操作は調理順序 G が実行可能であるという制約条件の下で行なう。

4. 計算機実験

本論文で提案した同卓スケジューリング問題のモデル化と SA によって得られるスケジュールを評価するための計算機実験を行なった。

4.1 実験方法

2.7 節で説明した 4 つの評価項目の 7 種類の組合せ（重み）を試し、それぞれの評価関数で得られたスケジュールの妥当性を評価する。7 種類の評価関数における重みは Table 6 のように設定する。実験条件は以下のとおりである。

- 料理人の数は 5 人。
- 最初の 1 枚の注文票に対する調理開始時刻を $t = 0$ とし

Table 7 Breakdown of the evaluation values (average and standard deviation) of the schedules obtained with the various evaluation functions

$E(G)$	LST	DST	TOC	NTH
E_1	2065 (18.6)	2654 (246.3)	2003 (293.6)	29 (4.8)
E_2	2404 (136.4)	307 (59.0)	5365 (1017.0)	40 (3.6)
E_3	2148 (46.7)	300 (35.5)	4289 (184.0)	41 (2.4)
E_4	2246 (54.2)	983 (138.3)	1359 (65.3)	39 (5.8)
E_5	2203 (105.6)	1353 (275.7)	1343 (94.7)	18 (3.5)
E_6	2273 (47.4)	1208 (183.9)	1414 (118.9)	14 (2.4)
E_7	2325 (55.1)	1235 (140.1)	1424 (105.1)	15 (1.7)

て、240 秒間隔で 1 注文票つづ、合計 3 つの注文票が到着する。

- 各注文票は 10 品の料理から構成される。
- 最初の 1 枚の注文票に対して静的スケジューリングを行ない、注文票が 1 枚到着するたびに即座に動的スケジューリングを開始する。一回の探索時間は 20 秒とする。マージン時間 d_{margin} は 30 秒とする。
- 各評価関数で 10 試行を行なう。

4.2 実験結果 1

Table 7 に 7 種類の評価関数を用いた SA で探索を行なって得られたそれぞれの調理スケジュールの評価値の内訳（平均値、分散）を示す。表中の灰色のセルは対応する評価項目が評価関数に含まれていないことを表わしている。また、**Fig. 4** に 7 つの評価関数ごとに、得られたスケジュールの典型例を示す。横軸の 1 メモリは 1 分で、縦軸に記載されている番号は注文明細番号を表わす。スケジュールは料理人ごとに色の濃淡を変えて表示している。

評価関数 E_1 を用いた場合、 LST のみを評価しているため、得られたスケジュールの LST 値は最も小さな値となっている。得られたスケジュールを見ると、各料理は自身の調理を開始から可能な限り早く完成するようにスケジュールが組まれる傾向にあることがわかる。つぎに、評価関数 E_2 を用いた場合、 DST のみを評価しているため、得られたスケジュールの DST 値は最小に近い値となっている。得られたスケジュールを見ても、各卓での料理の完成時刻が大体同じになっていることがわかる。一方、この場合の LST の値は最大となってしまっている。つぎに、評価関数 E_3 を用いた場合、 LST と DST の両方を考慮することで LST と DST が共に小さくなるようなスケジュールが作成されている。これは料理の同時同卓提供（かつ料理を早く提供）という観点からは望ましいスケジュールであるといえる。

E_3 で得られたスケジュールでは、いくつかの料理の調理において、オペレーション間の待機時間が比較長くなっている。たとえば、注文票 1 の注文明細番号 3 の料理の調理では、最終工程の盛付作業をそれが可能になってから 7 分後に開始している。このような調理途中の長期の待機時間を含むスケジュールは現実の調理業務にそぐわないと考えられる。そこで、評価関数 E_3 に評価項目 TOC を追加した評価関数 E_4 を考える。評価関数 E_4 を用いて得られたスケジュールの TOC

Fig. 5 Search process of the SA

の値は小さくなっていることが確認できる（しかし、 DST の値は悪化）。また、得られたスケジュールを見ても、料理途中の中断時間が少なくなっていることが確認できる。よって、評価項目 TOC を考慮することは、現実的な調理スケジュールを作成する上で有効であるといえる。

評価関数 E_5 – E_7 は評価関数 E_4 に評価項目 NTH をさまざまな重み δ を掛けて加えたものである。重み δ の値を大きくするほど各料理を 1 人の料理人で調理する傾向が強まり（多くの料理は同じ調理場で調理される）、調理手順の混乱を招かないスケジュールを得ることができていることが確認できる。ただし、それに伴い、 DST の値は悪化していることが確認できる。つまり、評価項目 NTH と DST はトレードオフの関係にあるので、重み δ の値は適切に調節する必要がある。

以上の実験結果より、同卓スケジューリング問題において現実に即した適切な調理スケジュールを作成するためには、4 つの評価項目 LST , DST , TOC , NTH をすべて考慮することが必要であると考えられる。

4.3 実験結果 2

実験結果 1 では順次到着する 3 つの注文票にして、動的スケジューリングで得られたスケジュールの最終結果を示した。本節では評価関数 E_7 を用いた場合の典型的な 1 試行を例に、SA によるスケジューリングの探索過程、および注文票が到着するたびにスケジュールが変更されるようすを示す。

Fig. 5 に SA の探索過程を示す。横軸は解の遷移回数を表わしているが、初期スケジューリングと 2 回の再スケジューリングを合わせて表示していることに注意されたい。縦軸は現在の解と暫定ベスト解の評価値を表わす。3 回のスケジューリング過程のそれぞれにおいて、現在の解の評価値が一時的に悪化しているのは SA の再加熱のためである。再加熱を繰り返すことで、暫定ベスト解の値が改善される場合があることがわかる。一方、グラフのようすからこれ以上再加熱を繰り返しても、さらに暫定ベスト解を更新することは難しいものと予想される。**Table 7** の結果より、1 試行ごとに得られるスケジュールの評価値にある程度のばらつきがあるので、（計算時間に余裕があり）さらに良いスケジュールを見つけたい場合には、これ以上再加熱を繰り返すよりも、試行回数を増やすほうが良いと考えられる。

Fig. 4 Schedules obtained with the various evaluation functions

Fig. 6 Schedules obtained in a step-by-step manner during the dynamic scheduling process

評価関数 E_7 を用いた場合で最終的なスケジュールが得られるまでの過程を示す。図に示した3つのスケジュールは左から、注文票1に対して作成したスケジュール、注文票2が到着した際に再スケジューリングして得られたスケジュール、注文票3が到着した際に再スケジューリングして得られたスケジュールである。図中の点線は注文票の到着時刻にマージン時間を加算した時刻である。

Fig. 6 に最終的なスケジュールが得られるまでの過程を示す。図に示した3つのスケジュールは左から、注文票1に対して作成したスケジュール、注文票2が到着した際に再スケジューリングして得られたスケジュール、注文票3が到着した際に再スケジューリングして得られたスケジュールである。図中の点線は注文票の到着時刻にマージン時間を加算した時刻である。

図より、注文票の到着時刻+マージン時間（図の点線）より前に開始時間がセットされているオペレーションの開始時刻は再スケジューリングでも変更していないことが確認できる。一方、それ以外のオペレーションについては僅かであるが開始時間が変更されている。また、再スケジューリングで固定されないオペレーションに関しては、すべて注文票の到着時刻+マージン時間（図の点線）以降に開始時間が設定されていることが確認できる。

5. おわりに

本研究ではレストランなどの飲食店で注文票が次々と到着するような動的環境に対して、同卓スケジューリング問題のモデル化を行ない、この問題に対してSAを用いた簡単な近似解法アルゴリズムを構築した。スケジュールの満足度を評価する評価項目として、卓ごとの全料理完成時間、卓ごとの料理提供時間の差、調理時間、担当外調理プロセス数を定義して、それらの評価項目を組合せて評価関数を作成して実験を行なった。その結果、料理の同時同卓提供を考慮しつつ、現

実に即した調理スケジュールを作成するためには、これらすべての評価項目を考慮することが必要であると考察した。

提案システムを実際の現場で使うためには課題も残されている。特に、スケジュールのロバスト性は重要で、立案スケジュールに対して実際の作業が遅れた場合に、その影響が時間の経過に伴って増大しないようなスケジュールを作成することが望まれる。今後はロバスト性も評価項目として取り入れた問題のモデル化に取り組んでいきたいと考えている。

参考文献

- 1) 繁野麻衣子：レストランの調理作業手順スケジューリング，オペレーションズ・リサーチ，**61**-10, 652/653 (2016)
- 2) 野中，信友，水山：料理の提供順序とタイミングを考慮した顧客満足度向上のための動的スケジューリング，2015年度精密工学会春季大会学術講演会講演論文集，97/98 (2015)
- 3) 今堀，柳浦：概説メタ戦略：オペレーションズ・リサーチ，**58**-12, 695/702 (2013)
- 4) 野々部，茨木：資源制約付きスケジューリング問題の定式化と近似解法，数理解析研究所講究録，**1120**, 88/97 (1999)
- 5) S. Hartmann and D. Briskorn: A survey of variants and extensions of the resource-constrained project scheduling problem, *European Journal of Operational Research*, **207**-1, 1/14 (2010)
- 6) R. Kolisch and S. Hartmann: Experimental investigation of heuristics for resource-constrained project scheduling, *European Journal of Operational Research*, **174**-1, 23/37 (2006)
- 7) J. Adams, E. Balas and D. Zawack: The Shifting Bottleneck Procedure for Job Shop Scheduling, *Management Science*, **34**-3, 391/401 (1988)
- 8) E. Nowicki and C. Smutnicki: An advanced tabu search algorithm for the job shop problem, *Journal of Scheduling*, **8**-2, 145/159 (2005)
- 9) 久保，ペドロソ，村松，レイス：あたらしい数値最適化—Python言語とGurobiで解く，231/244，近代科学社 (2012)
- 10) J. Blazewicz, W. Domschke and E. Pesch: The job shop scheduling problem: Conventional and new solution techniques, *European Journal of Operational Research*, **93**-1, 1/33 (1996)

[著 者 紹 介]

山 吹 卓 矢

2017 年徳島大学大学院先端技術科学教育部終了。同年、中央コンピューター（株）入社、現在に至る。

小 野 典 彦

1979 年東京工業大学理学部情報科学科卒業，81 年同大学大学院総合理工学研究科システム科学専攻修士課程修了。広島大学工学部助手，徳島大学工学部助教授を経て，97 年より同大学同学部教授，現在に至る。

永 田 裕 一（正会員）

1994 年東京工業大学応用物理学科卒業，2000 年同大学院総合理工学研究科知能システム科学専攻博士課程修了。北陸先端科学技術大学院大学情報科学専攻助手，東京工業大学大学院総合理工学研究科知能システム科学専攻助教，東京工業大学情報生命博士教育院特任准教授を経て，2014 年より徳島大学工学部知能情報工学科准教授，現在に至る。