algorithms

MDPI

Article

Optimal Cooking Procedure Presentation System for Multiple Recipes and Investigating Its Effect

Jin Nakabe [1,*], Teruhiro Mizumoto [2], Hirohiko Suwa [1,3] and Keiichi Yasumoto [1]

[1] Nara Institute of Science and Technology, Nara 630-0192, Japan; h-suwa@is.naist.jp (H.S.); yasumoto@is.naist.jp (K.Y.)
[2] Graduate School of Information Science and Technology, Osaka University, Osaka 565-0871, Japan; mizumoto@ist.osaka-u.ac.jp
[3] Institute of Physical and Chemical Research, Center for Advanced Intelligence Project (AIP), Tokyo 103-0027, Japan
* Correspondence: nakabe.jin.nb8@is.naist.jp

Abstract: As the number of users who cook their own food increases, there is increasing demand for an optimal cooking procedure for multiple dishes, but the optimal cooking procedure varies from user to user due to the difference of each user's cooking skill and environment. In this paper, we propose a system of presenting optimal cooking procedures that enables parallel cooking of multiple recipes. We formulate the problem of deciding optimal cooking procedures as a task scheduling problem by creating a task graph for each recipe. To reduce execution time, we propose two extensions to the preprocessing and bounding operation of PDF/IHS, a sequential optimization algorithm for the task scheduling problem, each taking into account the cooking characteristics. We confirmed that the proposed algorithm can reduce execution time by up to 44% compared to the base PDF/IHS, and increase execution time by about 900 times even when the number of required searches increases by 10,000 times. In addition, through the experiment with three recipes for 10 participants each, it was confirmed that by following the optimal cooking procedure for a certain menu, the actual cooking time was reduced by up to 13 min (14.8% of the time when users cooked freely) compared to the time when users cooked freely.

Keywords: cooking; optimization; algorithm; task scheduling problem

Citation: Nakabe, J.; Mizumoto, T.; Hirohiko, S.; Yasumoto, K. Optimal Cooking Procedure Presentation System for Multiple Recipes and Investigating Its Effect. *Algorithms* **2021**, *14*, 67. https://doi.org/10.3390/a14020067

Academic Editor: Frank Werner

Received: 13 January 2021
Accepted: 18 February 2021
Published: 23 February 2021

1. Introduction

Beginners who are not familiar with cooking take a long time to cook because they do not know how to cook efficiently. In order to support novice users, websites like Cookpad [1] and Allrecipes.com [2], which allow users to search for and exchange homemade cooking wisdom, are attracting attention. More and more people are using these websites, which provide easy-to-understand detailed cooking procedures, to help them decide what to cook for the day.

However, it is difficult to know which cooking tasks can be performed at the same time in order to reduce the overall cooking time when making multiple dishes. Whether parallel cooking is possible or not depends on the cooking environment of the user, such as the number of cooking tools, and the time required for each cooking task, which varies from user to user. Therefore, when cooking multiple dishes, it is necessary to present an optimal cooking procedure for each user, considering the dependency of multiple cooking tasks, the cooking environment of the user, and the time required by the user for each cooking task.

In this paper, we propose and evaluate a system to present the optimal cooking procedure for cooking multiple dishes in parallel, according to the cooking environment. The optimal cooking procedure corresponds to the procedure with the shortest cooking

time from the start of cooking to the end of cooking, including the washing tasks during cooking and the cleanup tasks after the end of cooking.

In order to create an optimal cooking procedure, we need to search for a cooking procedure that includes efficient parallel work, but the number of possible allocations of resources to cooking tasks is enormous, and searching all possible allocations is not practical. Therefore, we consider the search for the optimal cooking procedure as a variation of a task scheduling problem and formulate it as an optimal cooking procedure search problem. Furthermore, in order to obtain a solution to this problem, we propose two extensions to the existing optimization algorithms: the parallelized depth first/implicit heuristic search (PDF/IHS) [3] for preprocessing and bounding operation, considering the characteristics of cooking.

In order to evaluate the effectiveness of the algorithm, we used the proposed algorithm for a menu consisting of multiple recipes, and confirmed that (1) our method can reduce execution time by up to 44% compared to the base PDF/IHS, (2) the increase in execution time can be reduced to about 900 times even when the number of required searches increases by a factor of 10,000 (effectiveness), and (3) a cooking procedure with an error rate of 1% from the optimal cooking procedure can be obtained in about 10 s (practicality).

In order to evaluate the effectiveness of the system, we conducted an experiment with 10 participants. As a result, the proposed system has reduced the substantial cooking time by up to 13 min (14.8% of the free cooking time) compared to when the user cooks freely. Furthermore, from the results of the questionnaire to the participants in the experiment, we confirmed that the proposed system can reduce the cook's fatigue and provide a cooking procedure that is easy to cook with less sense of urgency and pressure.

The rest of the paper is organized as follows. Section 2 describes the existing work related to our study. Section 3 describes the relationship between the task scheduling problem and the optimal cooking procedure search problem, which we formulate as our target problem. Section 4 describes our proposed system and each component of the system. Section 5 describes the evaluation of the optimal cooking procedure search algorithm, the optimal cooking procedure, and the app, and finally Section 6 concludes the paper.

2. Related Work

This section describes research studies on the scheduling of cooking procedures and task scheduling problem and clarify the position of our study.

2.1. Scheduling of Cooking Procedures

In a study on the scheduling of cooking procedures, Matsushima et al. [4] used simulated annealing (SA) [5] to obtain the shortest possible cooking procedures for multiple dishes with a certain procedure for a cooking model. All of the cooking tasks were classified into one of six categories such as cutting or mixing, and the scheduling was performed using each cooking time calculated according to the skill level of the user and the ingredients. The scheduling also responds to changes in the cooking environment, such as the number of cooking tools. On the other hand, the scheduling does not take into account washing tasks or swapping of the task order within the same recipe. This may not be the optimal cooking procedure because it may not be possible to execute the cooking tasks in accordance with the cooking procedure that was created and because some of the cooking tasks may not be allocated during the free time when multiple recipes are cooked in parallel, even if they can be executed. Yamabuki et al. [6] proposed scheduling of cooking procedures for multiple people to prepare multiple dishes. They considered cooking as an extension of the resource constrained project scheduling problem (RCPSP) by viewing cooking tasks and cooks to be assigned as jobs and machines, respectively, and used SA to determine the cooking procedure. There are other studies to search for efficient cooking procedures, such as the creation of cooking models and the job-shop scheduling problem (JSP) [7–9], but these studies do not consider the washing task. In addition, previous studies assigned the cook as a machine, in theory, each cooking task can be performed when the cook is

assigned. However, in practice, it may be impossible to execute the cooking task because restrictions such as the number of cooking tools are not taken into account.

2.2. Task Scheduling Problem

The optimal cooking procedure creation problem can be considered as a task scheduling problem in which the cooking task is the task and the resources used in the cooking task are the processor. The task scheduling algorithms for solving this problem are classified into two categories: heuristic algorithms that obtain an approximate solution in a short time and optimization scheduling algorithms that obtain an optimal solution.

Several heuristic algorithms have been proposed as heuristic algorithms to obtain approximate solutions: list scheduling [10], which executes the task as soon as it becomes available; critical path (CP) [11], which allocates the task with the longest path length from the exit node of the task graph to each task; and the critical path/most immediate successors first (CP/MISF) [12], which prioritizes the task with the largest number of subsequent tasks if there are multiple tasks with the same CP length. Kasahara et al. proposed two optimization scheduling algorithms for obtaining optimal solutions: the sequential optimization algorithm DF/IHS [12], in which the initial solution is obtained by the CP/MISF and the search order is determined by the task assignment priority, and the bound operation is performed by the sharp lower bound [13] and the parallel algorithm PDF/IHS [3], which is an extension of the DF/IHS for parallel processing. As an improvement of the DF/IHS, Nakamura et al. [14] reduced the number of search nodes in the DF/IHS by comparing the search in progress with the nodes that have already been searched, and Matsuse et al. [15,16] proposed a method using a hash table for the comparison. In addition, to obtain an optimization algorithm that considers the overhead of data transfer to the DF/IHS, Shibuya et al. [17] proposed an algorithm that adds the remaining distance including communication overhead (REDIC) [18] to the DF/IHS to obtain a lower limit that considers the delay from each task to a specific range of subsequent tasks. Furthermore, as an improvement of this algorithm, some algorithms [19] have been proposed to perform more accurate lower limit computation by optimizing the sub-task graphs in the lower limit computation, and some algorithms [20] to reduce the computation time by hierarchical scheduling in which the sub-task graph and the whole task graph are scheduled multiple times. However, simply using these optimization algorithms as-is will not work efficiently as they do not take into account the characteristics of cooking, such as the need for multiple resources for cooking tasks at the same time and the later occurrence of washing tasks that are not included in the recipe.

2.3. Position of This Study

Existing studies on the scheduling of cooking procedures do not take into account swapping the order of cooking tasks within the same recipe or the washing tasks that inherently occur during cooking. In addition, using the existing optimization scheduling algorithms without any modification, it is impossible to capture the representative characteristics of cooking. Therefore, in this paper, we aim to construct an optimal cooking procedure presentation system that includes the washing tasks during cooking and cleanup tasks after the end of cooking. We consider the search for cooking procedures as a task scheduling problem and propose an optimal cooking procedure search algorithm based on the PDF/IHS method that takes into account the characteristics of cooking.

3. Definition of Problem

This section formulates the optimal cooking procedure search problem after describing the relationship between it and the general task scheduling problem.

3.1. Expanding the Task Scheduling Problem to the Optimal Cooking Procedure Search Problem

The task scheduling problem is the problem of deriving a schedule that minimizes the execution time (schedule length) of parallel processing of a set of n tasks $T = \{t_1, t_2, \ldots, t_n\}$

with arbitrary processing time and precedence constraints on m processors of equal performance, which is known to be a strong NP-hard [21]. This task set T, which is described by a directed acyclic graph (DAG) as a task graph $G(T, E)$ (where E is the set of directed edges), with one start node S and one end node E, all of which are reachable from the start node. Figure 1 shows an example of a task graph consisting of seven tasks.

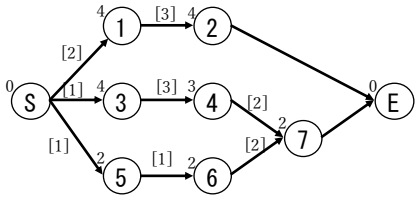


Figure 1. Example of a task graph.

The node in Figure 1 is a task, the number in the node is the task number, the number at the top left of the node is the processing time of the task, the number on edge is the overhead time, and each directed edge represents a partially-ordered constraint between tasks. The task scheduling problem is necessary to consider overhead time and to get a schedule that minimizes the execution time.

The optimal cooking procedure search problem is a task scheduling problem by considering each recipe's cooking tasks as tasks, the resources (user, knife, board, etc.) as processors, and the washing tasks as overhead. Here, as the cooking time required for each cooking/washing task is different for each user, it is necessary to consider which tasks can be assigned to a gap time to efficiently reduce the cooking time. Furthermore, the most important difference between the optimal cooking procedure search problem and the task scheduling problem is the treatment of the washing tasks. The overhead time of the task scheduling problem occurs when the preceding and following tasks, which are related to each other, are processed by different processors, as described above. On the other hand, the washing tasks in cooking are determined by the last cooking tasks that used the resources. For example, if the user is about to cut vegetables and has previously performed the cooking task of cutting meat, the knife and cutting board need to be washed (unless the vegetables are cooked with the meat). Thus, in the optimal cooking procedure search problem, the overhead cannot be defined in advance because the necessity of the washing task changes depending on the combination of cooking tasks. In the formulation of the optimal cooking procedure search problem, we must take this difference into account.

3.2. Formulation of the Optimal Cooking Procedure Search Problem

In this section, we summarize the input recipe information, the output cooking procedure, and the washing operations to be considered in the optimal cooking procedure search problem, and then describe the objective function.

3.2.1. Recipe Information

The recipe set $RR = \{R_1, R_2, \ldots, R_k\}$ is composed of k recipes, where k is the number of recipes selected by the user, and there is no limitation to the number of recipes k. Each recipe $R_i = \{r_{i_1}, r_{i_2}, \ldots, r_{i_{l_i}}\} (1 \leq i \leq k)$ is composed of l_i cooking tasks, where l_i is the number of cooking tasks in R_i. The cooking tasks in this study are subdivided into tasks dealing with only one food stuff, for example, (1) cutting onions and (2) cutting cabbage instead of (1) cutting onions and cabbage.

A cooking task r in a recipe has the following elements: {ingredient, quantity, cooking content, required resource, holding resource, cooking time, preparing time, cooking type, holding type, starting requirement, holding requirement, and previous task}. Table 1 shows an example of the cooking tasks included in a hamburger recipe except for the ingredient, quantity, and cooking-content.

Table 1. Cooking tasks included in hamburger steak recipe.

Task Number	Required Resource	Holding Resource	Cooking Time	Preparing Time	Cooking Type	Holding Type	Starting Requirement	Holding Requirement	Previous Task
1	user, knife, board	bowl	150	0	vegetable	A		2	
2	bowl, microwave	bowl	180	60	A	A	1	3	
3	user, bowl, tbsp	plate	390	0	A	A	2	4	
4	user, pan, stove, turner	pan, stove	330	0	B	B	3	6	
5	user, bowl, tbsp	bowl	120	0	C	C		6	
6	pan, stove, lid	pan	300	30	B	B	4,5	7	4
7	user, pan, turner, spoon		120	0	B		6		

The required resource is the set of cooking tools used in the cooking task. The holding resource is the set of resources needed to hold the cooked food after completing the cooking task. In the example of Table 1, cooking task (1) uses {user, knife, and board} for the cooking task and a {bowl} for holding the cooked food. The cooking time is the time required to perform the cooking task. This value is the value if the time is mentioned in the recipe, or the actual time required by the user to perform the cooking task in the past if it is not mentioned. The preparing time is the time required to prepare for the start of the cooking task. However, if the required resource includes "user", the preparing time is 0. The cooking type and the holding type are assigned to the resources included in the required resource and the holding resource after completing the cooking task, respectively, and used to determine the occurrence of the washing task. The definition of a washing task is explained in detail in Section 3.2.3. The starting requirement is the set of cooking tasks that must be completed before the cooking task can be performed. The holding requirement indicates the requirement to free the holding resource. In the example of Table 1, cooking tasks 4 and 5 must be completed before starting cooking task 6, and the holding resource for cooking task 6, {pan}, cannot use for any other cooking task until starting cooking task 7. The previous task represents the cooking task number where the end time of that cooking task and the start time of the next cooking task must match in the cooking procedure. In the example in Table 1, the end time of cooking task 4 and the start time of cooking task 6 must coincide within the cooking procedure.

In this study, we assume that the available resources are the user and the tools used for cooking (the number of resources is variable), and each number is represented by a vector $\boldsymbol{RS} = (RS_1, RS_2, \ldots, RS_g)$, where g is the number of resource types, and each resource number can be changed depending on the environment of the user, as long as it is greater than or equal to the number required for each cooking task. Note that immediately after the start of cooking, all resources are already available. For these resources, different resources can be used in parallel at a given time, and a number of resources can be used in parallel. However, for a holding resource, the same resource can be used if it has the same holding type even if that resource is currently in use.

3.2.2. Cooking Procedure

The output of the optimal cooking procedure problem, the cooking procedure, is created as a sequence of all of the cooking tasks contained in R in RR. The cooking procedure to be created is denoted by $S = \langle s_1, s_2, \ldots, s_n \rangle$, and n is the number of tasks to be executed sequentially. In addition, the S cooking procedure includes washing tasks that occur during the execution of S and cleanup tasks, whereby all of the used resources are washed at the end of cooking. By following S, the user can automatically parallelize the cooking tasks and cook efficiently.

The cooking procedure S must satisfy the following three constraints.

The first constraint is that, in order to complete all the recipes selected by the user, S includes all of the cooking tasks for all the recipes in RR and is represented by the following equation,

$$\bigcup_{R \in RR} \bigcup_{r \in R} r \subseteq \bigcup_{s \in S} s \tag{1}$$

The second constraint is that at all times in S, the number of resources used is less than or equal to the number of available resources. Given that the time in the cooking procedure

is t and that each number of resources used at a given time t is $\boldsymbol{rs_t} = (rs_{t,1}, rs_{t,2}, \ldots, rs_{t,g})$, this constraint is expressed as

$$\forall t(0 \leq t \leq Time(S)), \forall j(1 \leq j \leq g), rs_{t,j} \leq RS_j \tag{2}$$

where $Time(S)$ is the time it takes to finish cooking procedure S.

The third constraint is that in each recipe R, prior tasks $Pri(r)$ must be finished before the task $r \in R$ starts and this must also hold in S. This constraint is expressed as

$$\begin{gathered} \forall s \in S, \forall s' \in Pri(s), \\ FinishTime(s') \leq StartTime(s) \end{gathered} \tag{3}$$

$StartTime(s)$ and $FinishTime(s)$ are functions to obtain the start and end time of cooking task s in S, respectively.

3.2.3. Definition of Washing Task

It is necessary to wash the cooking tools at the beginning of a cooking task. For example, if meat was previously prepared with a knife on a cutting board, the user needs to wash the knife and cutting board before performing the cooking task of cutting vegetables. In this study, the washing task occurs when the type already assigned to each resource is different from the cooking type and holding type of the cooking task, and the resource becomes available when the washing task is performed. In the case of cooking in the order of the task number in Table 1, the bowl, which is the holding resource, is first assigned the holding type "A" by the cooking task 1. Then, when this bowl is used in cooking task 2 and 3, no washing task occurs because the type of the bowl (A) and the cooking type and holding type of the cooking tasks (A) match. On the other hand, when this bowl is used in cooking task 5, a washing task occurs because the type of bowl (A) and the type of cooking and holding (C) are different. The time required for the washing task is assumed to be a time vector $\boldsymbol{WT} = (WT_1, WT_2, \ldots, WT_g)$ defined for each resource, and a target resource and user are required to execute the washing task.

3.2.4. Objective Function

In this study, the optimal cooking procedure is the shortest cooking time from the start of cooking to the end of cooking, including the washing task during cooking and the cleanup task after the end of cooking, so the objective function is as follows:

$$Minimize\ (Time(S)) \tag{4}$$

4. System Overview

This section describes our proposed system and each component of the system.

4.1. Proposed System

Figure 2 shows the structure and data flow of our proposed system. Each number in Figure 2 shows the data flow and the app shows the Android application "Optimal Cooking Procedure Presentation Application" we implemented.

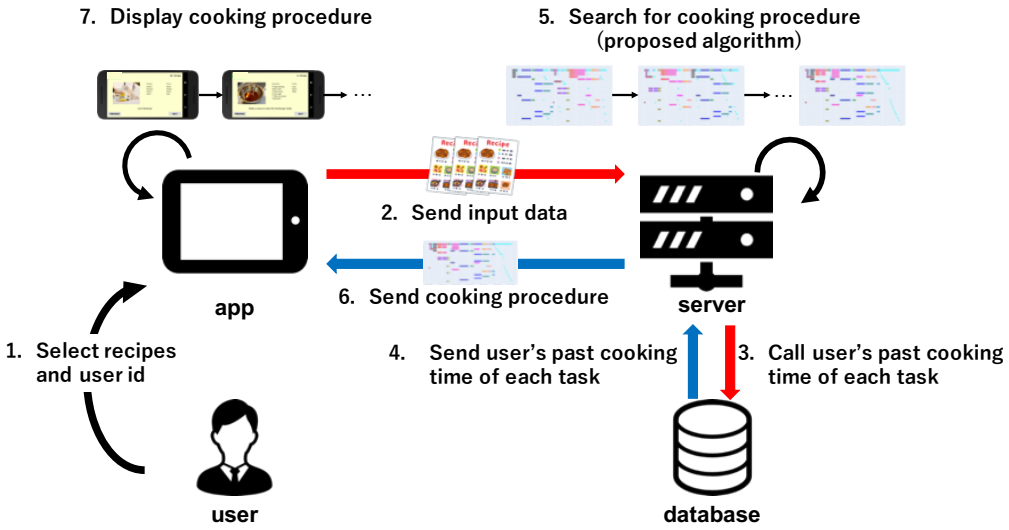


Figure 2. Structure and data flow of our proposed system.

4.2. Optimal Cooking Procedure Search Algorithm

In the proposed system, the user first selects recipes from those displayed in the app and enters the user id. The input data are sent to the server, and based on the selected recipes and the user id, cooking information such as cooking time is retrieved from the database. The server searches for the optimal cooking procedure using "optimal cooking procedure search algorithm" and sends the result to the app. The user can use the app to confirm the cooking task s/he is doing now and should perform next while cooking.

Sections 4.2 and 4.3 describe the optimal cooking procedure search algorithm and the app, respectively.

4.2.1. Base Algorithm: PDF/IHS

In this study, we use the PDF/IHS as the base algorithm to solve the optimal cooking procedure search problem, and improve its preprocessing and bounding operation. The PDF/IHS is an extension of the DF/IHS, for parallel processing, and the DF/IHS mainly consists of preprocessing, branching operation, and bounding operation. In the preprocessing of the DF/IHS, the task numbers are renumbered using the method used in the CP/MISF described in Section 2, and heuristically good solutions are collected on the left side of the search tree. In the search phase, it is possible to search from the left of the search tree using the DF/FIFO, starting from heuristically good solutions.

Figure 3 shows the task graph with the DF/IHS preprocessing applied to Figure 1. Figure 4 shows the partial search tree when searching for a solution using the DF/IHS method when Figure 3 assigned to two processors.

The gantt chart in Figure 4 shows the results of the allocation to processors up to that time, with processor 1 at the top and processor 2 at the bottom. Furthermore, t is the time when the branching operation is performed, and R is the task number that can be allocated to the processor at time t. When there are multiple assignable tasks, this partial search tree is created by adding the child nodes of the current node from the left in the order of decreasing task number. However, in the task scheduling problem, if the processing time of each task is different, the optimal solution may not be obtained by list scheduling that assigns tasks to processors as soon as they become available [22]. Therefore, in the DF/IHS method, a node is created including tasks (ϕ in Figure 4) that force the processor to be idle at the time of division. Here, there is actually some overhead in the allocation process, but for simplicity we ignore it here.

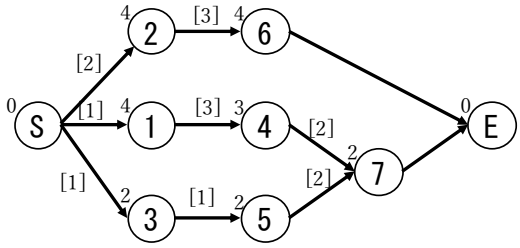


Figure 3. Re-numbered task graph in Figure 1.

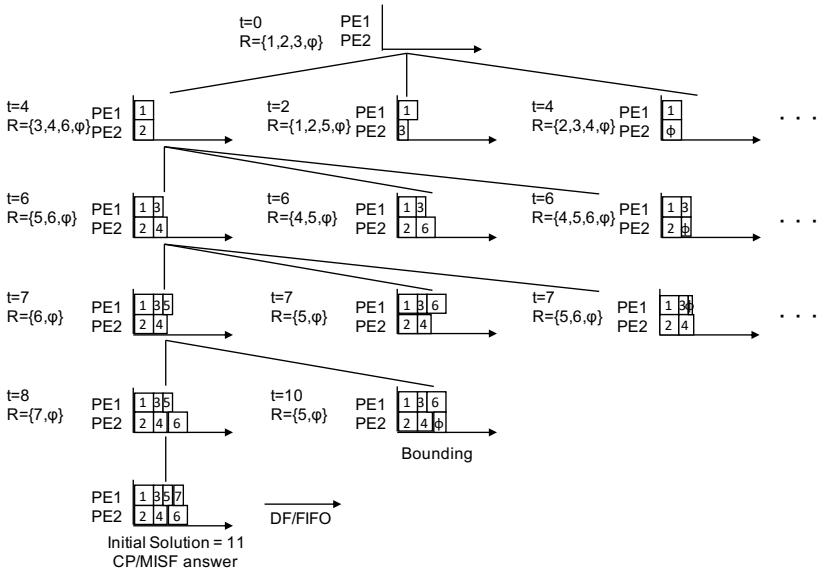


Figure 4. Partial search tree in DF/IHS.

4.2.2. Overview of the Optimal Cooking Procedure Search Algorithm

Algorithm 1 shows the pseudocode of a function that recursively creates a search tree used in the optimal cooking procedure search algorithm. In Algorithm 1, P is the set of cooking/washing tasks that have already been assigned a start/end time in the schedule of the cooking procedure currently being explored, E is the set of cooking tasks that satisfy the starting requirement in the current state of P, and F is the set that contains only the cooking tasks in P.

Algorithm 1 performs preprocessing on E and initialization of a tentative optimal cooking procedure O for the first call only. Figure 9 describes the preprocessing. Then, the function *isBounded* compares P and O to determine the bounding operation. Section 4.2.4 describes the bounding operation. Then, select one cooking task from the E, and call the function *Reconstruct*. Next, the selected cooking task is added to P, and the function *Reconstruct* determines whether the parallelized result satisfies all the constraints. Then, when the constraint is found to be satisfied, update E and F, and call the function *Recursive* again. At this time, the function *AddExecutableTask* adds the cooking task that satisfies the starting requirement to E by the cooking task e. After all the cooking tasks are added to P, the function *AddCleanUpTask* adds the cleanup task to P. Here, the washing tasks are added to the gap time when there exists a gap time in P during which the washing tasks can be performed. Finally, P and O are compared, and if P has a shorter total cooking time than O, P is updated with O.

Algorithm 2 shows the pseudocode for the *Reconstruct* function that parallelizes the cooking operations and checks the constraints used in Algorithm 1. In Algorithm 2, e is the cooking task to be added to P, U is the set of resources allocated to e, W is the set of resources in U that need to be washed before e is executed, and G is the set of gap time in P where the user is not used.

Algorithm 1: *Recursive Function*

input : E : *Executable Cooking task List*
F : *Finished Cooking task List*
P : *Provided Schedule*
output: O : *Optimal Cooking Schedule*

```
Pre-processing for E only once ;
Initialize O only once ;
if isBounded (O,P) then
    return
if E is not empty then
    foreach element e in E in order from top do
        isOk, P ← Reconstruct(e,P) ;
        if isOk then
            Remove(E, e) ;
            AddFinishTask(F,e) ;
            AddExecutableTask(E,e) ;
            Recursive(E, F, P) ;
else
    AddCleanUpTask(P) ;
    if time(P) < time(O) then
        O ← P ;
```

Figure 5 illustrates an example of how Algorithm 2 works when task D is added to the temporary schedule P in which tasks A, B, and C are already added.

Algorithm 2: *Reconstruct Function*

input : e : *Cooking Task*
P : *Provided Schedule*
output: *isOk* : *Does P Satisfy Continuity*
P : *Provided Schedule*

```
U ← AllocateResources(e) ;
W ← GetNeedWashingResources(U) ;
if W is not empty then
    G ← GetNotUsingUserTime(P) ;
    InsertWashingTask(P,W,G) ;
    foreach element w in W do
        AddWashingTask(P, w) ;
AddCookingTask(P,e) ;
isOk ← isSatisfyContinuity(P) ;
```

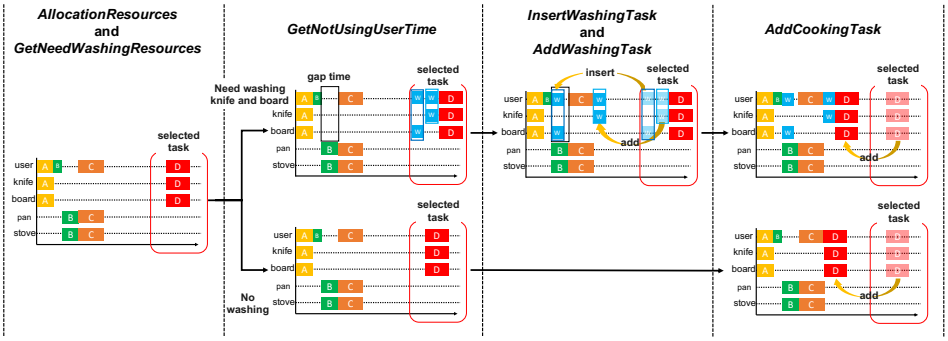


Figure 5. Example of how Algorithm 2 works.

In Algorithm 2, we first allocate the resources U to be used in e by the function *AllocationResources*. This function allocates resources in the following order when there are multiple candidate resources to be allocated.

- A resource that is not being used and does not require a washing task
- A resource that is not being used and requires a washing task
- Earliest available resource

Next, the function *GetNeedWashinResources* retrieves the set of resources W in U that need the washing task. After that, the function *GetNotUsingUserTime* obtains the set of gap time G during which the user can do the washing task. If the generated washing task can insert in G, it inserts into P by the function *InsertWashingTask*. If it cannot insert, it is added to P by the function *AddWashingTask*. After all the washing tasks are added, the function *AddCookingTask* adds e to P by prepending. Finally, the function $isSatisfyContinuity$ determines whether O satisfies the continuity of the recipe.

4.2.3. Proposed Method of Preprocessing

We propose two preprocessing methods for the optimal cooking procedure search algorithm, using the following indices:

- Method 1: Sorting by critical path (CP)
- Method 2: Sorting by user-free time

These preprocessing steps are done only once for E in the first line of Algorithm 1.

Method 1 is a method to sort tasks in order of CP for each cooking task in E. This method is also used in the DF/IHS. In cooking, a cooking task with a large CP means that there are many cooking tasks after it, or that there are cooking tasks that require a lot of time after it. If there is a cooking task that requires a lot of time later, the time occupied by a particular resource will increase, and other cooking tasks that use the resource cannot be performed. On the other hand, cooking tasks that use unoccupied resources can be performed in parallel. Therefore, assigning cooking tasks with large CP first increases the number of candidates for cooking tasks that can be performed in parallel. This preprocessing allows us to obtain an accurate initial solution and to update the tentative solution efficiently.

In method 2, in the task graph for a given recipe, assign a time value to cooking tasks that do not require a user and sort the cooking tasks in ascending order. This value represents the amount of time that the cooking task can cook in parallel with another cooking task. There are certain cooking tasks that require a user, and the washing task also requires a user. By performing cooking tasks that do not require a user and cooking tasks that do require a user at the same time, the cooking time can be reduced. Furthermore, by assigning cooking tasks that do not require a user to a variable (provided schedule) P at an early phase, the number of candidates for cooking tasks that can be executed in parallel increases. This preprocessing also allows us to obtain an accurate initial solution and to update the provided solution efficiently.

4.2.4. Proposed Method of Bounding Operation

As bounding operations for optimal cooking procedure search, we propose the following two bounding operations instead of those used in PDF/IHS:

- Method 1: Bounding by critical path and washing task time
- Method 2: Bounding by remaining time for each resource and washing task time

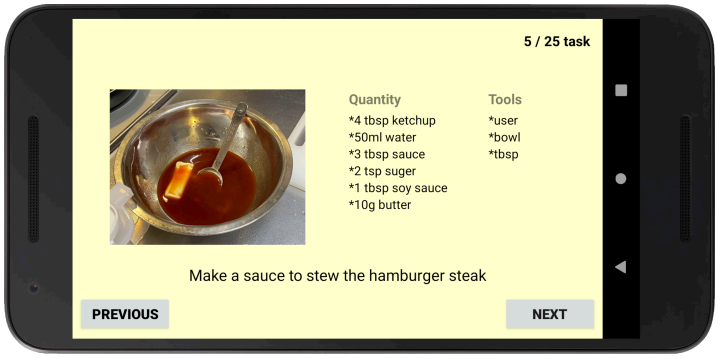
These bounding operations are performed by the function is *Bounded* in Algorithm 1.

The largest critical path of the cooking task not assigned to P will always be the time that is added to P. As we know at the time of P assignment that the resources required for washing tasks during cleanup are also known, we perform the bounding operation when the total time of the sum of the washing task time and critical path added to the tentative end time of P is not better than O. Note that when there is a cooking task that does not require the user to be on the edge of critical path, the washing task time is calculated considering the gap time, because it may be possible to insert the washing task in the gap time.

The remaining time for each resource in method 2 is the time that is always added to the tentative completion time of P because it cannot be parallelized reliably due to resource constraints. For example, when there is only one user, cooking tasks that require a user cannot be performed in parallel. Therefore, the total cooking time that includes the user in the required resource of cooking tasks that have not been added to P is always added to the tentative end time of P. As in Method 1, at the time of P assignment, we also know the resources that need to be washed during cleanup. Therefore, the bounding operation is performed when the cooking time, which is the sum of the time already required by the allocation of P plus the maximum of the total usage time for each of all resources included in the cooking tasks not allocated to P divided by the number of resources plus the washing time, is not better than O.

4.3. Optimal Cooking Procedure Presentation Application

Figure 6 shows the interface of the app used by the user to confirm the optimal cooking procedure.



(**a**) Cooking task display screen

(**b**) Washing task display screen

Figure 6. User interface.

In this app, the user can see the task image, the content of the task, and the current progress for both cooking and washing tasks and can use the buttons at bottom left/right to navigate to the previous/next task screen. In particular, in the case of cooking tasks, it is possible to see by pictures what the current cooking task will be in when it is finished, so even a beginner user can easily know how to perform the coking task. In addition, unlike most websites, only a single cooking/washing task is displayed, allowing the user to easily grasp the quantity and contents of the cooking/washing task and concentrate on a single task.

5. Evaluation

This section describes the evaluation of the optimal cooking procedure search algorithm for actual menus created from multiple recipes. In addition, we evaluate the optimal cooking procedure and the app through an experiment to show the usefulness of this study.

5.1. Evaluation of the Optimal Cooking Procedure Search Algorithm

5.1.1. Experiment Overview

We have converted eight real recipes into a form that can be used by our algorithm and created two common menus (Table 2) from these recipes. We show the effectiveness of the proposed algorithm by applying it to these two menus and comparing the execution time. Menus 1 and 2 consist of {salt-grilled mackerel, miso soup, and Japanese omelet}, and {hamburg steak, miso soup, and potato salad}, respectively. The total number of cooking tasks and the estimated total cooking time for menus 1 and 2 are 13 and 18 tasks, and 2550 and 3960 s, respectively, with menu 2 requiring more cooking tasks and more cooking time. Therefore, when all the cooking procedures are searched to satisfy the constraints in the recipe, the number of required searches for menu 2 is approximately 10,000 times greater than for menu 1. For these menus, we apply and evaluate the following six methods: the base PDF/IHS; SA used by Matsushima et al.'s research [4], which is the most similar to our research; and four methods, from a combination of two preprocessing and two bounding operations that we propose.

Table 2. Menu information.

	Recipe Name	Task Number	Estimated Total Cooking Time	Number of Required Searches
menu 1	salt-grilled mackerel	5	900	8.65×10^5
	miso soup	5	990	
	Japanese omelet	3	660	
menu 2	hamburg steak	7	1680	8.82×10^9
	miso soup	5	990	
	potato salad	6	1290	

The evaluation is done by (1) execution time: the time until the algorithm finishes executing, (2) convergence time: the time until the tentative solution in the algorithm converges to the optimal cooking procedure, and (3) reached time: the time until the cooking procedure in which the error from the total cooking time of the optimal cooking procedure is 1% is defined as the semi-optimal cooking procedure and is reached.

We use (1) the execution time to compare the basic performance of the algorithms, (2) the convergence time to identify the time required to obtain the optimal solution, and (3) the reached time to identify the time required to obtain the solution when a quasi-optimal solution is allowed (defined in this paper as 1% error from the optimal solution) with actual operation in mind.

The algorithms are implemented in Python and run on a machine with Intel Core i7 3.4 GHz CPU, 32 GB memory, and 8 concurrent processes.

5.1.2. Results

Table 3 shows the total cooking time for the cooking procedures obtained by the base PDF/IHS, SA, and proposed algorithms; the total cooking time for the SA is the average total cooking time obtained when 10 random initial cooking procedures are given. The optimal total cooking time for menus 1 and 2 is 2250 s and 3630 s, respectively, based on the PDF/IHS results. The proposed algorithm reaches the optimal total cooking time, while the average total cooking time of the SA is larger.

Table 3. Total cooking time obtained by the algorithm.

	Total Cooking Time [s]	
	Menu 1	**Menu 2**
Base PDF/IHS	**2250**	**3630**
SA (10 times Avg.)	2441	3786
Proposed Algorithm	**2250**	**3630**

Table 4 shows the execution time, convergence time, and reached time of the five methods except for the SA. Here, each combination of two preprocessing and two bounding operations that we propose is denoted as combination 1 to 4, respectively. For menu 1, the shortest execution time is 1.32 s for combination 2, the convergence time is 0.43 s for combination 4, and the reached time is 0.12 s for combination 1 or 2. In addition, the execution time is less than 2 s for all combinations. For menu 2, the shortest execution time is 1202 s for combination 4, convergence time is 221 s for combination 4, and reached time is 6.36 s for combination 2. Regarding execution time, the base PDF/IHS performed the worst, with 2141 s execution time, which is 15 min longer than the fastest (combination 4). In all methods, the reached time is within 10 s.

Table 4. Execution time, convergence time, and reached time of the algorithm.

	Execution Time [s]		**Convergence Time [s]**		**Reached Time [s]**	
	Menu 1	**Menu 2**	**Menu 1**	**Menu 2**	**Menu 1**	**Menu 2**
Base PDF/IHS	1.98	2141	1.19	796	0.68	7.08
combination 1 pre-processing 1 + bounding 1	1.45	1645	0.52	584	**0.12**	7.30
combination 2 pre-processing 1 + bounding 2	**1.32**	1207	0.49	437	**0.12**	**6.36**
combination 3 pre-processing 2 + bounding 1	1.52	1633	0.50	286	0.14	9.58
combination 4 pre-processing 2 + bounding 2	1.35	**1202**	**0.43**	**221**	0.14	7.48

Figures 7 and 8 show the transition of updating the tentative solution. In each figure, the horizontal axis represents the elapsed time and the vertical axis represents the total cooking time. Each plot in the figure shows the elapsed time and the total cooking time when the tentative solution was actually updated. The light blue and purple horizontal lines in the figure represent the total cooking time of the semi-optimal and optimal cooking procedures, respectively. From Figures 7 and 8, we can confirm that the tentative solutions for both menus are rapidly updated after the start of the algorithm.

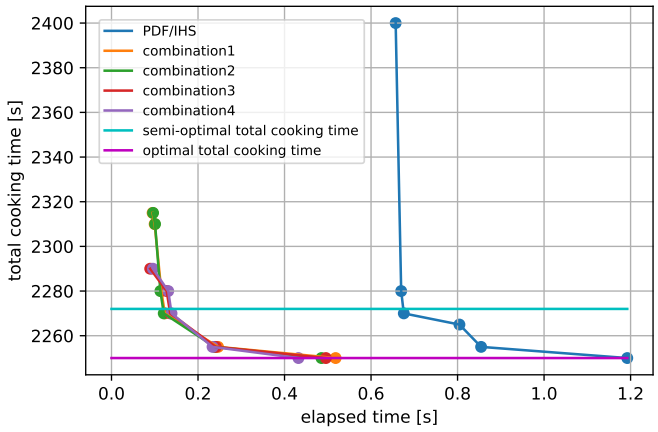


Figure 7. Tentative solution updates (menu 1).

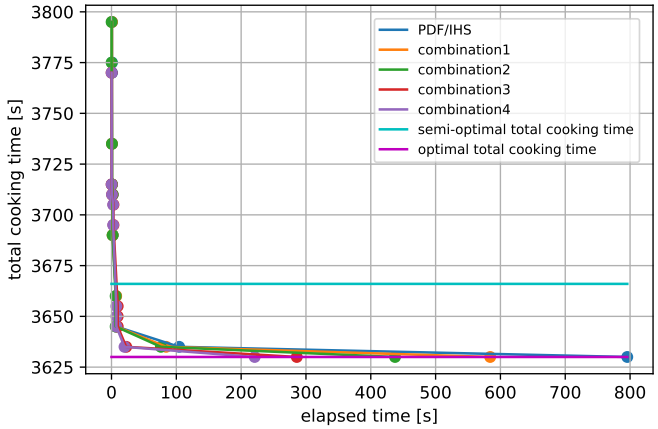


Figure 8. Tentative solution updates (menu 2).

5.1.3. Discussion

From Table 3, we can see that the average cooking time obtained by SA is significantly worse than the optimal total cooking time. This is because when the order of the two cooking tasks in the cooking procedure is switched in the SA, there are many cases where the procedure does not work due to the constraints of "holding requirement" and "previous task." Therefore, the SA method is not suitable for the optimal cooking procedure search problem if it is unchanged.

From Table 4, comparing the base PDF/IHS with our proposed algorithm, we can confirm that the execution time is greatly reduced from 1.98 s to 1.32 s for menu 1 and from 2141 s to 1202 s for menu 2. This is a 33% and 44% execution time reduction for menus 1 and 2, respectively. This indicates that our proposed method works effectively for the optimal cooking procedure search problem. We can say that the combination of preprocessing and bounding operations suitable for the optimal cooking procedure search algorithm is the combination including the bounding operation method 2. The reason for the superior results of the bounding operation method 2 is that many cooking tasks use "user" as resources, so even if the recipes containing the cooking tasks are different, the remaining time for each resource across multiple recipes can be calculated, and this method works more effectively than method 1, which can only calculate the critical path of the same recipe. In addition, while the number of required searches for menu 2 is approximately 10,000 times larger than that for menu 1 (Table 2), the execution time is about 900 times (Table 4). This suggests that the proposed algorithm is effective in reducing the execution time even when the number of required searches increases. However, the

minimum execution time for menu 2 is about 20 min (1202 s), which is not acceptable when considering actual operation. On the other hand, the proposed algorithm can obtain a semi-optimal cooking procedure with an error rate of 1% from the optimal solution in about 10 s for all combinations, even for menu 2 (Table 4). We also observed that the tentative solution was updated rapidly after the algorithm started (Figures 7 and 8). Therefore, we believe that the proposed algorithm can sufficiently present significant cooking procedures in real-time by, for example, terminating the calculation in 10 s, and has sufficient practicality.

5.2. Evaluation of Optimal Cooking Procedure Presenting Application

5.2.1. Experiment Overview

In order to confirm that the proposed app can shorten the cooking time of users, we conducted an experiment using menu 2 with 10 male and female participants (all graduate students). First, we conducted a questionnaire survey to determine the cooking skill level of the participants, and divided them into two groups, A and B, so that their cooking skill levels would be the same. Then, the participants cooked as they usually do and cooked using the proposed app. In order to suppress the bias caused by the order of the experiments, Group A cooked as usual and then cooked using the proposed app, while Group B cooked using the proposed app and then cooked as usual. The participants check the cooking procedures on a paper basis when they cook as usual, and check them using the app when they cook using the proposed app. Both paper and app show the same tasks, but the app also shows the cooking tools that the task requires as shown in Figure 6.

We conduct our evaluation through quantitative and qualitative evaluations. In the quantitative evaluation, we confirm that the user can reduce the cooking time by using the proposed app. Here, the cooking time for each cooking task is essentially unchanged regardless of whether the proposed app is used or not, because the tasks are exactly the same. In reality, however, the total cooking time varies greatly due to habituation and fatigue, and it is necessary to compare the total cooking time after considering these biases. Therefore, we recorded the time taken for each cooking task to determine the change in time between the same cooking tasks. For the qualitative evaluation, we confirm the impact of the proposed app on the user. Specifically, we confirm (1) the level of fatigue, (2) the sense of urgency and pressure, (3) the progress as planned, (4) the ease of cooking, and (5) the quality of the dish.

5.2.2. Results

As a result of the experiment, we observed that there were two types of participants who behaved differently from what we expected when cooking as usual. The first was the participants who parallelized the cooking tasks that were not assumed. For example, we assumed that participants could not perform other cooking tasks while they were cooking "grill hamburger steak on both sides" based on the recipe. However, several participants performed other cooking tasks during this cooking task. The second is the participants who rarely do the washing task when they cook as usual. These participants used a sponge to wash the cooking tools when using the proposed app. On the other hand, when they were cooking as usual, they used the cooking tools for other cooking tasks without rinsing them with water or washing them at all. As these actions greatly affect the cooking time, it is necessary to evaluate the results considering their effect.

Tables 5 and 6 show the experimental results for each participant in groups A and B. Participants with unexpected parallel cooking tasks and those who did not wash the cooking tools are marked, and there were four participants in each case. The columns "cooking as usual" and "using app" show the actual measured times for all the cooking without considering the bias. The column "change in cooking task" shows the increase or decrease between the total cooking time for each cooking task when cooking with the app and the total cooking time when cooking as usual. Increase or decrease of up to 20 min can be observed due to habituation and fatigue. The column "substantial change in cooking time" is the change in total cooking time considering habituation and fatigue,

and is obtained by subtracting the cooking as usual and the change in cooking task time from the using app. The column "proportion of substantial change in cooking time" is the proportion of "substantial change in cooking time" to "cooking as usual", and represents the percentage reduction in cooking time that takes into account the bias when using the app. We have achieved a time reduction of up to 13 min (14.8% of cooking as usual) by using the proposed app (ID = 1 in Table 5). Overall, the substantial cooking time of half of the participants was reduced.

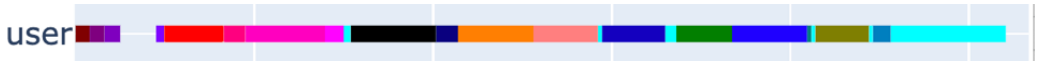
Table 5. Result of experiment for each user (Group A).

ID	Unexpected Parallelism	No Washing	Cooking as Usual	Using App	Change in Cooking Task	Substantial Change in Cooking Time	Proportion of Substantial Change in Cooking Time
1			1:29:30	1:18:23	+2:06	−**13:13**	−**14.8**%
3	✓	✓	1:04:52	1:03:56	−8:46	+9:44	+15.0%
5	✓		1:11:56	0:54:21	−20:40	+3:05	+4.29%
6			1:48:08	1:48:25	+10:43	−10:26	−9.46%
7			1:52:32	1:35:45	−6:13	−10:34	−9.39%

Table 6. Result of experiment for each user (Group B).

ID	Unexpected Parallelism	No Washing	Cooking as Usual	Using App	Change in Cooking Task	Substantial Change in Cooking Time	Proportion of Substantial Change in Cooking Time
4	✓	✓	0:57:13	1:06:05	+5:57	+2:55	+5.09%
8			1:10:16	1:06:45	+1:07	−4:38	−6.59%
9	✓	✓	1:19:42	1:26:15	+0:23	+6:10	+7.80%
10		✓	1:21:11	1:41:27	+16:52	+3:24	+4.19%
11			1:31:24	1:29:50	+5:24	−6:58	−7.62%

Figure 9 shows the usage of the resource "user" when the participant (ID = 1) cooks. Each horizontal bar represents a cooking/washing task. Horizontal bars of the same color common in Figure 9a,b represent the same cooking task except for the light blue color. The light blue horizontal bar represents the washing task. From Figure 9, we find that there is a time when the user is not doing anything when cooking as usual, but when using the app, the user is always performing a task.



(**a**) Resource usage when cooking as usual

(**b**) Resource usage when cooking with the app

Figure 9. Resource usage (ID = 1).

Table 7 shows the results of the qualitative evaluation conducted on each participant after the experiment. The use of the app reduced the level of fatigue by 0.7 points and the sense of urgency and pressure by 1.4 points. In addition, on-time progress and ease of cooking increased by 1.4 points each. On the other hand, the quality of the dish decreased by 0.3 points.

Table 7. Questionnaire results on cooking and app.

Questionnaire	Cooking as Usual	Using App
fatigue [1:very low, 5:very high]	3.7	3.0
urgency and pressure [1:very low, 5:very high]	3.1	1.7
progress as planned [1:low satisfaction, 5:high satisfaction]	3.1	4.5
ease of cooking [1:very difficult, 5:very easy]	3.4	4.8
quality [1:low satisfaction, 5:high satisfaction]	3.9	3.6

5.2.3. Discussion

From Tables 5 and 6, half of the participants (ID = 3, 4, 5, 9, 11) increased the substantial total cooking time. We believe that this is due to the unexpected parallelization and almost no washing tasks when cooking as usual. As the proposed app does not perform unsearchable parallelization or necessary washing, it is considered inappropriate for comparison. On the other hand, using the proposed app properly reduced the substantial total cooking time of half of the experimental participants (ID = 1, 6, 7, 8, 11). This is because our proposed system assigned parallel cooking and washing tasks to the gaps that were not utilized when cooking as usual (Figure 9). We believe that the proposed app is effective because it reduces the actual cooking time by a maximum of 13 min (14.8%) and an average of 9 min (9.5%).

From Table 7, the proposed app (1) reduces the level of fatigue, (2) reduces the sense of urgency and pressure, (3) helps to keep on schedule, and (4) makes cooking easier. This can be attributed to the fact that the app gave more time to the participants because they did not have to think about the cooking procedure. We also received positive comments from the participants: "I did not feel urgency or pressure because I only had to follow the cooking procedure" and "I did not feel the stress of having to wash the cooking tools during the cooking process because they were already ready for use when I wanted to use them". Therefore, the proposed app not only reduces the cooking time but also makes cooking more comfortable.

On the other hand, the quality of the dish was higher when it was cooked as usual. This is because the optimal cooking procedure only aims at minimizing the total cooking time, and thus the completion time varies from recipe to recipe. In our experiment, there was a large difference between the time when the participants finished making the miso soup and the time when they finished making all the recipes, and this led to this evaluation because the miso soup became cold. We think this problem can be improved by presenting the cooking procedure that minimizes the total cooking time and has the smallest difference between the completion time of the dish that should be warm and the cooking time.

6. Conclusions

The purpose of this study is to create an optimal cooking procedure that minimizes the total cooking time from the start of cooking to the end of cooking, including the washing task. We considered the creation of the optimal cooking procedure as a task scheduling problem and proposed an algorithm that extends the existing optimization methods to include the cooking-specific properties. In evaluation experiments, we showed that the proposed algorithm can reduce the execution time by up to 44% compared to the base PDF/IHS and increases the execution time by about 900 times even when the number of required searches increases by 10,000 times (usefulness), and that a cooking procedure with an error rate of 1% compared to the optimal cooking procedure can be derived in about 10 s (practicality). As a result of the quantitative evaluation through the experiment with three recipes for 10 participants each, the proposed application reduced by up to 13 min (14.8% of the time when users cooked freely) compared to the time when users cooked freely. In addition, as a result of qualitative evaluation through the questionnaire, the proposed application reduced fatigue, impatience, and pressure, made keeping on schedule and cooking easier. From this, we concluded that the proposed algorithm and application are effective.

Author Contributions: Conceptualization, J.N., T.M., H.S. and K.Y.; data curation, J.N.; formal analysis, J.N.; funding acquisition, T.M., H.S. and K.Y.; investigation, J.N.; methodology, J.N.; project administration, J.N.; resources, T.M., H.S. and K.Y.; software, J.N.; supervision, T.M., H.S. and K.Y.; validation, J.N.; visualization, J.N.; writing—original draft, J.N.; writing—review and editing, J.N., T.M., H.S. and K.Y. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Informed Consent Statement: Informed consent was obtained from all subjects involved in the study.

Conflicts of Interest: The authors declare no conflicts of interest. The founding sponsors had no role in the design of the study; in the collection, analyses, or interpretation of the data; in the writing of the manuscript; or in the decision to publish the results.

References

1. Cookpad Inc. Available online: https://cookpad.com/ (accessed on 19 December 2020).
2. Allrecipes.com Inc. Available online: https://www.allrecipes.com/ (accessed on 19 December 2020).
3. Kasahara, H.; Itoh, A.; Tanaka, H.; Itoh, K. A parallel optimization algorithm for minimum execution-time multiprocessor scheduling problem. *Syst. Comput. Jpn.* **1992**, *23*, 54–65. [CrossRef]
4. Matsushima, Y.; Funabiki, N. Practices of Cooking-Step Scheduling Algorithm for Homemade Cooking. In Proceedings of the 2015 IIAI 4th International Congress on Advanced Applied Informatics, Washington, DC, USA, 12–16 July 2015; pp. 500–505.
5. Van Laarhoven, P.J.; Aarts, E.H. Simulated annealing. In *Simulated Annealing: Theory and Applications*; Springer: Berlin/Heidelberg, Germany, 1987; pp. 7–15.
6. Yamabuki, T.; Ono, N.; Nagata, Y. Modeling of the Simultaneous Table Scheduling Problem and Dynamic Scheduling for this Problem. *Soc. Instrum. Control Eng.* **2018**, *54*, 346–356. [CrossRef]
7. Kimura, Y.; Shimizu, K.; Tsuboi, T.; Hasegawa, D.; Ishikawa, K.; Kimura, K.; Tanaka, M.; Ozeki, K.; Zhou, J.; Shigeno, M. An approach to cooking process scheduling for a family restaurant. *J. Adv. Mech. Des. Syst. Manuf.* **2018**, *12*, JAMDSM0076. [CrossRef]
8. Zhou, J.; Tsuboi, T.; Hasegawa, D.; Ishikawa, K.; Kimura, K.; Tanaka, M.; Ozeki, K.; Shigeno, M. Design and Validation of Rules for a Cooking Process Scheduling Model. *IPSJ Trans. Math. Model. Appl.* **2018**, *11*, 63–74.
9. Ishino, C.; Morimoto, N.; Yamada, T. Disjunctive Constraints Using Integer Range for Food Preparation Scheduling Considering Machine Type. *Spec. Interest Group Tech. Rep. IPSJ* **2020**, *2020*, 1–6.
10. Coffman, E.G.; Bruno, J.L. *Computer and Job-Shop Scheduling Theory*; John Wiley & Sons: Hoboken, NJ, USA, 1976.
11. Hu, T.C. Parallel sequencing and assembly line problems. *Oper. Res.* **1961**, *9*, 841–848. [CrossRef]
12. Kasahara, H.; Narita, S. Practical multiprocessor scheduling algorithms for efficient parallel processing. *IEEE Trans. Comput.* **1984**, *33*, 1023–1029. [CrossRef]
13. Fernandez, E.B.; Bussell, B. Bounds on the number of processors and time for multiprocessor optimal schedules. *IEEE Trans. Comput.* **1973**, *100*, 745–751. [CrossRef]
14. Nakamura, A.; Maekawa, Y. A Reduction Algorithm of Branching Nodes for Solving Task Scheduling Problems. *Inf. Process. Soc. Jpn. PRO* **2014**, *7*, 1–9.
15. Matsuse, H.; Nakamura, A.; Tominaga, H.; Maekawa, Y. Reducing the Number of Search Nodes in Task Scheduling Problem Using Hash Table of DF/IHS Method. In Proceedings of the 78th National Convention of IPSJ, Kanagawa, Japan, 10 March 2016; Volume 2016, pp. 197–198.
16. Matsuse, H.; Nakamura, A.; Tominaga, H.; Maekawa, Y. A Sppedup Method for PDF/IHS by Reducing of Branching Nodes in Task Scheduling Problems. *IPSJ Trans. Adv. Comput. Syst.* **2018**, *11*, 17–26.
17. Shibuya, T.; Kurita, K.; Kai, M. B-017 Evaluation of a Parallelized Branch and Bound Method for the Task Scheduling Problem Considering Communication Overhead. *Forum Inf. Technol.* **2014**, *13*, 149–154.
18. Utsunomiya, M.; Shioda, R.; Kai, M. Heuristic search based on branch and bound method for task scheduling considering communication overhead. In Proceedings of the 2011 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing, Victoria, BC, Canada, 23–26 August 2011; pp. 256–261.
19. Development of Parallelized Solver for Task Scheduling Problems with Communication Delays: Improvements of Search Effectiveness Using Optimal Scheduling of Sub-Task-Graphs. Available online: http://hdl.handle.net/10928/813 (accessed on 19 December 2020).
20. Detecting Hierarchically Structured Macro-Tasks within a Task Graph and Its Task Scheduling Method. Available online: http://hdl.handle.net/10928/1133 (accessed on 19 December 2020).
21. Garey, M.R. Computers and Intractability: A Guide to the Theory of Np-Completeness. *Rev. Esc. Enferm. Usp* **1979**, *44*, 340.
22. Ramamoorthy, C.V.; Chandy, K.M.; Gonzalez, M.J. Optimal scheduling strategies in a multiprocessor system. *IEEE Trans. Comput.* **1972**, *100*, 137–146. [CrossRef]