An indirect search algorithm for disaster restoration with precedence and synchronization constraints

Kira, Akifumi
Institute of Mathematics for Industry, Kyushu University

Iwane, Hidenao
Knowledge Information Processing Laboratory, Fujitsu Laboratories Ltd.

Anai, Hirokazu
Knowledge Information Processing Laboratory, Fujitsu Laboratories Ltd.

Kimura, Yutaka
Faculty of Systems Science and Technology, Akita Prefectural University

他

出版情報：9 (7), pp.1-15, 2017-07-05. Springer
バージョン：

Kira *et al. Pacific Journal of Mathematics for Industry* (2017) 9:7
DOI 10.1186/s40736-017-0032-5

Pacific Journal of Mathematics for Industry

ORIGINAL ARTICLE Open Access

An indirect search algorithm for disaster restoration with precedence and synchronization constraints

Akifumi Kira[1*], Hidenao Iwane[2], Hirokazu Anai[2], Yutaka Kimura[3] and Katsuki Fujisawa[1,4]

Abstract

When a massive disaster occurs, to repair the damaged part of lifeline networks, planning is needed to appropriately allocate tasks to two or more restoration teams and optimize their traveling routes. However, precedence and synchronization constraints make restoration teams interdependent of one another, and impede a successful solution by standard local search. In this paper, we propose an indirect local search method using the product set of team-wise permutations as an auxiliary search space. It is shown that our method successfully avoids the interdependence problem induced by the precedence and synchronization constraints, and that it has the big advantage of non-deteriorating perturbations being available for iterated local search.

Keywords: Mathematical optimization, Scheduling, Vehicle routing problem, Indirect local search, Linear extensions

1 Introduction

In the wake of the Great East Japan Earthquake, hands-on research and development is now required to solve concrete social problems. When a massive disaster such as an earthquake or a typhoon occurs, some damage to service networks is unavoidable (e.g., electricity, water service, gas and so on). To repair the damage to the networks as fast as possible, planning is needed to appropriately allocate tasks to two or more restoration teams and optimize their traveling routes. Such a scheduling problem is a variant of the vehicle routing problem (VRP) which designs optimal delivery or collection routes from one or several depots to a number of geographically scattered customers. There is a wide variety of VRPs and a broad literature on this class of problems.

The first systemic studies of this problem using VRPs in the context of Japan, appears to be Watanabe et al. [9, 10]. In these case studies, the problem is formulated as a VRP with time windows (VRPTW) and standard local search heuristics are then applied. However, Yamashita et al. [11] point out that establishing scheduling techniques for handling precedence and synchronization constraints has become a very important issue. If electricity is supplied to spot j via spot i, restoration teams cannot start operating spot j until the restoration of spot i is completed. Moreover, several teams may participate in restoring a damaged spot. The main difficulty in dealing with these constraints is that restoration teams are not independent of one another. In other words, a change in one route may affects other routes and may render all other routes infeasible. This is not usually the case in VRPs. This difficulty, which often impedes a successful solution by standard local search, is called the interdependence problem. See Drexl [4] for a complete review of this problem and a comprehensive collection of other types of synchronization constraints imposed in VRPs.

In the literature, Bredström and Rönnqvist [2] as well as Afifi et al. [1] study a variant of VRPTW in which some customers require simultaneous visits by two (or more) vehicles, with motivation to apply to elderly health care services. In our case, we note that how many teams participate in restoration of each spot is not a constraint but a decision variable for optimizing the whole restoration schedule.

A good technique several authors propose for overcoming the interdependence situation is the use of indirect search (Derigs and Döhmer [3], Li et al. [6], Nonobe and

*Correspondence: kira@imi.kyushu-u.ac.jp
[1]Institute of Mathematics for Industry, Kyushu University, 744 Motooka, Nishi-ku, 819-0395 Fukuoka, Japan
Full list of author information is available at the end of the article

Ibaraki [7], and others). In indirect search, given an optimization problem, we first define (i) an auxiliary search space $\mathcal{X}^{\text{aux}}$, which is different from the solution space $\mathcal{X}$ of the original problem, (ii) a decoder $f : \mathcal{X}^{\text{aux}} \to \mathcal{X}$, and (iii) search techniques and move types operating on the auxiliary space. Then, a local search procedure is executed not on $\mathcal{X}$ but on $\mathcal{X}^{\text{aux}}$ with evaluating search points by the decoder f. Though the computational cost of the decoder is added, it is known that this approach is powerful for problems having complicated constraints. Derigs and Döhmer [3] show that the approach of indirect local search is not only flexible and simple but when applied to the VRP with pickup and delivery and time windows (VRPPDTW) it also gives results which are competitive with state-of-the-art VRPPDTW-methods by Li and Lim [5], as well as Pankratz [8].

In this paper, we propose an enhanced indirect search method which uses the product set of team-wise permutations as an auxiliary search space, though many successful indirect search algorithms use permutations as an auxiliary search space (in the context of VRP, it is permutations of customers to be served). It is shown that our method successfully avoids the interdependence problem induced by the precedence and synchronization constraints. In addition, when we improve the capability of our algorithm to escape from local optima by using a metaheuristic technique called iterated local search, we have the big advantage of non-deteriorating perturbations being available.

In Section 2, after describing the main feature of our problem, we define our notation and formulate the mathematical optimization problem. We also summarize the difficulties with the interdependence problem in our setting. In Section 3, we develop an enhanced indirect search algorithm that avoids the interdependence problem induced by the precedence and synchronization constraints. In Section 4, we perform computational experiments in a practical setting using a geographic information system (GIS) and geospatial information. The results obtained from numerical examples in this study are then described.

2 The case study

2.1 Problem description

We first enumerate the main feature of our problem.

2.1.1 Precedence relation between pairs of damaged spots

If electricity is supplied to spot j via spot i, then it is necessary to restore spot i before restoring spot j. Namely, a restoration team cannot start operating spot j, but they may wait at spot j until the restoration of spot i is completed. We denote this by $i \prec j$. Notice that $(\mathcal{V}, \prec)$ is a strictly partially ordered set (or strict poset), where $\mathcal{V}$ is the set of damaged spots. In our application, we can assume that the Hasse diagram of the poset is a directed forest (see Fig. 8 in Section 4). Moreover, the whole damaged area is divided into multiple areas in advance, and each restoration team has their area of responsibility. For any pair (i,j), $i \prec j$ implies that spots i and j are located in the same area.

2.1.2 Areas of responsibility

Each restoration team should give priority to its area of responsibility. Namely, they cannot move to another area until their own area is completed. These are interpreted as team-wise precedence constraints, and compatible with the precedence constraints. We use $i \prec_k j$ to denote that spot i is in team k's area of responsibility but spot j is in another area, and $(\mathcal{V}, \prec_k)$ is also a strict poset. The number of restoration teams is greater than the number of areas. Hence two or more teams may be allocated in a common area.

2.1.3 Synchronized restoration by two or more teams

A spot can be repaired by two or more restoration teams if each team can help decrease the restoration time of the spot by at least c minutes (otherwise, the spot is repaired by one team). Each team can join the restoration of a spot during the restoration. However, they must stay until the restoration is completed. We set $c = 20$ in our case study.

2.1.4 Min-max objective

The goal is to ensure that entire restoration process is completed as quickly as possible. In other words, we minimize the time required for the team that takes the longest time to finish their restoration work and return to the depot.

2.2 A MILP formulation

To accurately formulate our problem as a mathematical optimization problem, we first need to define our notation.

- Let $\mathcal{V} \cup \{0\} = \{0, 1, \ldots, |\mathcal{V}|\}$ be the set of all spots to be repaired and a depot. "0" represents the depot.
- Let $\mathcal{K} = \{1, 2, \ldots, |\mathcal{K}|\}$ denote the set of all restoration teams.
- Let $\mathcal{P}$ be the set of all triplets (i,j,k) such that team k should respect the precedence constraint between spots i and j. Namely,

$$\begin{aligned} \mathcal{A} &:= \{(i,j) \mid i \prec j,\ i,j \in \mathcal{V}\}, \\ \mathcal{A}_k &:= \{(i,j) \mid i \prec_k j,\ i,j \in \mathcal{V}\}, \quad k \in \mathcal{K}, \\ \mathcal{P} &:= \mathcal{A} \times \mathcal{K} \cup \bigcup_{k \in \mathcal{K}} \{(i,j,k) \mid (i,j) \in \mathcal{A}_k\}. \end{aligned}$$

- c is the threshold for the synchronized restoration: Two or more teams can restore the same spot if each team can help decrease the restoration time of the spot by at least c minutes.
- d_{ij} is the time required to move from spot i to spot j.

- w_i represents the volume of work required to complete the restoration of spot i.
- x_{ij}^k denotes the binary integer decision variable that equals 1 if and only if team k moves directly from spot i to spot j.
- y_i^k denotes the binary integer decision variable that equals 1 if and only if team k visits spot i.
- b_i^k represents the time point at which team k begins to work at spot i. The value of b_i^k is meaningful only if $y_i^k = 1$.
- e_i denotes the time point at which the restoration of spot i is completed, where all teams leave the depot to start the restoration process at $e_0 = 0$.
- t is the time point at which the whole restoration process is completed.
- M is a real number sufficiently large, which is used in the so-called big-M method.

Using the above notation, the desired optimization problem is formulated as a mixed integer linear programming (MILP) problem.

$$\begin{aligned}
\text{Minimize} \quad & t \\
\text{subject to} \quad & e_i + d_{i0} \le t, \quad \forall i \in \mathcal{V}, && \text{(1a)}\\
& y_i^k = \sum_{j \in \mathcal{V}} x_{ij}^k, \quad \forall i \in \mathcal{V} \cup \{0\},\ \forall k \in \mathcal{K}, && \text{(1b)}\\
& y_j^k = \sum_{i \in \mathcal{V}} x_{ij}^k, \quad \forall j \in \mathcal{V} \cup \{0\},\ \forall k \in \mathcal{K}, && \text{(1c)}\\
& e_i \le b_j^k + M\left(1 - y_j^k\right), \quad \forall (i,j,k) \in \mathcal{P}, && \text{(1d)}\\
& e_i + d_{ij} \le b_j^k + M\left(1 - x_{ij}^k\right), \\
& \forall i, \forall j \in \mathcal{V} \cup \{0\},\ \forall k \in \mathcal{K}, && \text{(1e)}\\
& b_i^k \le e_i, \quad \forall i \in \mathcal{V},\ \forall k \in \mathcal{K}, && \text{(1f)}\\
& e_i - b_i^k \le M y_i^k, \quad \forall i \in \mathcal{V},\ \forall k \in \mathcal{K}, && \text{(1g)}\\
& \sum_{k \in \mathcal{K}} \left(e_i - b_i^k\right) = w_i, \quad \forall i \in \mathcal{V}, && \text{(1h)}\\
& c \times \sum_{g \in \mathcal{K} \setminus \{k\}} y_i^g \le e_i - b_i^k + M\left(1 - y_i^k\right), \\
& \forall i \in \mathcal{V},\ \forall k \in \mathcal{K}, && \text{(1i)}\\
& x_{ij}^k \in \{0,1\}, \quad \forall i, \forall j \in \mathcal{V} \cup \{0\},\ \forall k \in \mathcal{K}, && \text{(1j)}\\
& y_i^k \in \{0,1\}, \quad \forall i \in \mathcal{V} \cup \{0\},\ \forall k \in \mathcal{K}, && \text{(1k)}\\
& 0 \le b_i^k < \infty, \quad \forall i \in \mathcal{V} \cup \{0\},\ \forall k \in \mathcal{K}, && \text{(1l)}\\
& 0 \le e_i < \infty, \quad \forall i \in \mathcal{V} \cup \{0\}, && \text{(1m)}\\
& 0 \le t < \infty. && \text{(1n)}
\end{aligned}$$

From the objective function and constraint (1a), we minimize the time required for the team that takes the longest time to finish its restoration work and return to the depot. Constraints (1b) and (1c) mean that if team k visits spot i, then team k must move to spot i from another exactly once, and must move somewhere from spot i exactly once. The precedence constraints and the areas of responsibility are considered in (1d). Travel time is considered in (1e). If team k visits spot j next to spot i, i.e., $x_{ij}^k = 1$, then time interval d_{ij} is required for traveling between them. Notice that $e_0 = 0$ is automatically satisfied because of the minimization of the objective function. Constraint (1f) represents that the time team k starts restoring spot i is before the time when it is completed. Notice that (1e) and (1f) eliminate subtours that do not include the depot. Moreover, if team k does not visit spot i, i.e., $y_i^k = 0$, it follows from (1f) and (1g) that $e_i - b_i^k = 0$. Thus, constraint (1h) ensures that the restoration of spot i is not completed until the total working time of the restoration teams working there exceeds w_i. From constraint (1i), we have

$$y_i^k = 1, \quad \sum_{g \in \mathcal{K} \setminus \{k\}} y_i^g \ge 1 \Longrightarrow \frac{e_i - b_i^k}{\sum_{g \in \mathcal{K} \setminus \{k\}} y_i^g} \ge c \qquad (2)$$

for every $i \in \mathcal{V}$ and every $k \in \mathcal{K}$. Therefore, constraint (1i) means that a spot may be served by two or more teams if each team can help decrease the restoration time by at least c minutes.

However, the problem has many big-M constraints, the structure of the traveling salesman problem (TSP), a min-max objective function, and symmetries between restoration teams. For this reason, it is thoroughly intractable to find a solution by using a general-purpose MIP solver.

2.3 Interdependence problem

Let us examine the interdependence problem in the setting laid out in Table 1.

Team 1's route, shown as in Fig. 1, does not break the given precedence constraints in itself. The same applies to team 2's route. However, taken as a whole, team 1's route and team 2's route do break the precedence constraint $5 \prec 6$. Synchronization at spot 8 is also incomplete, because it is prohibited to leave there until the restoration is completed. In other words, whether or not a route is feasible depends on the other routes.

In case (a), we can render both routes feasible by inserting waiting times into them (see Fig. 2). However, the modified schedule has a longer total duration. From the

Table 1 An instance with nine spots and two teams

	Area 1	Area 2
List of spots	$\{1, 2\}$	$\{3, 4, \ldots, 9\}$
List of teams	$\{1\}$	$\{2\}$
Precedence constraints	–	$5 \prec 6,\ 7 \prec 9$

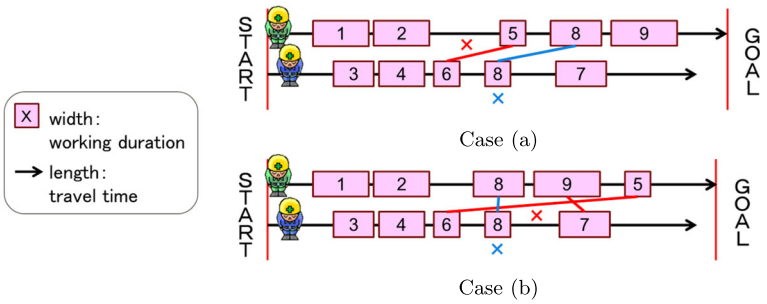

Fig. 1 Infeasible schedules

view point of optimization, this is not good. In case (b), by contrast, inserting waiting times never leads a feasible schedule.

3 Proposed method

In this section, we develop an enhanced indirect search algorithm that avoids the interdependence problem induced by the precedence and synchronization constraints. Figure 3 illustrates the concept of indirect search.

3.1 Linear extensions and search space

Let $(\mathcal{V}, \ll)$ be the transitive union of $(\mathcal{V}, \prec)$ and $(\mathcal{V}, \prec_k)$. Namely,

$$\{(i,j) \in \mathcal{V} \times \mathcal{V} \mid i \ll j\} = \mathcal{A} \cup \mathcal{A}_k.$$

Let $\mathcal{L}_k$ denote the set of all linear extensions of $(\mathcal{V}, \ll)$, where a linear extension $\sigma_k \in \mathcal{L}_k$ is a permutation of spots that is compatible with the precedence constraints and with the team-wise precedence constraints for team k's area of responsibility.

Definition 3.1 (linear extension) *A linear extension of a finite poset $(\mathcal{Q}, \prec)$ is a total ordering $\sigma = \sigma(1)\,\sigma(2)\ldots\sigma(|\mathcal{Q}|)$ of its elements such that $i \prec j$ whenever $\sigma(i) \prec \sigma(j)$.*

Now, we define the search space by

$$\mathcal{X}^{\mathrm{aux}} := \prod_{k \in \mathcal{K}} \mathcal{L}_k.$$

A search point $x \in \mathcal{X}^{\mathrm{aux}}$ is expressed by a $|\mathcal{K}| \times |\mathcal{V}|$ matrix. The following example is a search point for the problem shown in Table 1.

$$x = \begin{pmatrix} \sigma_1 \\ \sigma_2 \end{pmatrix} = \begin{pmatrix} \sigma_1(1) & \sigma_1(2) & \cdots & \sigma_1(9) \\ \sigma_2(1) & \sigma_2(2) & \cdots & \sigma_2(9) \end{pmatrix} = \begin{pmatrix} 1 & 2 & 3 & 5 & 7 & 6 & 8 & 9 & 4 \\ 3 & 7 & 5 & 6 & 8 & 4 & 9 & 1 & 2 \end{pmatrix} \tag{3}$$

Since the precedence constraints $5 \prec 6$ and $7 \prec 9$ are given, 5 and 7 are located to the left of 6 and 9 in either row, respectively. Moreover, team k should give priority to their area of responsibility. Hence spot 1 and spot 2 are located to the left of the other spots in the first row. Conversely, they are located to the right of the other spots in the second row.

Definition 3.2 (topological ordering) *A topological ordering of a directed acyclic graph (DAG) $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is a linear ordering of its vertices such that if there is a path from u to v, then u appears before v in the ordering.*

We note that $(\mathcal{V}, \ll)$ constitutes a DAG $\mathcal{G} = (\mathcal{V}, \mathcal{A} \cup \mathcal{A}_k)$. A linear extension of the poset is the same thing as a topological ordering of the DAG. We can obtain an initial search point $x \in \mathcal{X}^{\mathrm{aux}}$ by a topological ordering method.

3.2 Decoder

Let us define a decoder that relates each search point $x \in \mathcal{X}^{\mathrm{aux}}$ to a feasible schedule $f(x)$ in the following

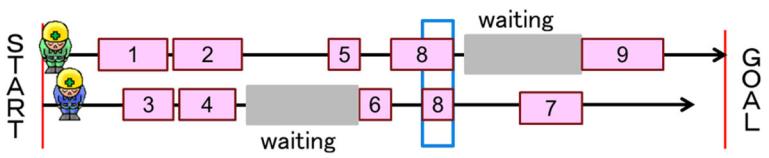

Fig. 2 Repair of an infeasible schedule by inserting waiting times

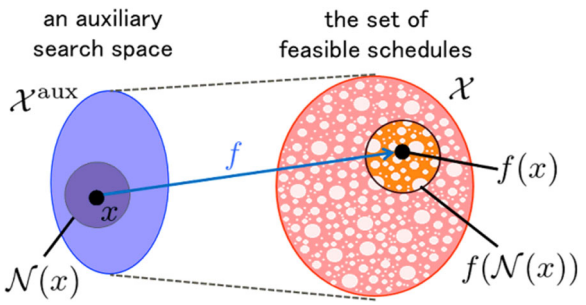

Fig. 3 Indirect search

manner. To determine which spot to visit after completing a restoration somewhere, team k examines each spot as a candidate in the order $\sigma_k(1), \sigma_k(2), \ldots, \sigma_k(|\mathcal{V}|)$. Team k is permitted to visit spot $\sigma_k(i)$ if no team has left for spot $\sigma_k(i)$, or if some team(s) has already left for spot $\sigma_k(i)$ but each team operating spot $\sigma_k(i)$ can decrease the restoration time by at least c minutes. Otherwise, team k skips spot $\sigma_k(i)$ and examines spot $\sigma_k(i+1)$. Figure 4 illustrates the feasible schedule $f(x)$, derived by repeating this process, for x shown as in (3). In x, the circled numbers represent the spots that are actually visited.

To show an implementation of our decoder, we introduce additional variables.

- s_k represents the state of team k.

$$s_k = \begin{cases} \text{Initial} & \text{Team } k \text{ leaves the depot now and begin the whole process} \\ \text{Moving} & \text{Team } k \text{ is moving to a spot or the depot} \\ \text{Working} & \text{Team } k \text{ is working at a spot} \\ \text{Waiting} & \text{Team } k \text{ is waiting at a spot} \\ \text{Completed} & \text{Team } k \text{ finished their task and arrived at the depot.} \end{cases}$$

- $v_k \in \mathcal{V} \cup \{0\}$ is the location of team k. If $s_k = \text{Moving}$, v_k represents the current destination. Otherwise, it represents the place they are currently at.
- p_k is an index variable that points to an element of σ_k.
- t_k represents the time until their current movement or work is completed.

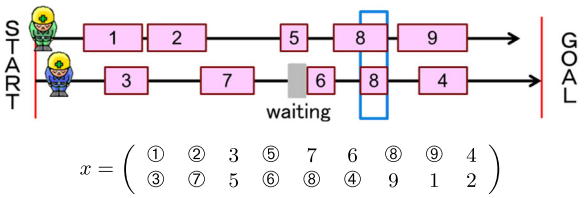

Fig. 4 Feasible schedule $f(x)$

Now, our decoder DECODER() can be implemented as follows:

Function DECODER(parameters: $x \in \mathcal{X}^{\mathrm{aux}}$)

Initialize $t \leftarrow 0$, $\mathcal{K}' \leftarrow \mathcal{K}$;
Initialize $w'_i \leftarrow w_i$, $\mathcal{K}_i \leftarrow \emptyset$ for all $i \in \mathcal{V}$;
Initialize $s_k \leftarrow$ Initial, $v_k \leftarrow 0$, $p_k \leftarrow 1$, $t_k \leftarrow 0$ for all $k \in \mathcal{K}$;
while $\mathcal{K}' \neq \emptyset$ **do**
  Execute **Algorithm 1**;
  Execute **Algorithm 2**;
return t;

Two subroutines are shown in Algorithms 1 and 2.

Algorithm 1: Subroutine for determining next action

forall the $k \in \mathcal{K}'$ such that $t_k = 0$ **do**
  switch the value of s_k **do**
    case Working or Initial /* $e_{v_k} = t$ */
      Declare a local integer variable i;
      Set $i \leftarrow v_k$;
      while $p_k \leq |\mathcal{V}|$, and VISITABLE(arguments: k, $\sigma_k(p_k)$) returns false **do**
        $p_k \leftarrow p_k + 1$;
      if $p_k \leq |\mathcal{V}|$ **then**
        $v_k \leftarrow \sigma_k(p_k)$, $\mathcal{K}_{v_k} \leftarrow \mathcal{K}_{v_k} \cup \{k\}$; /* $y^k_{v_k} = 1$ */
        $p_k \leftarrow p_k + 1$;
      else
        $v_k \leftarrow 0$; /* go back to the depot */
      $s_k \leftarrow$ Moving, $t_k \leftarrow d_{i,v_k}$; /* $x^k_{i,v_k} = 1$ */
    case Moving
      if $v_k = 0$ **then**
        $s_k \leftarrow$ Completed, $\mathcal{K}' \leftarrow \mathcal{K}' \setminus \{k\}$;
      else
        $s_k \leftarrow$ Waiting;
    case Waiting or Completed
      (Do nothing)
forall the $k \in \mathcal{K}'$ such that $s_k =$ Waiting **do**
  if the work at spot v_k can be started **then**
    /* We can judge this in $O(1)$ time under the assumption that the Hasse diagram of $(\mathcal{V}, \ll)$ is given as a directed forest */
    $s_k \leftarrow$ Working; /* $b^k_{v_k} = t$ */

Algorithm 2: Subroutine for updating t_k

forall the $k \in \mathcal{K}'$ such that $s_k =$ Working **do**
  $t_k \leftarrow w'_{v_k}/n_{v_k}$, where $n_i := \#\{g \in \mathcal{K}_i \mid s_g = \text{Working}\}$;
Declare a local floating-point variable $t_{\min}$;
$t_{\min} \leftarrow \min\{t_k \mid k \in \mathcal{K}'$ such that $s_k =$ Moving or Working$\}$;
forall the $k \in \mathcal{K}'$ such that $s_k =$ Moving or Working **do**
  $t_k \leftarrow t_k - t_{\min}$;
  if $s_k =$ Working **then**
    $w'_{v_k} \leftarrow w'_{v_k} - t_{\min}$;
$t \leftarrow t + t_{\min}$;

Function VISITABLE(parameters: $k \in \mathcal{K}$, $j \in \mathcal{V}$)

/* Decide whether team k can join the restoration of spot j. */
if $\mathcal{K}_j = \emptyset$ **then** /* no team has left for spot j */
  return true;
else
  /* Suppose that there is no additional waiting time at spot j due to the precedence constraints. */
  Declare local floating-point variables $\{\tau_g\}_{g \in \mathcal{K}_j \cup \{k\}}$ and set

$$\tau_g \leftarrow \begin{cases} d_{v_k,j} & \text{if } g = k \\ 0 & \text{if } g \neq k, s_g = \text{Waiting or Working} \\ t_g & \text{if } g \neq k, s_g = \text{Moving} \end{cases} \tag{4}$$

  for all $g \in \mathcal{K}_j \cup \{k\}$; /* team g starts operating spot j in τ_g minutes. */
  Sort $\{\tau_g\}_{g \in \mathcal{K}_j \cup \{k\}}$ as $\tau'_1, \tau'_2, \ldots, \tau'_m$ in such a way that $\tau'_1 \leq \tau'_2 \leq \cdots \leq \tau'_m$, where $m := |\mathcal{K}_j \cup \{k\}|$;
  Declare a local floating-point variable w;
  Set $w \leftarrow w'_j - \sum_{g=1}^{m-1} g \times (\tau'_{g+1} - \tau'_g)$;
  /* w represents the volume of remaining work when the last team arrives. Notice that each team can help decrease the restoration time by at least c if and only if the last arriving team can achieve it. */
  if $\frac{w}{m-1} - \frac{w}{m} \geq c$ **then**
    return true;
  else
    return false;

In **Function** VISITABLE, we decide whether team k can join the restoration of spot j under the assumption that there is no additional waiting time due to the precedence constraints, which is stricter than the actual constraint (1i). On the other hand, deciding it accurately with considering the waiting time is impractical. If there exists a spot i such that $i \prec j$, restoration teams cannot start operating spot j until the restoration of spot i is completed. Thus, we should replace (4) into

$$\tau_g \leftarrow \begin{cases} \max(e_i - t, d_{v_k,j}) & \text{if } g = k \\ \max(e_i - t, 0) & \text{if } g \neq k, s_g = \text{Waiting or Working} \\ \max(e_i - t, t_g) & \text{if } g \neq k, s_g = \text{Moving.} \end{cases}$$

In this case, the larger the value of e_i is, the more the last arriving team conceals their traveling time and helps decrease the restoration time. However, the final value of e_i cannot be found when **Function** VISITABLE (arguments: k, j) is called from line ♯ of Algorithm 1, because it will be improved if another team joins the restoration of spot i. Therefore, whether the team k can join the restoration of spot j depends on whether another team can join the restoration of spot i. In this situation, one possible way for deciding it accurately is to use a backtrack search method, but it will take an exponential time. Hence, our decision method assuming $e_i \leq t$ in advance is more suitable for efficiency.

Theorem 3.1 *If we consider the additional constraint*

$$\sum_{k \in \mathcal{K}} y_i^k \leq 3, \quad \forall i \in \mathcal{V},$$

then **Function** DECODER *runs in* $O(|\mathcal{K}||\mathcal{V}|)$ *time.*

Proof Algorithm 2 runs in $O(|\mathcal{K}|)$ time. Moreover, under the additional constraints, we have

$$\sum_{i \in \mathcal{V}} \sum_{k \in \mathcal{K}} y_i^k \leq 3|\mathcal{V}|.$$

This implies that the main while loop in **Function** DECODER is repeated $O(|\mathcal{V}|)$ times. Hence, it takes $O(|\mathcal{K}||\mathcal{V}|)$ time to execute the loop of Algorithm 2. On the other hand, Algorithm 1 does not necessarily run in $O(|\mathcal{K}|)$ time, because **Function** VISITABLE may be called repeatedly in line ♯. However, **Function** VISITABLE is called exactly once for each $(k, j) \in \mathcal{K} \times \mathcal{V}$ in the main while roop in **Function** DECODER. Therefore, it also takes $O(|\mathcal{K}||\mathcal{V}|)$ time to execute the loop of Algorithm 1. □

3.3 Neighbourhood

We shall define a move type operating on $\mathcal{X}^{\text{aux}}$.

Definition 3.3 (intra-swap and its feasibility) *For any* $\sigma_k = \sigma_k(1)\sigma_k(2)\cdots\sigma_k(|\mathcal{V}|) \in \mathcal{L}_k$, *an intra-swap move is*

a transposition $(i\ j)$ *that swaps* $\sigma_k(i)$ *and* $\sigma_k(j)$ *for some* $i, j \in \{1, 2, \ldots, |\mathcal{V}|\}$. *Namely,*

$$\sigma_k \circ (i\ j) = \sigma_k(1) \cdots \sigma_k(j) \cdots \sigma_k(i) \cdots \sigma_k(|\mathcal{V}|).$$

An intra-swap move is said to be feasible if $\sigma'_k := \sigma_k \circ (i\ j) \in \mathcal{L}_k$.

Lemma 3.1 *An intra-swap* $(i\ j)$ *operating a linear extension* $\sigma_k = \sigma_k(1)\sigma_k(2)\cdots\sigma_k(|\mathcal{V}|) \in \mathcal{L}_k$ *is feasible if and only if*

$$\begin{cases} \sigma_k(i) \not\prec \sigma_k(h) & i < h \le j, \\ \sigma_k(h) \not\prec \sigma_k(j) & i < h < j, \\ \sigma_k(i) \not\prec_k \sigma_k(j). \end{cases}$$

Proof The result follows directly from the definition of $\mathcal{L}_k$. □

From Lemma 3.1, we can judge whether an intra-swap move is feasible or not in $O(|\mathcal{V}|)$ time. We define the intra-swap neighbourhood $\mathcal{N} : \mathcal{X}^{\text{aux}} \to 2^{\mathcal{X}^{\text{aux}}}$ in the following manner:

$$\mathcal{N}(x) := \bigcup_{k \in \mathcal{K}} \{x' = (\sigma_{-k}, \sigma'_k) \mid \sigma'_k = \sigma_k \circ (i\ j) \in \mathcal{L}_k, \\ i, j = 1, 2, \ldots, |\mathcal{V}|\}, \\ x = (\sigma_1, \sigma_2, \ldots, \sigma_{|\mathcal{K}|}) \in \mathcal{X},$$

where $(\sigma_{-k}, \sigma'_k) := (\sigma_1, \ldots, \sigma_{k-1}, \sigma'_k, \sigma_{k+1}, \ldots, \sigma_{|\mathcal{K}|})$.

Suppose that $x = (\sigma_1, \sigma_2, \ldots, \sigma_{|\mathcal{K}|}) \in \mathcal{X}$ is the provisional solution. We first choose $k \in \mathcal{K}$, select $i, j \in \mathcal{V}$, and check whether the intra-swap $(i\ j)$ operating σ_k is feasible. If it is feasible, then the intra-swap move is applied to x and the new search point x' is evaluated by the decoder. If the objective value of $f(x')$ is better than that of $f(x)$, then we update the provisional solution by $x \leftarrow x'$. Figure 5 shows that the schedule is improved from $f(x)$ to $f(x')$ by swapping $\sigma_2(2)$ and $\sigma_2(6)$. A procedure for searching for a better schedule is shown in **Function** IndirectSearch.

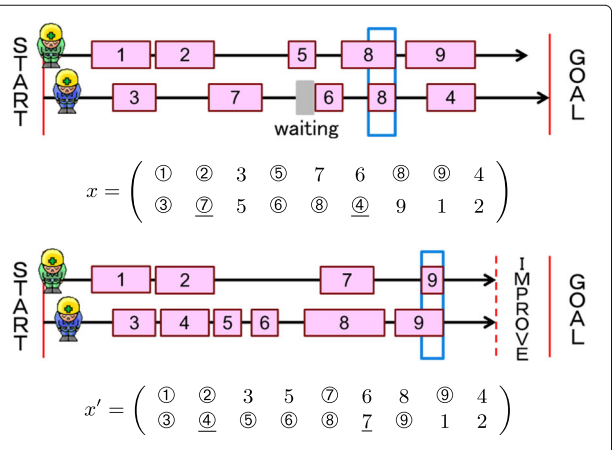

Fig. 5 An example of an improvement from an intra-swap operation

Function INDIRECTSEARCH(parameters: $x \in \mathcal{X}^{\text{aux}}$)

Declare floating-point variables $v_{\min}$ and v;
$v_{\min} \leftarrow$ Call DECODER(arguments: x);
♮ **forall the** $x' \in \mathcal{N}(x)$ **do**
  $v \leftarrow$ Call DECODER(arguments: x');
  if $v < v_{\min}$ **then**
    $v_{\min} \leftarrow v$, $x \leftarrow x'$;
    go to Line ♮;
return x;

3.4 Iterated indirect local search

In this subsection, we improve the capability of our algorithm to escape from local optima by using a metaheuristic technique. In multi-start local search, a number of initial solutions are generated, and local search procedure is applied to each of them. Finally, the best solution obtained in the entire search is output. In iterated local search, which is an effective variant of multi-start local search, the initial solutions for local search are generated by perturbating good solutions found in the previous search. On this point, we have the big advantage of non-deteriorating perturbations being available.

Definition 3.4 (value-invariant swap) *An intra-swap move is called to be value-invariant if it converts* $x \in \mathcal{X}^{\text{aux}}$ *to* $x' \in \mathcal{N}(x)$ *such that the objective values of* $f(x)$ *and* $f(x')$ *are the same each other.*

Our iterated indirect local search algorithm is summarized as follows.

Function ITERATEDSEARCH(parameters: $x \in \mathcal{X}^{\text{aux}}$, $\bar{m} \in \mathbb{Z}_+, \bar{n} \in \mathbb{Z}_+$)

for $m \leftarrow 1$ **to** $\bar{m}$ **do**
  for $n \leftarrow 1$ **to** $\bar{n}$ **do**
♭ Choose an value-invariant feasible intra-swap randomly, and add it to x;
  $x \leftarrow$ Call INDIRECTSEARCH(arguments: x);
return x;

4 Computational results

To test our algorithms in a practical setting, we generate a virtual instance on a real map. A depot and 57 damaged spots are located in Minami ward, Fukuoka city, Fukuoka prefecture, Japan. The authors depict them in Fig. 6. The volumes of work $\{w_i\}$ and their precedence constraints are depicted in Fig. 7 and in Fig. 8, respectively. We

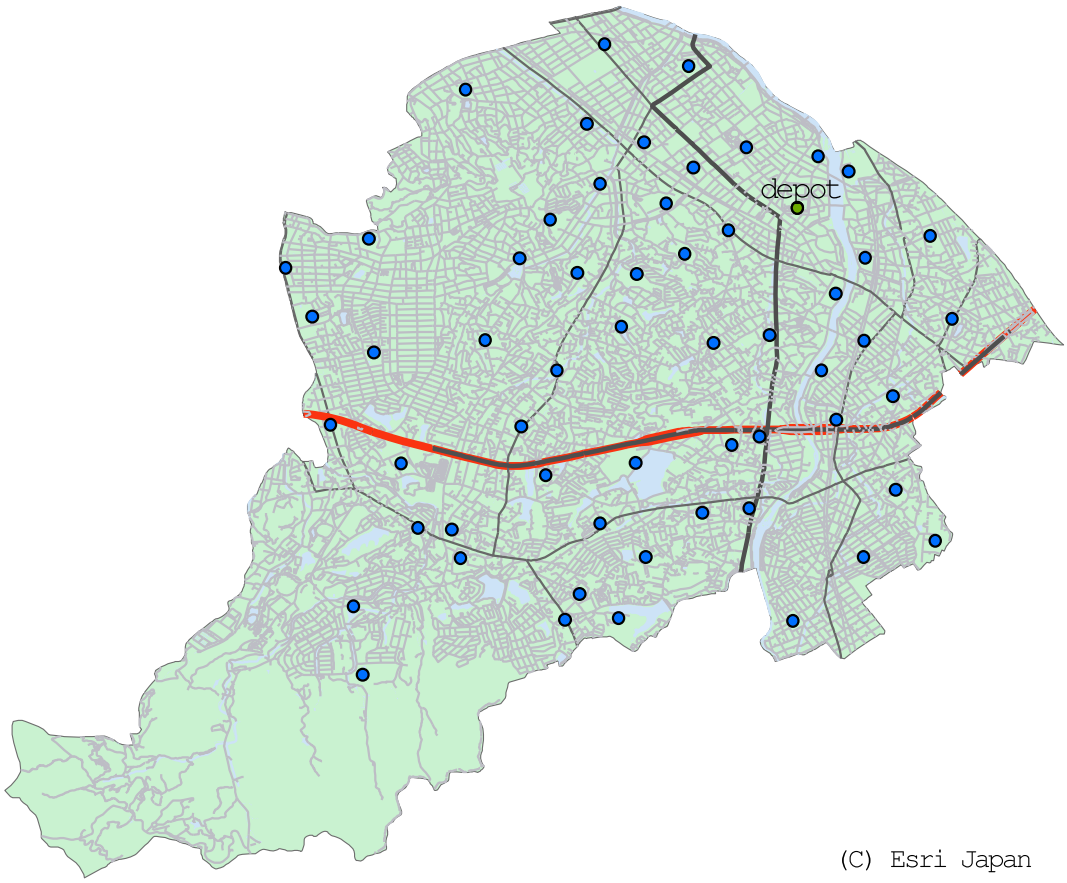

Fig. 6 Locations of damage spots

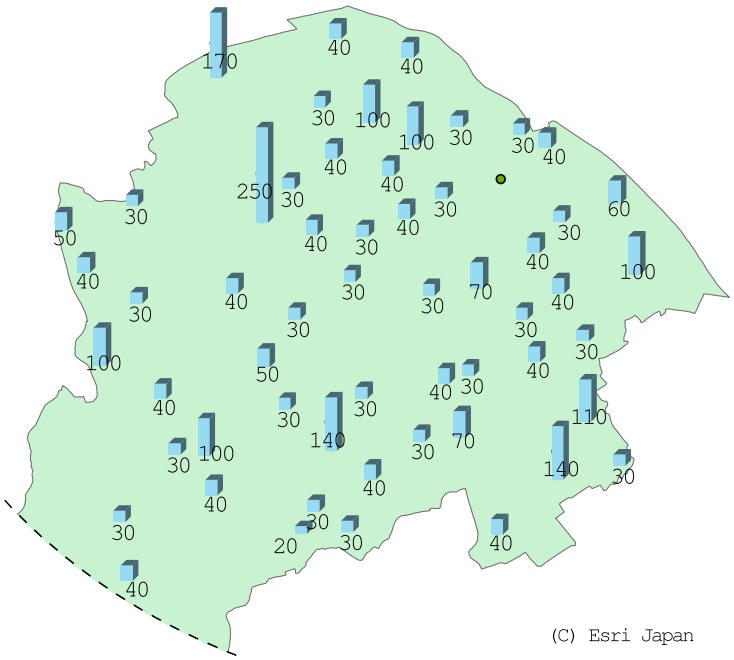

Fig. 7 The volumes of work $\{w_i\}$

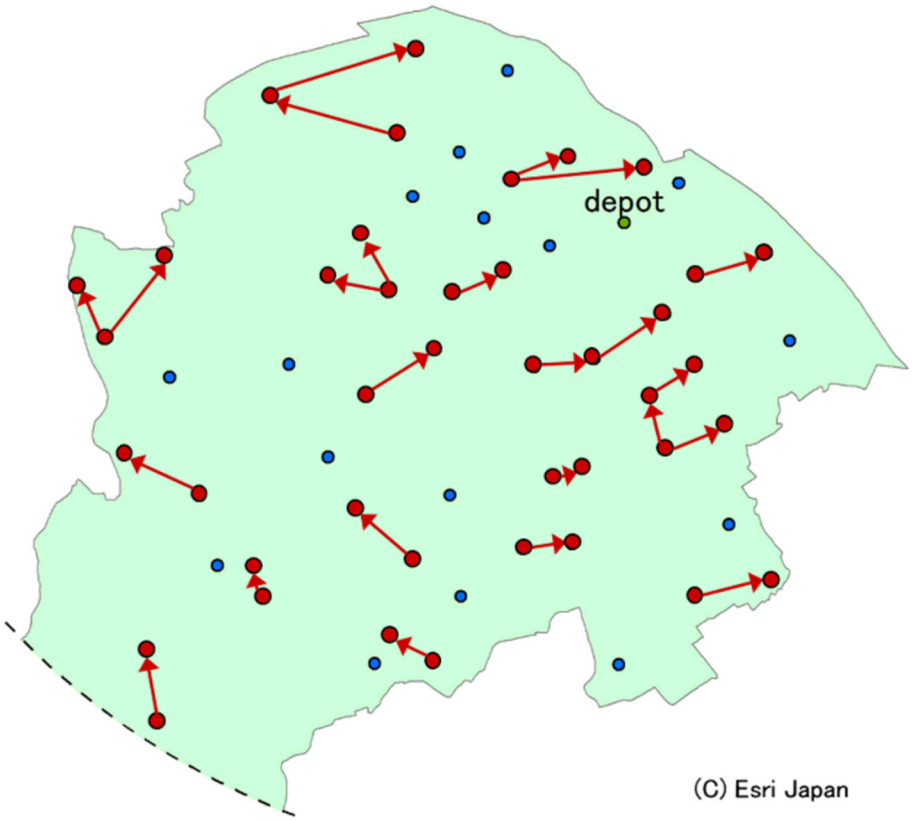

Fig. 8 Hasse diagram of $(\boldsymbol{\mathcal{V}}, \prec)$

suppose that there are eight restoration teams in total, and that their areas of responsibility are the whole damaged area. In other words, we do not consider team-wise precedence constraints[1]. For each pair (i, j) in $(\mathcal{V} \cup \{0\}) \times (\mathcal{V} \cup \{0\})$, we set d_{ij} to be the minimum travel time (min) from i to j along the real road network. The regulations of traffic such as one-way roads and restricted turns are also taken into consideration. To do this, we solve the shortest path problem, in advance, by using GIS software ArcGIS for Desktop ver. 10.4 (Esri Inc., USA) and ArcGIS Data Collection Road Network 2015 (Esri Japan Corporation).

In our computational experiments, we first generate a topological ordering σ of the Hasse diagram of $(\mathcal{V}, \prec)$, and use $x_0 = (\sigma, \sigma, \ldots, \sigma) \in \mathcal{X}^{\mathrm{aux}}$ as an initial search point. Then we execute **Function** IteratedSearch with arguments: $(x, \bar{m}, \bar{n}) = (x_0, 1000, 1000)$ repeatedly with changing the seed for random numbers used in line ♭ of the function. We implemented the algorithm using C++ Language and executed it on a desktop PC with Intel® Core™ i7-3770K processor of 3.4 GHz and 16 GB memory installed. The results are shown in Table 2 and Fig. 9. Routes 1 to 8 output by trial C are depicted in Figs. 10, 11, 12, 13, 14, 15, 16 and 17, respectively. The spots that are served by two or more teams are highlighted in yellow.

We note that our auxiliary search space, the product set of team-wise permutations (linear extensions) is larger than the set of permutations that is used in the usual indirect search. Therefore, it seems reasonable that our algorithm should be able to find a good sub-optimal solution. Estimating the optimization gap is quite hard in our challenging problem, but Fig. 9 clearly shows that our

Table 2 Computational results

	Objective value (min)		CPU time (sec) [# of calls to Function DECODER]		
Seed	Initial sol.	Final sol.	1st iteration	1000 iterations	Final update
A	655.889	452.000	0.298 [11,691]	71.739 [8,253,466]	33.754 [3,853,235]
B	↓	451.000	0.678 [40,122]	75.119 [8,397,206]	61.158 [6,799,271]
C	↓	449.000	0.187 [7,199]	74.202 [8,312,991]	45.523 [5,110,908]

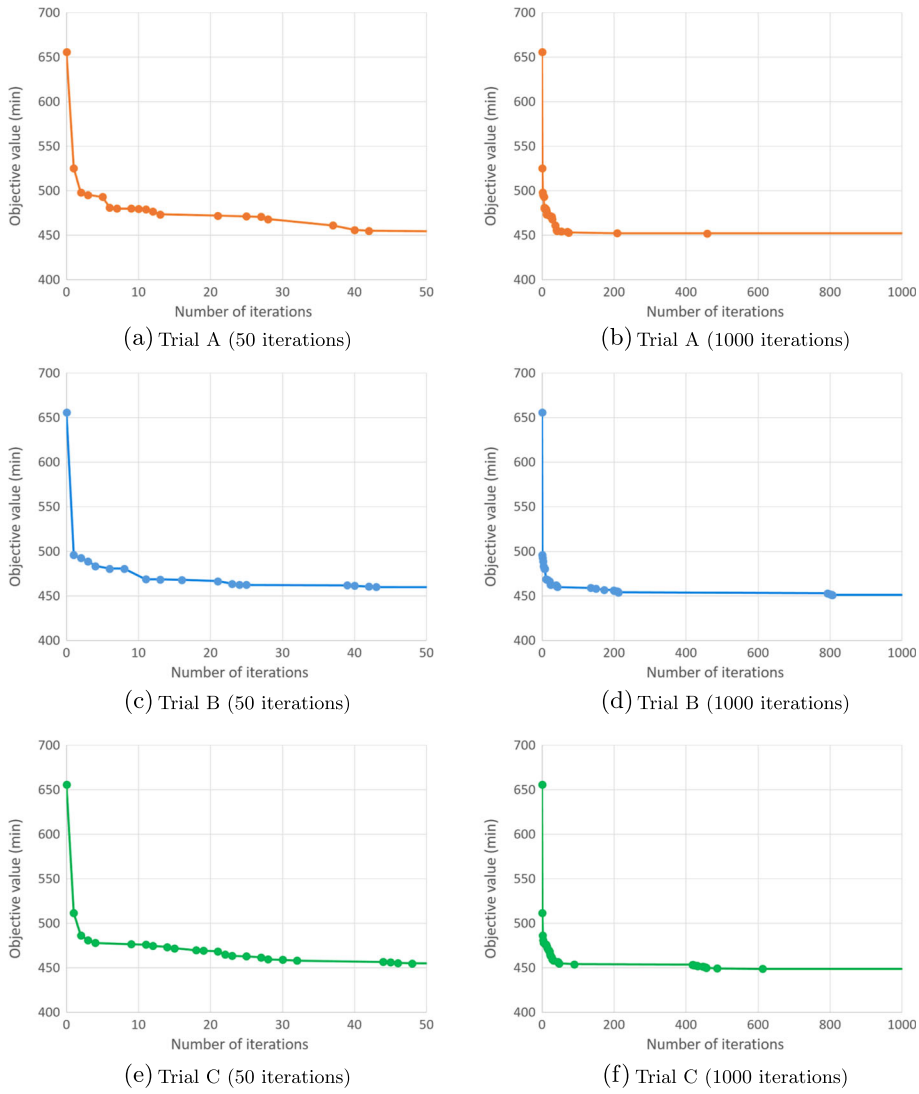

Fig. 9 Effect of the iterated local search

indirect search has a high local search ability and that its iteration with non-deteriorating perturbations is useful to escape from local optima. The final solution, found in trial C, achieved a 13% decrease of the objective value, compared with the output by an existing solver developed in Fujitsu group[2].

In each trial, 1000 iterations are finished in about 75 s, and **Function** DECODER is called more than eight million times. This computational efficiency is mainly because of the $O(|\mathcal{K}||\mathcal{V}||)$ implementation of the decoder. It would be able to quickly present the most recent plan that responds to changing conditions, including the extent of an area affected by a disaster and the pace of recovery work.

The authors think that the large auxiliary search space, the linear time implementation of the decoder, the iterated local search using non-deteriorating perturbations are responsible for the successful results.

5 Conclusion and future work

Based on the concept of the indirect search, we have proposed a simple local search method using the product set of team-wise permutations as an auxiliary search space. We have demonstrated that this new method successfully avoids the interdependence problem induced by the precedence and synchronization constraints, and that it has the big advantage of non-deteriorating perturbations being available for iterated local search. The authors believe that this approach will also be useful for other scheduling problems. We would like to investigate applications to other problems as challenges in the future.

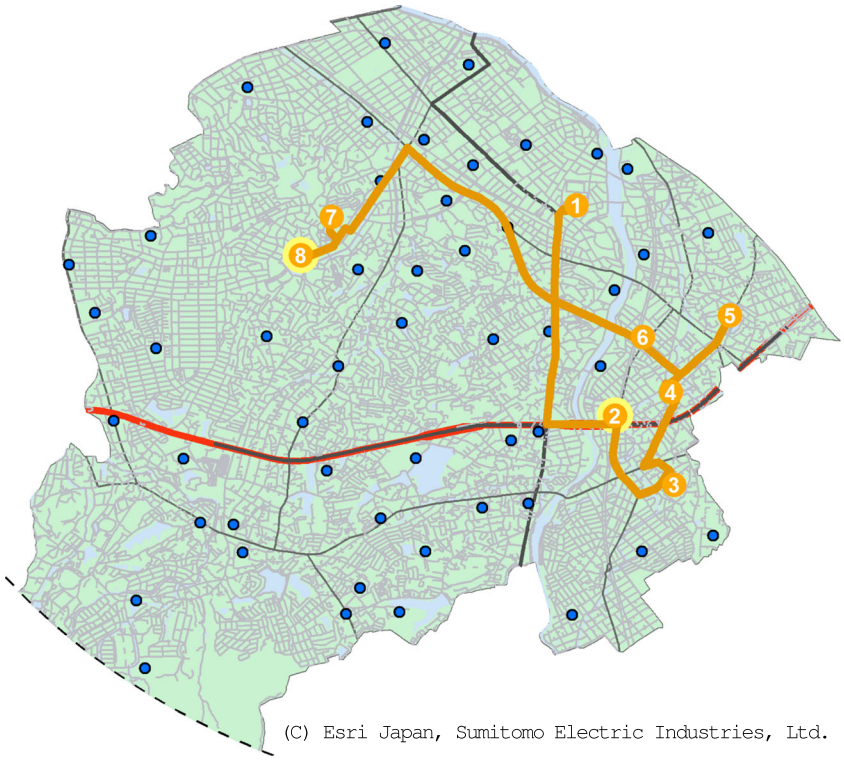

Fig. 10 Route 1

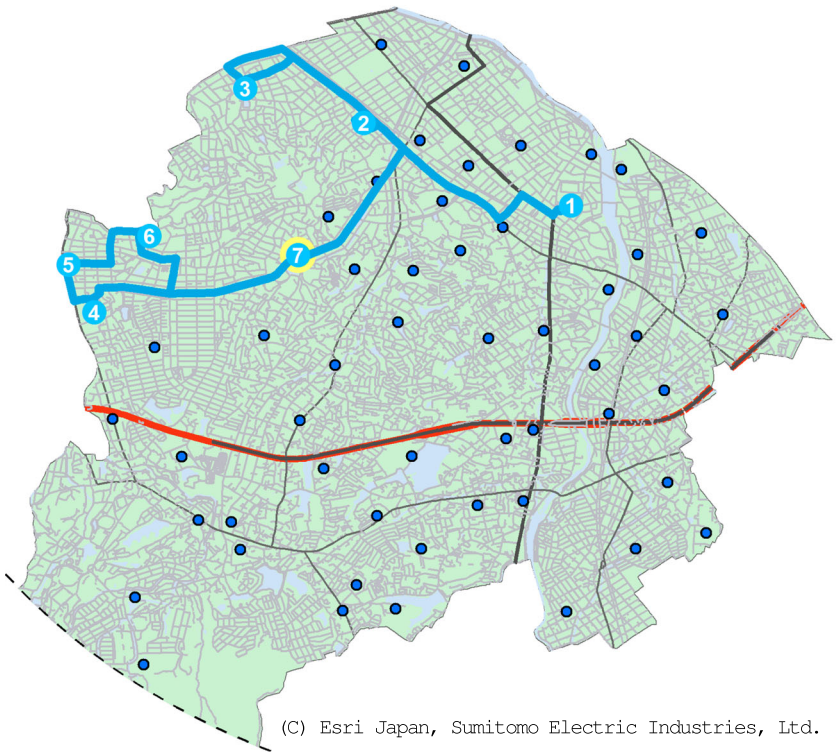

Fig. 11 Route 2

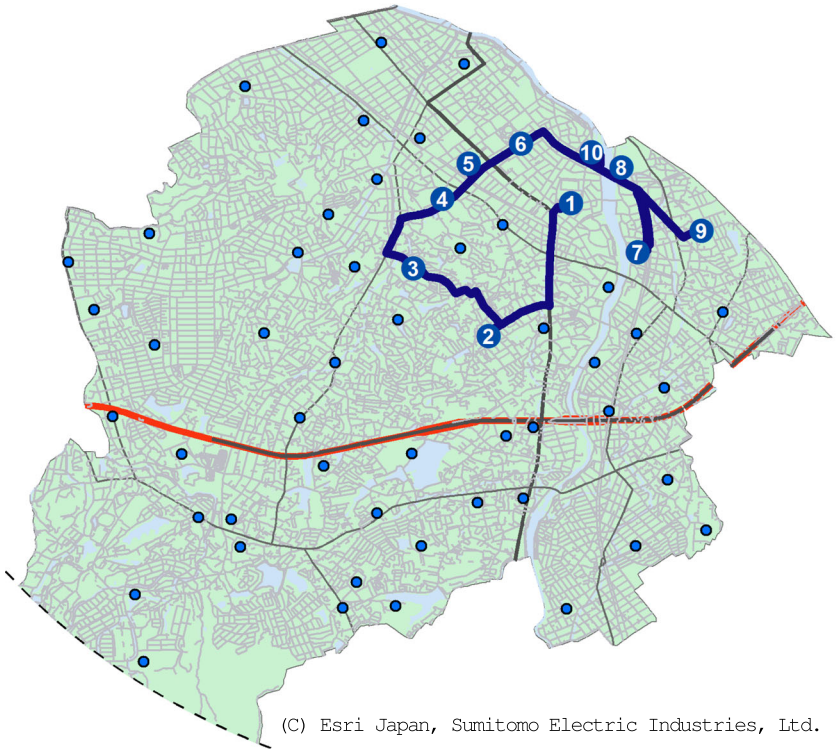

Fig. 12 Route 3

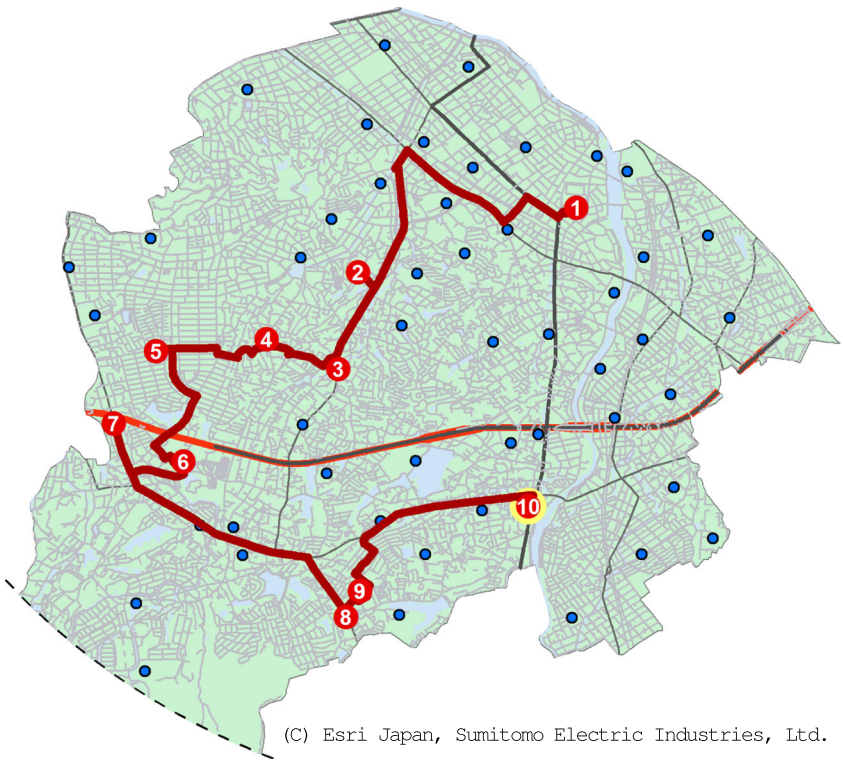

Fig. 13 Route 4

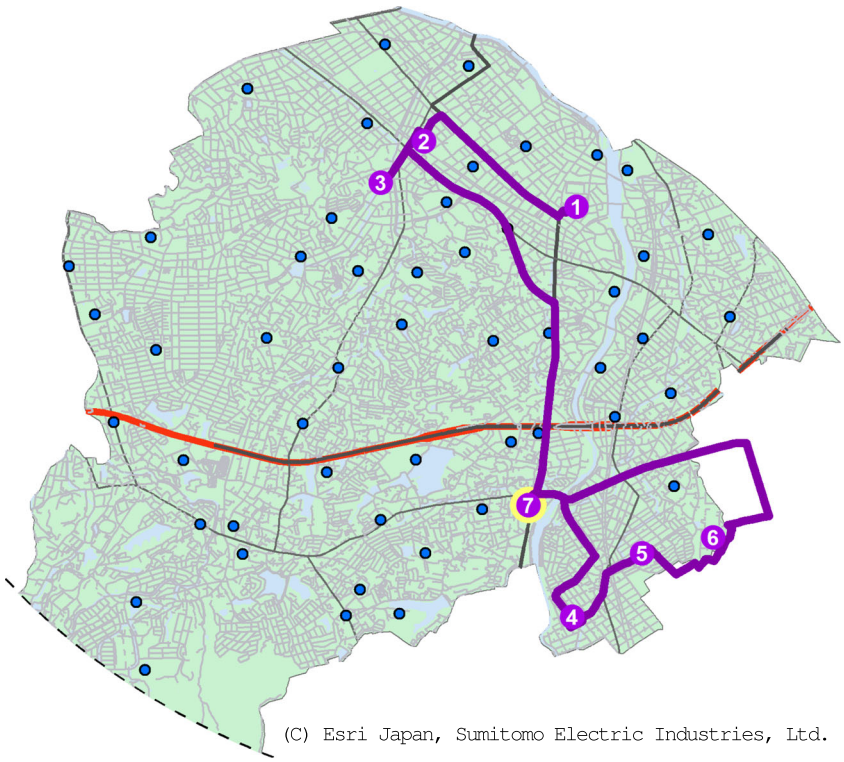

Fig. 14 Route 5

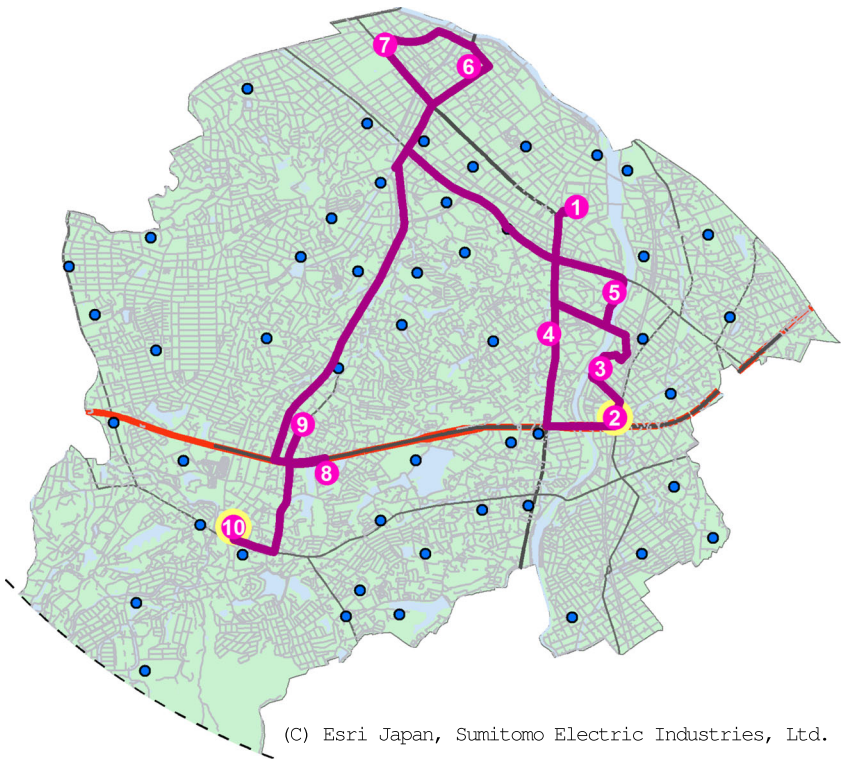

Fig. 15 Route 6

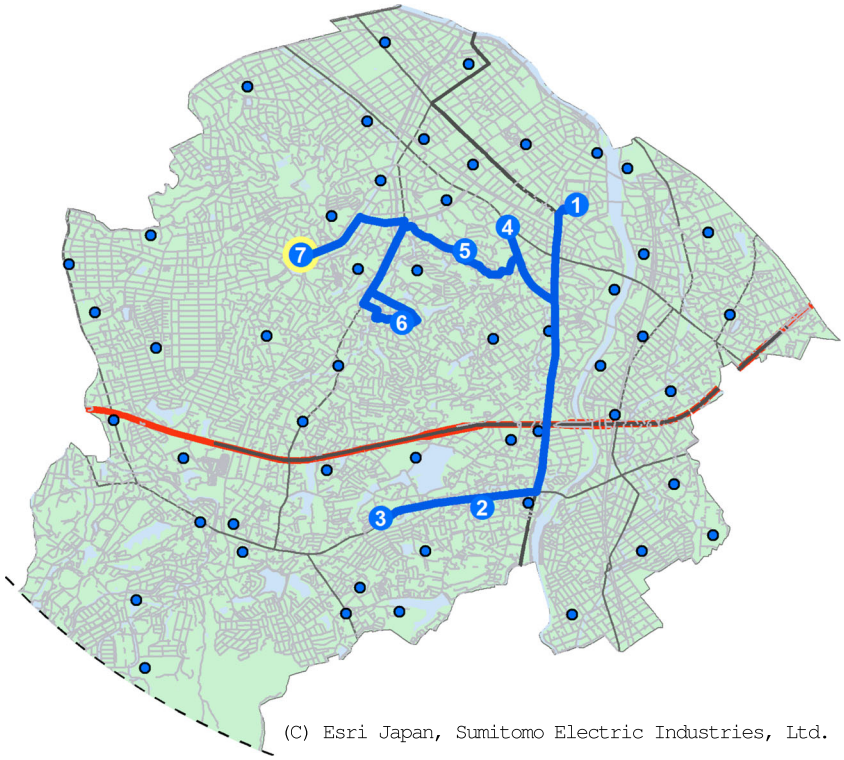

Fig. 16 Route 7

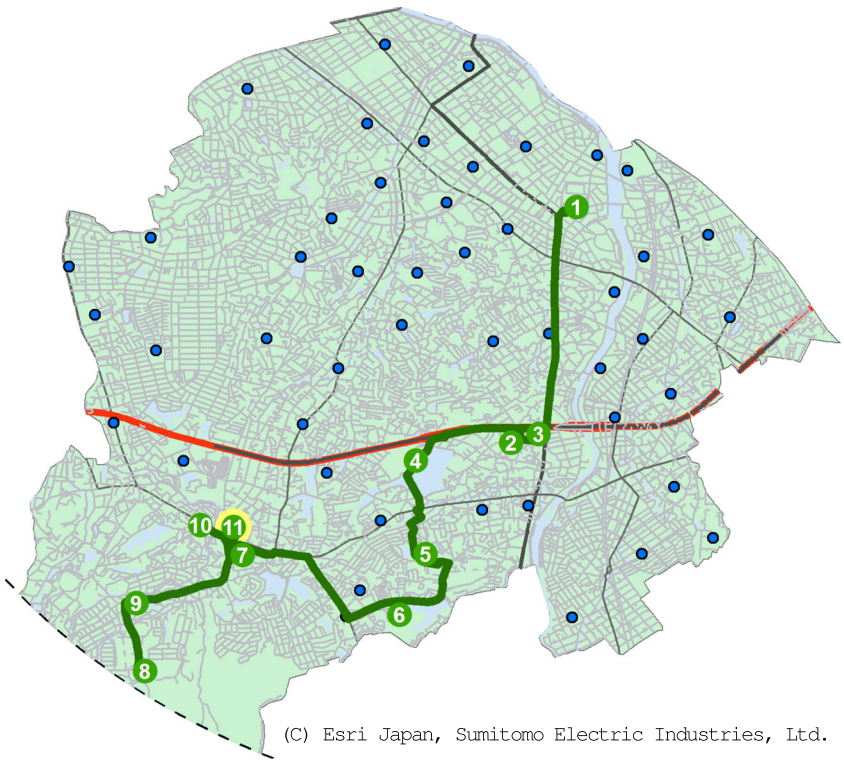

Fig. 17 Route 8

Endnotes

[1] Rather, this situation is computationally harder for our algorithm, because team-wise precedence constraints reduce the size of $\mathcal{N}(x)$ for every $x \in \mathcal{X}^{\text{aux}}$.

[2] This existing method is a greedy construction method, though the detail cannot be publishable due to business secret.

Acknowledgements
The authors wish to thank Dr. Hidefumi Kawasaki for leading us to this joint research. We are deeply grateful to Dr. Shunji Umetani for his valuable advice regarding this investigation. Akifumi Kira was supported in part by JSPS KAKENHI Grant Number 26730010 and the Joint Research Fund of Fujitsu Limited. Yutaka Kimura was supported by JSPS KAKENHI Grant Number 26400207. Katsuki Fujisawa was supported by JST CREST and JSPS KAKENHI Grant Number 16H01707.

Publisher's Note

Author details
[1] Institute of Mathematics for Industry, Kyushu University, 744 Motooka, Nishi-ku, 819-0395 Fukuoka, Japan. [2] Knowledge Information Processing Laboratory, Fujitsu Laboratories Ltd., 4-1-1 Kamikodanaka, Nakahara-ku, 211-8588 Kawasaki, Japan. [3] Faculty of Systems Science and Technology, Akita Prefectural University, 84-4 Aza Ebinokuchi, Tsuchiya, 015-0055 Yurihonjo, Akita, Japan. [4] JST CREST, 4-1-8 Honcho, Kawaguchi, 332-0012 Saitama, Japan.

Received: 12 October 2016 Revised: 23 March 2017 Accepted: 9 May 2017
Published online: 05 July 2017

References

1. Afifi, S, Dang, D-C, Moukrim, A: A simulated annealing algorithm for the vehicle routing problem with time windows and synchronization constraints. In: Learning and Intelligent Optimization, LNCS **7997**, pp. 259–265. Springer, (2013). http://www.springer.com/la/book/9783642449727
2. Bredström, D, Rönnqvist, Mikael: Combined vehicle routing and scheduling with temporal precedence and synchronization constraints. Eur. J. Oper. Res. **191**(1), 19–31 (2008). Elsevier
3. Derigs, U, Döhmer, T: Indirect search for the vehicle routing problem with pickup and delivery and time windows. OR Spectrum **30**(1), 149–165 (2008)
4. Drexl, M: Synchronization in vehicle routing-a survey of vrps with multiple synchronization constraints. Transp. Sci. **46**(3), 297–316 (2012)
5. Li, H, Lim, A: A metaheuristic for the pickup and delivery problem with time windows. Intl. J. Artif. Intell. Tools **12**(02), 173–186 (2003)
6. Li, Y, Lim, A, Rodrigues, B: Manpower allocation with time windows and job-teaming constraints. Nav. Res. Logist. (NRL) **52**(4), 302–311 (2005)
7. Nonobe, K, Ibaraki, T: Formulation and tabu search algorithm for the resource constrained project scheduling problem. In: Essays and Surveys in Metaheuristics, pp. 557–588. Springer, (2002). http://www.springer.com/la/book/9780792375203
8. Pankratz, G: A grouping genetic algorithm for the pickup and delivery problem with time windows. Or Spectrum **27**(1), 21–41 (2005)
9. Watanabe, I, Kenichi, T, Satoshi, U: An optimal rout planning method for emergency restoration work in natural disasters. CRIEPI Research Report (R07025) (2008). http://criepi.denken.or.jp/jp/kenkikaku/report/detail/R07025.html
10. Watanabe, I, Kenichi, T, Satoshi, U: An optimal allocation method of restoration teams for emergency restoration work. CRIEPI Research Report (R08008) (2009). http://criepi.denken.or.jp/jp/kenkikaku/report/detail/R08008.html
11. Yamashita, M, Funakoshi, M, Ishii, H, Kashiwagi, T, Moki, M: Advanced approaches to respond to emergency disasters in kyushu electric's distribution department. The papers of Technical Meeting on Power Systems Engineering. IEE Japan, 143 (2012). (in Japanese without abstract). https://ieej.ixsq.nii.ac.jp/ej/?active_action=repository_view_main_item_detail&page_id=13&block_id=1%208&item_id=55303&item_no=1