令和 5 年度　学士論文

時間制約付き carrier vehicle routing 問題に対する二次錐計画問題を用いた解法

東京工業大学　情報理工学院　数理・計算科学系
学籍番号　19B31048
脇田康平
指導教員 山下 真 教授

2023 年 2 月 27 日

目次

1 はじめに

ドローン（無人航空機）は、搭乗者がいなくても空中を飛行し荷物を積載できるといった特徴をもつため、有人では高難度であった農業、測量などといった幅広い分野にも利用が広がっている。さらに、人手不足や交通渋滞で困難に直面している物流業界においてもその利用が期待されている。

一方、ドローンは空中を飛行するという特性上、大容量のバッテリーを積むことができず、小型機の場合は一度の飛行時間が一時間にも満たないような場合がある点などデメリットも持っている。そこで、物流業界などにおいては距離の離れている倉庫と目的地を直接ドローンを用いて荷物を輸送せず、トラックなどの輸送機器にドローンと荷物を載せ、目的地付近でドローンを飛ばして輸送することが検討されている。この方法では、市街地などの駐車スペースが限られるような現実の問題に応用しやすい利点もあり、また、ドローンを飛ばしていない時間 (つまりトラックでドローンを移動させている間) に充電をすることも可能である。

これ以降ドローンを一般化したものを vehicle, 輸送トラックを一般化したものをそれぞれ carrier と呼ぶ。それぞれの主な性質を以下に示す。

vehicle 速度が速く直接目的地に向かうのに用いる。しかしバッテリー量に限界があり一回の航行可能距離には上限が存在する。そのため、目的地に近づくまでは carrier によって運ばれる。バッテリー量は carrier と合流すると上限まで戻る。飛行限界までのフライト時間に比較するとバッテリー交換に要する時間は短時間であるため、本論文では [4, 12] と同様にバッテリー交換の時間を 0 であると仮定する。

carrier 速度が遅いため直接目的地に向かうのに用いず、離陸地点まで vehicle を輸送したのち着陸地点まで移動し vehicle を回収する用途で用いる。移動距離に上限は存在しない。

以後 vehicle と carrier が分離することを離陸、vehicle と carrier が合流することを着陸と呼ぶ。

vehicle と carrier を用いた配送計画問題は既に様々な研究者の研究対象となっている。vehicle と carrier を併用して配送に用いる場合、通常の一種類の輸送機器のみで行う配送計画問題と異なり、vehicle の離着陸地点を決定する必要がある。離着陸可能座標の候補が予め与えられた場合、つまり離散変数となる場合は Luo ら [15] や Lin ら [13]、Park ら [16] などを含め、様々な研究が存在している。

本論文で対象とするような配送計画問題は、ユークリッド平面上の任意の座標で離着陸が可能であるような設定であり座標は連続変数として現れる。このような問題は、海上での船とヘリコプターなどが応用例として考えられ [17]、carrier-vehicle travel salesman problem (CVTSP) とも呼ばれている。CVTSP は、Gambella ら [5] のモデルでは混合整数二次錐計画問題 (mixed-integer second-order cone programming, MISOCP) として定式化されており、branch-and-cut アルゴリズムなどで厳密解を求めることができる [20]。CVTSP は TSP を一般化した問題とも捉えられるため、計算量のクラスとしては NP 困難なクラスに属している。したがって、一般に短時間で

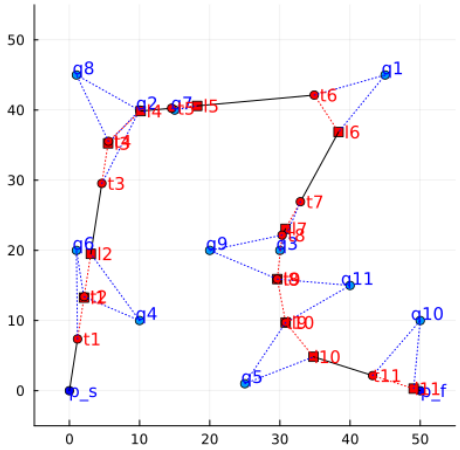


図 1　ソルバーで CVTSP を説いた例

の厳密の求解は難しく、ヒューリスティクスを適用して近似解を求める手法も研究されている。Larasati ら [12] の研究では焼きなまし法をベースにした手法を用いている。この問題には荷物に応じたバッテリー消費量や速度変化などを考慮したモデルなども存在し、一方で Klauco ら [11] の研究では、vehicle の一度のフライの制限範囲内であれば複数以上の目的地を連続して訪れることができるモデルを定式化している。

CVTSP の数理モデルに着目すると、Garone ら [6] では、事前に目的地をたどる順番を与えられたときの vehicle の離着陸地点を決める問題が定式化されている。その後、目的地も離着陸点の決定と同時に求められるモデルが Gambella ら [4] の研究によって示された。ここで示されたモデルを用いて CVTSP を実際に解いてみると、図 1 のような経路が得られる。この図では、船は p_s からスタートしている。vehicle は i 番目 $(i = 1, \ldots, 11)$ のフライトで赤丸の t_i で船を離陸、目的地 $q_1, \ldots, q_{11}$ のいずれか 1 か所を訪問して、赤四角の l_i で船に戻っている。最終的に船は vehicle を回収して p_f に至っている。しかし、これらのモデルでは各目的地での時間制限は考慮されていない。

本論文では、目的地のそれぞれが配達の時間制限を持つ CVTSP を定義し、焼きなまし法と呼ばれるヒューリスティック手法を用いて高速で良好な近似解を構築する。提案手法では、この問題を目的地をたどる順番を求める問題と、vehicle の離着陸地点を求める問題の 2 つに分割し、前者は離散最適化問題であるためヒューリスティック手法を用いて決定する。後者は連続最適化問題であり、二次錐計画問題として定式化した後に内点法に基づく CPLEX というソルバーを用いて決定した。

数値実験では、対象とする問題を直接 MISOCP として定式化して CPLEX で求めた手法と提案手法との比較を行った。目的地数が 17 以下の場合の誤差が 10% 程度であるが、実行時間は最大で 40 分の 1 に減少した。

本論文の構成は次のようになっている。第 2 節で本論文で用いる焼きなまし法と MISOCP の概略と 2 つの先行研究について説明し、第 3 節では時間制約付き CVTSP (CVTSP with time

window, CVTSPTW) 問題を定義して MISOCP として定式化する。第 4 節では CVTSPTW への数値計算手法を提案し、第 5 節では生成した CVTSPTW 問題のインスタンスを用いて数値実験を行い、その結果に対する考察を述べる。最後に第 6 節で本論文のまとめと今後の課題について述べる。

1.1 記法

この論文で扱う記法を述べる。

- $\|\cdot\|$ はユークリッドノルムとする。
- 上付き添え字 T は行列やベクトルの転置を示す。
- $\boldsymbol{x}^T\boldsymbol{y}$ はベクトル $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$ の内積 $\sum_{i=1}^n x_i y_i$ である。
- ２つのベクトル $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$ に対して、Hadamard 積 $\boldsymbol{x} \circ \boldsymbol{y}$ は成分ごとの積 $[\boldsymbol{x} \circ \boldsymbol{y}]_i = x_i \cdot y_i \ (i = 1, \ldots, n)$ により定義される。

2 先行研究

この節では本論文で用いている事前知識について説明する。まず、数理最適化問題のクラスの一つである混合整数二次錐計画問題 (MISOCP) を説明し、CVTSP を MISOCP としてモデル化した Gambella らの手法を紹介する。次に、メタヒューリスティックの一種である焼きなまし法を説明し、この手法を用いて CVTSP を解いた研究である Larasati らの手法を紹介する。

2.1 混合整数二次錐計画問題

二次錐計画問題はノルムを扱えるため汎用性が高いだけでなく、内点法を用いて効率的に求解をすることができるという特徴がある [18]。SOCP はロバスト線形計画問題や、ロバスト最小二乗法やアレイ・アンテナの重量設計、ロボットアームの握力の最適化といった応用例がある [14]。本節の定義は [2] を参考にしている。

まず、n 次元の二次錐 $\mathcal{K}$ を

$$\mathcal{K} = \begin{cases} (z_0, \bar{z}) \in \mathbb{R} \times \mathbb{R}^{n-1} : z_0 \geq ||\bar{z}|| & (n > 1) \\ z \in \mathbb{R} : z \geq 0 & (n = 1) \end{cases}$$

により定義する。二次錐計画問題 (second-order cone programming, 以下 SOCP) の一般的な形は

$$\begin{array}{rcl} \min & : & \boldsymbol{f}^T \boldsymbol{x} \\ \text{subject to} & : & \boldsymbol{F}\boldsymbol{x} = \boldsymbol{g},\ \boldsymbol{x} \in \mathcal{K} \end{array}$$

で定義できる。ここで、$\boldsymbol{f} \in \mathbb{R}^n, \boldsymbol{g} \in \mathbb{R}^p$ は定数ベクトルであり、$\boldsymbol{F} \in \mathbb{R}^{p \times n}$ は定数行列である。また、$\boldsymbol{x} \in \mathbb{R}^n$ が決定変数ベクトルである。複数の二次錐の直積が二次錐であること、および線形変換などを適用すると、SOCP はノルムを用いて以下のように表記することも可能である。

$$\begin{array}{rcl} \min & : & \boldsymbol{f}^T \boldsymbol{x} \\ \text{subject to} & : & ||\boldsymbol{A}_i \boldsymbol{x} + \boldsymbol{b}_i||_2 \leq \boldsymbol{c}_i^T \boldsymbol{x} + d_i,\ i = 1, \ldots, m \\ & & \boldsymbol{F}\boldsymbol{x} = \boldsymbol{g} \end{array}$$

ここで、$\boldsymbol{A}_i, \boldsymbol{b}_i, \boldsymbol{c}_i$ は適切なサイズの入力行列や入力ベクトルであり、 d_i は入力定数である。このような SOCP に整数制約を付加したものが混合整数二次錐計画問題 (mixed-integer second-order cone programming, MISOCP) である。

$$\begin{array}{rcl} \min & : & \boldsymbol{f}^T \boldsymbol{x} \\ \text{subject to} & : & ||\boldsymbol{A}_i \boldsymbol{x} + \boldsymbol{b}_i||_2 \leq \boldsymbol{c}_i^T \boldsymbol{x} + d_i,\ i = 1, \ldots, m \\ & & \boldsymbol{F}\boldsymbol{x} = \boldsymbol{g} \\ & & x_i \in \mathbb{Z}, (i = 1, \ldots, n) \end{array} \tag{1}$$

第 3 節において本研究で提案する数理モデルも、MISOCP (1) として定式化を行う。

MISOCP は決定変数に離散変数を含むため、厳密な最適解を求解するには組合せ最適化のアルゴリズムなどを用いる必要がある。代表的な手法としては、branch-and-bound (分枝限定法) や

branch-and-cut (分枝切除法) [3]、切除平面法 [8] などが挙げられる。分枝限定法などは商用ソルバーの CPLEX [1] にも実装されているが、変数の数が 50 程度以上になると長時間の計算が必要となり、実用的な時間で厳密解を求めることは容易ではない。

2.2 Gambella らの手法

Gambella ら [4] は CVTSP を MISOCP により定式化を行った。Garone ら [6] は事前に目的地を辿る順番が与えられている状況で vehicle の離着陸地点を決める問題である carrier-vehicle problem (CVP) のモデルを議論しているが、Gambella らは目的地を辿る順番を決定変数の扱いにすることで、目的地を辿る順番と vehicle の離着陸地点を同時に決定する問題として CVTSP を定式化している。このモデルは、depot から出発しすべての目的地を辿り再び depot に戻る時間の総和である線形な目的関数と、離着陸地点と目的地の距離と移動時間と速さの関係であるノルムを含んだ不等式、変数に対する下限と上限を定めた線形な制約式からなっているため MISOCP(1) の一種である。

Gambella ら [4] は、CVTSP に対して ranking-based solution algorithm という手法を提案した。CVTSP は目的地をたどる順番を決める離散最適化問題と、順番を決めたときに vehicle の離着陸座標を決める連続最適化問題に分けられる。この手法は、以下の 6 ステップで成り立つ。

1. 目的地をたどる順番を生成する。
2. その順番に対する下界を計算し、更新する。
3. 求めた下界が上界以上ならアルゴリズムを終了する。
4. step1 で求めた順番に対する最適な vehicle の離着陸地点を求め、これを上界とする。
5. 上界が今までの上界よりも小さければ更新する。
6. 反復回数を 1 増加させて、step1 に戻る

数値実験の結果では、既存のソルバーを用いた結果と比較して、精度が向上し実行時間も短い結果になったと報告されている。

しかしながら、Gambella ら [4] の CVTSP では、時間枠を取り込んでは無い。本論文では CVTSP に時間枠制約を付加することによって CVTSPTW の数理モデルを構築する。

2.3 焼きなまし法の説明

焼きなまし法 (Simulated Annealing, SA) とは、最適化問題に対し現実的な時間で近似解を求めるための確率的メタヒューリスティクスの一種であり、Kirkpatrick ら [10] によって考案された。この手法は巡回セールスマン問題 [7] や充足可能性問題 [19] といった探索空間が離散である問題に主に適用されている。目的関数値が良くなる場合にのみ解を更新する山登り法 (Hill Climging approach) と比較したとき、SA は目的関数値が悪くなる場合も確率的に解を更新することからより局所最適値に陥りにくい手法であるといえる。大域的最適解を求める手法である branch-and-cut

は解が良くなる可能性のある状態空間をすべて探索するのに対し、SA は現在の解の近傍のみを探索している。そのため SA は branch-and-cut と比較した場合に大域的最適解は求まらない可能性があるが、高速に近似解を求められる利点がある。SA の収束については Henderson らの研究 [9] などで議論されている。

SA は目的関数値が悪くなる場合に遷移する確率を算出するためのパラメータとして温度をもつ。温度が高いほど様々な解を探索する確率が高く、温度が低いほど目的関数値が向上する解にのみ遷移する確率が高まるように SA は設計され、通常は高い初期温度から徐々に低い最終温度へと下げていく。これにより、広い解を探索しながら最終的には収束するようなスケジュールとすることができる。Kirkpatrick ら [10] が示した最小化問題に対する遷移確率関数を示す。ここで s は現在の状態、$\bar{s}$ は次の状態、T はパラメータ温度であり、最小化する目的関数を f とする。

$$P(s, \bar{s}, T) = \begin{cases} 1 & \text{if } \ f(\bar{s}) < f(s) \\ \exp\left(\frac{f(s)-f(\bar{s})}{T}\right) & \text{if } \ f(\bar{s}) \geq f(s) \end{cases}$$

最小化問題に対する焼きなまし法の枠組みを Algorithm 1 に示す。このアルゴリズムは目的関数値との誤差を終了条件に含んでいないため最終結果を保証できないが、多くの実用的な最適化問題に対して現実的な時間で近似解を求めることができる。終了条件には、Algorithm 1 にあるように全体の反復回数が一定数に到達したときだけでなく、一定回数の反復で目的関数値が改善されないときなども考えられる。

2.4 Larasati らの手法

Larasati らの手法 [12] は CVTSP に対するヒューリステックとして ALG-SA を提案している。Algorithm 2 は ALG-SA の枠組みである。このアルゴリズムにおいて、x_0 は解の一つ、$f(x_0)$ は x_0 における目的関数値を表す。

Larasati ら [12] は数値実験の結果として、目的地の数が 3 から 9 の範囲では最適値とのギャップが 1% 未満、11 から 15 の範囲では 10% 未満と報告している。また、計算時間に関しては、目的地を 25 点に増加させても 0.57 秒ほどであった。一方、商用ソルバーのひとつである Gurobi では 13 点以上になると 1800 秒では求解ができなかった。よって Larasati らが提案した手法は、ターゲット数が増えるほど有用であるといえる。

Algorithm 1 焼きなまし法

1: イテレーション数 $k = 0$ と最大イテレーション数 $k_{\max}$ をとる
2: 初期状態 $state$ を一つとり、最良状態 $bestState$ とする。その時の目的関数値 $score$ を最良スコア $bestScore$ とする。
3: **while** $k < k_{\max}$ **do**
4: $k, k_{\max}$ をもとに現在の温度 temp を求める。
5: 新しい状態 $newState$ を得る。
6: 新しい状態の目的関数値 $newScore$ を得る。
7: $temp$ と $newScore, score$ をもとに遷移確率 $prob$ を求める。$newScore < score$ である場合は $prob = 1$ とする。
8: **if** $newScore < bestScore$ **then**
9: $bestScore \Leftarrow newScore$
10: $bestState \Leftarrow newState$
11: **end if**
12: $rand$ を [0,1] の乱数として得る。
13: **if** $rand < prob$ **then**
14: $state \Leftarrow newState$
15: $score \Leftarrow newScore$
16: **end if**
17: **end while**
18: $bestState$ を出力する

Algorithm 2 ALG-SA

INPUT: 目的地の座標
初期温度 T_{init}, 目標温度 T_f, 最大ステップ数 $S_{\max}$ とする
while $T_k > T_f$ **do**
  for 新規の解の数 $n_{solution}$ **do**
    新しい解 $x_0 + \Delta x$ を取る。
    if $f(x_0 + \Delta x) < f(x_0)$ **then**
      $f_{new} = f(x_0 + \Delta x)$
      $x_0 = x_0 + \Delta x$
    else
      $\Delta f = f(x_0 + \Delta x) - f(x_0)$
      r を 0 から 1 の乱数として取る。
      if $r > \exp(\Delta f/T)$ **then**
        $f_{new} = f(x_0 + \Delta x)$
        $x_0 = x_0 + \Delta x$
      else
        $f_{new} = f(x_0)$
      end if
    end if
  end for
  $f = f_{new}$
  $k = k + 1$
  温度 T_k を線形関数で更新する。
end while
最適な訪れる順序 x_0 を出力

表 1　問題例

名称	座標	時間枠
q_o	(0, 0)	None
q_f	(50,50)	None
q_1	(45,45)	[0,4]
q_2	(10,40)	[3,6]
q_3	(30,20)	[2,6]

3　問題設定

3.1　問題の定義

本研究が対象とする CVTSPTW (Carrier Vehicle Traveling Salesman Problem with Time Window) についての説明を与える。

CVTSPTW の目的は座標 q_o から出発し、すべての目的地を訪れた後 q_f に至るまでの時間の最小化である。移動に関してユークリッド平面上の任意の点に移動することができ、離着陸点も任意の座標を指定できる。目的地 $q_i (i = 1, \ldots, n)$ には時間制約 $[u_{i1}, u_{i2}]$ が設定されており、この時間内に必ず訪れなければならない。本問題に対して時間制約を $(0, \infty)$ とし vehicle の航続可能距離を ∞ と設定すると vehicle だけですべての目的地を訪れることになる。したがって、最終目的地までの移動時間の最小化は TSP と同じ設定となることから、CVTSPTW は TSP より少なくとも難しい問題であることが分かり、一般に NP 困難である。

目的地の数が 3 である場合の例を示す。座標と時間枠をそれぞれ表 1 で定義する。このときの最適ルートと離着陸点を図示すると図 2 となる。この図に置いて、青色の丸は開始地点、終了地点、目的地のいずれかを表す。赤色の丸は離陸地点、赤色の四角は着陸地点を表す。黒の実線は carrier と vehicle が同時に移動している箇所を表し、赤の点線は carrier のみ青の点線は vehicle のみが移動している箇所を表す。時間枠がない場合の訪れる順序は q_2, q_1, q_3, 時間枠がある場合は、q_3, q_2, q_1 となっている。これは、q_2 の受け入れ可能開始時刻が到着時間よりも遅いため、受け入れ開始時刻が早い q_3 を先にたどることによる。

3.2　MISOCP として定式化

ここでは、対象問題を解くためのモデルを MISOCP (1) を用いて定式化する。これは、Gambella らの先行研究 [4] の時間枠 $\boldsymbol{u}$ を加えた拡張と捉えることができる。

まずは定数を導入する。

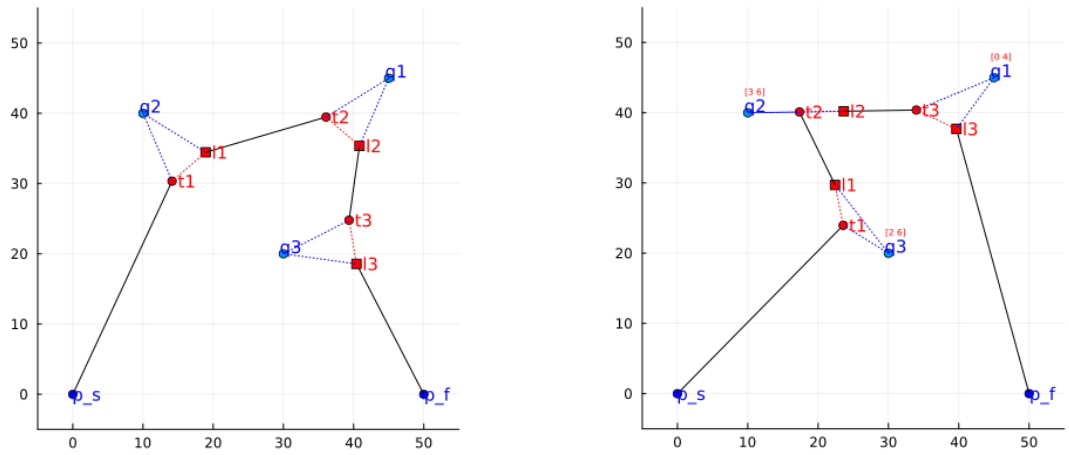


図 2　問題例 (表 1) の最適解と時間枠なしの場合の最適解の比較

定数

$n \in \mathbb{N}$　目的地の数

$\boldsymbol{q}_i \in \mathbb{R}^2,\ i = 1, \ldots, n$　目的地 i の座標

$\boldsymbol{q}_{\min} \in \mathbb{R}^2$　$\boldsymbol{q}_i$ の下限

$\boldsymbol{q}_{\max} \in \mathbb{R}^2$　$\boldsymbol{q}_i$ の上限

$V_v \in \mathbb{R}_+$　vehicle の速さ

$V_c \in \mathbb{R}_+$　carrier の速さ

$a \in \mathbb{R}_+$　vehicle が一回の輸送で移動できる距離の上限

$\boldsymbol{p}_s \in \mathbb{R}^2$　開始座標

$\boldsymbol{p}_f \in \mathbb{R}^2$　終了座標

$u_{i1} \in \mathbb{R},\ i = 1, \ldots, n$　目的地 i の受け入れ開始時刻

$u_{i2} \in \mathbb{R},\ i = 1, \ldots, n$　目的地 i の受け入れ開時刻

これらの定数のうち、下限と上限は q_i の座標に関する制限であり、具体的には各 $i = 1, \cdots, n$ に対して、$q_{\min,1} \leq q_{i,1} \leq q_{\max,1}$ および $q_{\min,2} \leq q_{i,2} \leq q_{\max,2}$ となっている。

次に決定変数を定義する。

変数

$\boldsymbol{Q}_i \in \mathbb{R}^2,\ i = 1, \ldots, n$　i 番目に訪れる目的地の座標

$\boldsymbol{w} \in \{0,1\}^{n \times n},\ i = 1, \ldots, n, j = 1, \ldots, n$　i 番目に目的地 j に訪れる場合を $w_{ij} = 1$、それ以外を $w_{ij} = 0$ とするバイナリ変数からなる行列

$\boldsymbol{p}_{to,i} \in \mathbb{R}^2,\ i = 1, \ldots, n$　$\boldsymbol{Q}_i$ に向けて離陸するときの座標

$\boldsymbol{p}_{l,i} \in \mathbb{R}^2,\ i = 1, \ldots, n$　$\boldsymbol{Q}_i$ を訪れた後に着陸するときの座標

$t_{i,1} \in \mathbb{R},\ i = 1, \ldots, n$　vehicle が $\boldsymbol{p}_{to,i}$ から $\boldsymbol{Q}_i$ に移動するのにかかる時間

$t_{i,2} \in \mathbb{R},\ i = 1, \ldots, n$　vehicle が $\boldsymbol{Q}_i$ から $\boldsymbol{p}_{l,i}$ に移動するのにかかる時間

$t_i \in \mathbb{R},\ i = 1, \ldots, n$　carrier が $\boldsymbol{p}_{to,i}$ から $\boldsymbol{p}_{l,i}$ に移動するのにかかる時間

$T_1 \in \mathbb{R}$　carrier が $\boldsymbol{p}_o$ から $\boldsymbol{p}_{to,1}$ に移動するのにかかる時間

$T_i \in \mathbb{R},\ i = 2, \ldots, n$　carrier が $\boldsymbol{p}_{l,i-1}$ から $\boldsymbol{p}_{to,i}$ に移動するのにかかる時間

$T_{n+1} \in \mathbb{R}$　carrier が $\boldsymbol{p}_{l,n}$ から $\boldsymbol{p}_f$ に移動するのにかかる時

$U_{i1} \in \mathbb{R}, i = 1, \ldots, n$　$\boldsymbol{Q}_i$ の受け入れ開始時刻

$U_{i2} \in \mathbb{R}, i = 1, \ldots, n$　$\boldsymbol{Q}_i$ の受け入れ終了時刻

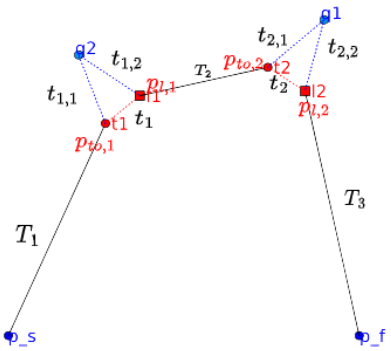


図 3　CVTSPTW の実行可能解の例

CVTSPTW に含まれる TSP の部分がどのように変数で表現されるかを示すために図 3 に実行可能解の例を示す。青丸で示した $\boldsymbol{q}_1, \boldsymbol{q}_2$ が目的地であるが、この実行可能解では $\boldsymbol{q}_2, \boldsymbol{q}_1$ の順番で訪れていることから、$\boldsymbol{Q}_1 = \boldsymbol{q}_2, \boldsymbol{Q}_2 = \boldsymbol{q}_1$ となる。よって $\boldsymbol{w} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$ である。時間制約に関して、$U_{11} < T_1 + t_{1,1} < U_{12}$, $U_{21} < T_1 + t_{1,1} + t_{1,2} + T_2 + t_{2,1} < U_{22}$ を満たせば十分であることもわかる。

このような定数と変数により CVTSPTW を以下のように MISOCP としてモデル化する。

$$\min : \sum_{i=1}^{n} t_i + \sum_{i=1}^{n+1} T_i \tag{2}$$

$$\text{subject to} : \|\boldsymbol{Q}_i - \boldsymbol{p}_{to,i}\| \leq V_v t_{i,1} \ (\forall i = 1, \ldots, n) \tag{3}$$

$$\|\boldsymbol{Q}_i - \boldsymbol{p}_{l,i}\| \leq V_v t_{i,2} \ (\forall i = 1, \ldots, n) \tag{4}$$

$$\|\boldsymbol{p}_{to,i} - \boldsymbol{p}_{l,i}\| \leq V_c t_i \ (\forall i = 1, \ldots, n) \tag{5}$$

$$\|\boldsymbol{p}_o - \boldsymbol{p}_{to,1}\| \leq V_c T_1 \tag{6}$$

$$\|\boldsymbol{p}_{l,i-1} - \boldsymbol{p}_{to,i}\| \leq V_c T_i \ (\forall i = 2, \ldots, n) \tag{7}$$

$$\|\boldsymbol{p}_f - \boldsymbol{p}_{l,n}\| \leq V_c T_{n+1} \tag{8}$$

$$t_{i,1} + t_{i,2} \leq t_i \ (\forall i = 1, \ldots, n) \tag{9}$$

$$\boldsymbol{Q}_i = \sum_{j=1}^{n} \boldsymbol{w}_{i,j} \boldsymbol{q}_j \ (\forall i = 1, \ldots, n) \tag{10}$$

$$\sum_{j=1}^{n} \boldsymbol{w}_{i,j} = 1 \ (\forall i = 1, \ldots, n) \tag{11}$$

$$\sum_{i=1}^{n} \boldsymbol{w}_{i,j} = 1 \ (\forall j = 1, \ldots, n) \tag{12}$$

$$t_{i,1} \geq 0 \ (\forall i = 1, \ldots, n) \tag{13}$$

$$t_{i,2} \geq 0 \ (\forall i = 1, \ldots, n) \tag{14}$$

$$0 \leq t_1 \leq a \ (\forall i = 1, \ldots, n) \tag{15}$$

$$T_i \geq 0 \ (\forall i = 1, \ldots, n) \tag{16}$$

$$\boldsymbol{q}_{\min} \leq \boldsymbol{Q}_i \leq \boldsymbol{q}_{\max} \ (\forall i = 1, \ldots, n) \tag{17}$$

$$\boldsymbol{w}_{ij} \in \{0, 1\} \ (\forall i = 1, \ldots, n, \forall j = 1, \ldots, n) \tag{18}$$

$$U_{i,1} \leq T_1 + t_{1,1} + \sum_{j=1}^{i-1} t_{j,2} + T_{i+1} + t_{i+1,1} \leq U_{i,2} \tag{19}$$

$$U_{i,1} = \sum_{j=1}^{n} \boldsymbol{w}_{i,j} u_{j,1} \ (\forall i = 1, \ldots, n) \tag{20}$$

$$U_{i,2} = \sum_{j=1}^{n} \boldsymbol{w}_{i,j} u_{j,2} \ (\forall i = 1, \ldots, n) \tag{21}$$

目的関数 (2) は総行程にかかる時間であり、具体的には carrier が q_o を出発し、vehicle がすべての目的地を訪問した後に carrier が q_f に至るまでの時間である。制約式 (3)(4) は vehicle が $p_{to,i}$ から離陸して Q_i に至る時間、Q_i から $p_{l,i}$ に着陸する時間をそれぞれ表している。(5) は carrier が $p_{to,i}$ から $p_{l,i}$ に至る時間を表している。このとき vehicle と carrier は分離している点に注意する。制約式 (6)-(8) は carrier と vehicle が同時に移動しているときの時間を表している。(9) は離陸した vehicle が carrier に着陸できることを制約している。(10) は i 番目に訪れる座標が q_j であることを保証するための制約である。(11),(12) はそれぞれ i 番目に訪れる目的地は 1 つであること、各目的地には一度訪れることを保証している。(13)-(17) はぞれぞれ vehicle, carrier の移動時

間が負とならないことを保証している。特に (15) は vehicle が移動できる時間の上限を a で制限している。(19) は各時間制約に違反しないことを保証するための制約である。具体的には、各目的地 Q_i に訪れるまでの vehicle の移動時間の総和 $T_{sum,i}$ が $T_{sum,i} \in [U_{i,1}, U_{i,2}]$ を満たすことを保証している。(20),(21) は Q_i に対応する時間窓を割り当てるための制約である。

制約式 (3)-(8) は二次錐制約であり (9)-(21) は線形制約であること、決定変数 w_{ij} は整数制約を持つため、このモデルは MISOCP である。

4　提案手法

本節では提案手法について述べる。Larasati ら (2022)[12] を参考にし、CVTSPWT に対して使えるよう拡張した。時間制約をつけるために、目的地を訪れる順番が決まった後の離着陸地点を決める問題に対して式 (2) で定義した最適化問題を用いる。

提案するアルゴリズムは 1. 訪れる順番を決める離散最適化問題、2. 最適な離着陸点を決める連続最適化問題の 2 つの最適化問題を解くことで近似解をもとめるものである。もともとの問題は MISOCP であり、CPLEX や GUROBI などの MISOCP を扱える既存のソルバーを用いれば厳密解の求解が可能であるが、長時間の計算時間が必要である。一方、たどる順番が与えられて最適な離着陸地点を求める問題は SOCP となり高速にソルバーを用いて最適解が求まる。

最適な訪れる順番を決める問題は、離散計画問題であり一般に汎用ソルバーを用いての計算には非常に時間がかかる。そこで本論文ではメタヒューリスティックの一種である焼きなまし法を用いて解の探索を行う。焼きなまし法では、第 2.3 節で述べたように、局所最適解をなるべく避けるために、解の評価値が悪化する場合も確率的に遷移する特徴を持つため、すべての状態を列挙する方法や、解が改善された場合のみ遷移を行う方法と比べて、効率的に良い解が得られる。

提案手法の Algolithm 3 の詳細を示す。このアルゴリズムは 2.3 章で示した焼きなまし法をベースとした手法である。

まずは提案手法の中で用いている関数を提示する。遷移確率を計算する関数 $getProb()$、温度を計算する関数 $getTemp()$、初期状態を得る関数 $getFirstState()$、状態に対する目的関数値を得る関数 $getScore()$、次の状態を得る関数 $getNextState()$ をそれぞれ定義する。

状態 $state$ は目的地を訪れる順番を表す行列であり、第 3.2 節の定式化における $\boldsymbol{w}$ と同様の役割である。

まず、$state$ を受容する確率を計算する関数を式 (22) で定義する。遷移確率はスコアが改善していれば 1, そうでなければ 0 以上 1 以下の正実数を返す必要がある。スコアが改善しているほど 1 に近いことと温度が高いほど遷移しやすいことを表現するために 2.3 章で紹介した以下の定義を用いる。

$$getProb(temp, newScore, score) = \begin{cases} 1 & \text{if } newScore < score \\ \exp(\frac{score - newScore}{temp}) & \text{if } newScore \geq score \end{cases} \tag{22}$$

次に、現在の試行回数から、温度パラメータを計算する関数を式 (23) とする。温度は初期の大きい値 $firstTemp$ から最終的な小さい値 $lastTemp$ へと徐々に下がっていくように設計する。今回の関数は k に対する単調減少な線形関数であり、線形スケジュールと呼ばれる。

$$getTemp(k, k_{\max}) = firstTemp + \frac{k}{k_{\max}}(lastTemp - firstTemp) \tag{23}$$

式 (24) で初期状態を求める関数を定義する。初期状態は、アルゴリズムを実行する時の最初の解であり、高速で求められることが望ましい。初期状態は一般的にランダムに求められることが多

Algorithm 3 提案手法

INPUT: 目的地と開始終了地点の座標、目的地の時間枠
定数の設定: 初期温度 T_{first}, 目標温度 T_{last}, イテレーション回数 $k_{\max}$
$k \Leftarrow 0$
$state \Leftarrow getFirstState()$
$score \Leftarrow getScore(state)$
$bestState \Leftarrow state, bestScore \Leftarrow score$
while bestState が実行可能解でない、または、前回のループで bestScore が更新された **do**
  while $k < k_{\max}$ **do**
    $temp \Leftarrow getTemp(k, k_{\max})$
    $newState \Leftarrow getNextState(state)$
    $newScore \Leftarrow getScore(newState)$
    $prob \Leftarrow getProb(temp, newScore, score)$
    if $newScore < bestScore$ **then**
      $bestScore \Leftarrow newScore$
      $bestState \Leftarrow newState$
    end if
    $rand \Leftarrow random(0, 1)$
    if $rand < prob$ **then**
      $state \Leftarrow newState$
      $score \Leftarrow newScore$
    end if
  end while
end while
OUTPUT:$bestState, bestScore$ がそれぞれ最適解と最適値

く、そのような場合でも適切に実行すれば焼きなまし法は収束する。本論文でもたどる順番がランダムとなるような解を返す関数として構成している。

$$getFirstState() = \text{任意の置換行列} \tag{24}$$

状態のスコアを求める関数を式 (25) で定義する。本論文では、スコアを第 3.2 章で定義した最適化問題の状態 $\boldsymbol{w}$ を決定変数でなく入力とした最適化問題を構成し、その最適値として定義する。この変換により、整数変数がなくなるため、モデルのクラスが MISOCP から SOCP となり、高速な求解が可能となる。ここで、スコアとは状態と理想とする状態との近さを表現できていればよく、必ずしも最適化問題の最適値でなくても良いことに注意する。

それぞれの目的地に対して訪れることが可能な時間枠の制約があるため、入力された順番によっては実行可能な離着陸地点が存在しない場合がある。その場合は通常の最適化問題であれば解なし

となり最適値が求まらないが、焼きなまし法では繰り返しの処理の中で解の評価を行う必要があるため、今回は一定の大きい値を代わりに返すことで解の評価値の代わりとしている。

$$getScore(state) = \begin{cases} SOCP \text{ の最適値} & \text{if state に対して実行可能な解が存在する} \\ M & \text{if state に対して実行可能な解が存在しない} \end{cases} \tag{25}$$

ただし、M は十分大きい実数である。

状態を入力として新たな状態を求める関数 getNextState を定義する。焼きなまし法では、もとの状態を少し変化させたものを新しい状態として得ることが多く、この新たな状態を近傍と呼ぶ。本論文では近傍を 2-swap により得る。2-swap は、現在の訪れる順番のうち、任意の二箇所を選び、それらを入れ替える操作として定義する。通常の TSP では訪れる経路の中で交差している箇所の交差を解消する操作である 2-opt も効果的である。

図 4 は提案手法を用いて、目的地数が 9 の問題インスタンスを作成し、実際に解いたときの画像である。

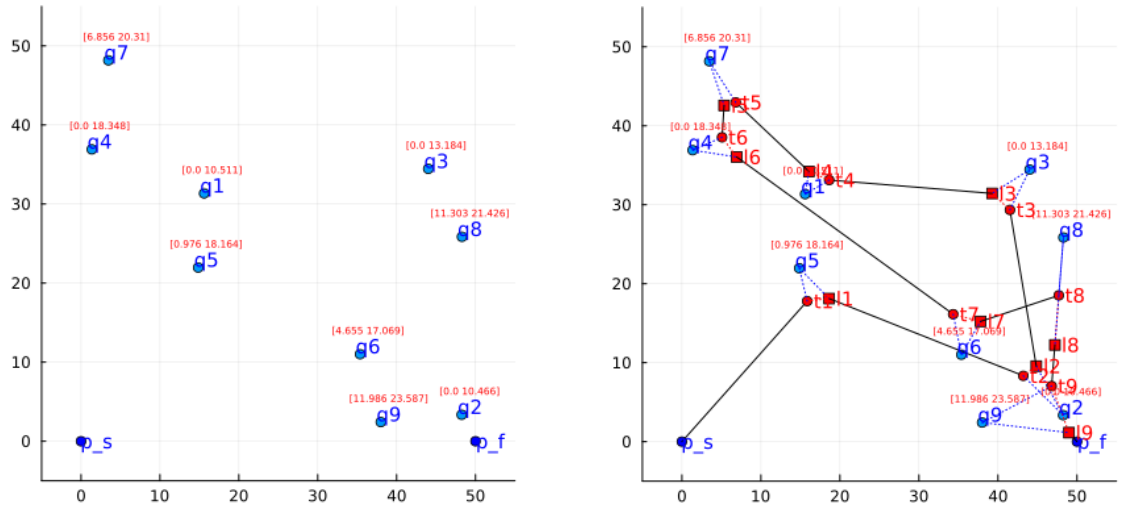


図 4　提案手法を用いて CVTSPTW を解いた例

表 2　定数の設定

定数	値
$\boldsymbol{q}_i (i = 1, \ldots, n)$	第 5.2 節にて生成する
$\boldsymbol{q}_{\min}$	(0,0)
$\boldsymbol{q}_{\max}$	(60,60)
V_v	60
V_c	18
a	21/60
$\boldsymbol{p}_o$	(0,0)
$\boldsymbol{p}_f$	(50,0)
$u_{i,1}$	第 5.2 節にて生成する
$u_{i,2}$	第 5.2 節にて生成する

5　数値実験

この節では、CVTSPTW のデータセットを作成し、実際に提案手法を用いて解くことでその性能を評価する。本論文では、SOCP を解くソルバーとして、IBM 社の CPLEX を用いた。

実験を行った実行環境は以下である。

- OS: macOS Monterey 12.6.2
- プロセッサ: Intel Dual Core i5 1.6 GHz
- RAM: 8.0GB
- プログラミング言語: Julia

5.1　パラメータの設定

第 3.2 節で定義した定数の具体的な値として数値実験では表 2 の値を用いた。基本的には $\boldsymbol{q}_{\min}, \boldsymbol{q}_{\max}$ の値から分かるように $[0, 60] \times [0, 60]$ の 2 次元範囲に目的地が存在することを想定して、値を設定している。

提案手法における各種パラメータについても表 3 として定める。実運用するのであればデータの性質ごとに調整する必要があるが、本論文では簡単のためにすべて固定する。なお、第 5.3.3 節の実験に関しては、反復回数の影響を考察するために反復回数を実験ごとに変更しているので注意する。

表 3　提案手法のパラメータの設定

パラメータ	値
$\mathrm{T_{first}}$	1
$\mathrm{T_{last}}$	0.001
$\mathrm{k_{max}}$	50
実行不可能時の最適化問題の値 (M)	500

Algorithm 4 テスト問題における目的地の座標の生成手法

1: 目的地の数 n を設定する。
2: $\boldsymbol{q}_i, i = 1, \cdots, n$ の座標を $[\boldsymbol{q}_{\min,1}, \boldsymbol{q}_{\max,1}], [\boldsymbol{q}_{\min,2}, \boldsymbol{q}_{\max,2}]$ の一様分布に従って決定する。
3: $\boldsymbol{p}_o$ から $\boldsymbol{q}_i$,$i = 1, \ldots, k-1$ をすべてたどり、$\boldsymbol{q}_k$ にたどり着く時間 t_k をそれぞれ求める。時間は距離の総和を carrier の速さで割り算出する。
4: 時間枠のサイズ $wid_i, i = 1, \cdots, n$ を $[n, n+10]$ の一様分布として求める。
5: $\boldsymbol{q}_i, i = 1, \cdots, n$ に対する時間枠 $\boldsymbol{u}_i$ を、$[t_k - wid_i/2, t_k + wid_i/2]$(ただし、$t_k - wid_i/2 < 0$ となる場合は、$[0, wid_i]$) と設定する。

5.2　問題の生成

数値実験用のテストデータにおける目的地の座標は Algorithm 4 を用いて生成した。基本的には、目的地を 2 次元空間上にランダムに生成し、実行可能解を少なくとも一つは持つように時間枠を設定している。

5.3　実験結果

今回の実験では、目的地数を $n = 7, 9, 11, 13, 15, 17, 30, 50, 70$ と変化させて、Algorithm 4 を適用して問題を生成した。問題の規模に応じて $7 \leq n \leq 17$ の問題を small instance、$n = 30, 50, 70$ のデータを large instance と分類した。前者は主に既存ソルバーとの性能の比較に用い、後者は提案手法の限界性能を測る目的で用いる。乱数のシードを変更することで、small instance と $n = 30$ はそれぞれ 7 つ、large instance はそれぞれ 3 つのデータセットを生成した。CPLEX を用いた手法での計測は small instance に対してそれぞれ一回おこなった。提案手法は悪化する解の採択をコントロールする部分で確率を用いるため、数値実験も乱数のシードを変更して計測を行い、具体的には small instance に対してそれぞれ 10 回、large instance に対してはそれぞれ 5 回行った。また、実験時間が 600 秒を超えた場合、実行可能解が存在する場合はその時点での最良解を、そうでない場合は TIME OUT を返すものとした。なお、これ以降の表における計算時間の単位は秒である。

表 4　CPLEX と提案手法の実験結果

目的地数 n	CPLEX 平均計算時間	CPLEX 標準偏差計算時間	提案手法 平均計算時間	提案手法 標準偏差計算時間	提案手法 平均 OptGap	提案手法 標準偏差 OptGap
7	4.686	2.409	5.404	0.794	3.646(%)	0.037
9	10.453	8.695	7.145	1.840	5.802(%)	0.0776
11	42.565	47.487	21.464	4.893	2.186(%)	0.020
13	287.809	219.355	10.144	1.05	6.027(%) 5.987(%)(*)	0.055 –(*)
15	516.408	221.280	11.949	0.896	1.522(%) 8.091(%)(*)	– 0.086(*)
17	600.029	0.031	15.192	0.916	10.997(%)(*)	0.151

5.3.1　計算時間と精度の評価

CPLEX との提案手法の実験結果の比較をまとめたのが表 4 である。ここで、最初の列は目的地の数を示しており、第 2 列および第 3 列は CPLEX で得られた計算時間の平均と標準偏差を表している。同様に第 4 列および第 5 列は提案手法で得られた計算時間の平均と標準偏差である。また、第 6,7 列は CPLEX の最適値の平均 ($\mathrm{CPLEX_{ave}}$) と提案手法の出力値の平均 $\mathrm{SA_{ave}}$ の差の比率 $\mathrm{OptGap} = \frac{\mathrm{SA_{ave}} - \mathrm{CPLEX_{ave}}}{\mathrm{CPLEX_{ave}}}$ である。また、表の中で (*) がついている値は CPLEX が時間制限内で得られた最良値を用いて計算した。$n = 13$ の平均 OptGap のように二段となっている箇所の上段は CPLEX が制限時間内に終了した場合の最適値を用いており、下段は CPLEX が制限時間内に終了しなかった場合のその時点での最良値を用いている。また、データの数が一つしかなく、標準偏差を求められないものは–とした。

表 4 の結果より、CPLEX は n が 11 以下の場合は 600 秒という制限時間内に最適解を得ることができた。一方、$n = 13, 15, 17$ のいくつかのケースでは時間内に最適解を得られなかった。提案手法では、n が 17 以下のケースでは、平均計算時間が最大で 21 秒弱であり、時間内に近似解を求めることができている。

図 5 は提案手法と CPLEX との OptGap を表したものである。青色の線は最適値と提案手法の出力値との差を、赤色の線は制限時間内の CPLEX の最良値と提案手法の出力値との差を表している。$n = 17$ 以下の範囲では CPLEX ソルバーと提案手法とのギャップはおおよそ 11% ほどであり、n が増えるほどギャップが大きくなっていることが確認できる。

図 6 は提案手法と CPLEX の実行時間を表したものである。CPLEX は n が大きくなるほど実行時間が急速に大きくなっているが、一方で提案手法では実行時間にそれほど変化がないことがわかる。

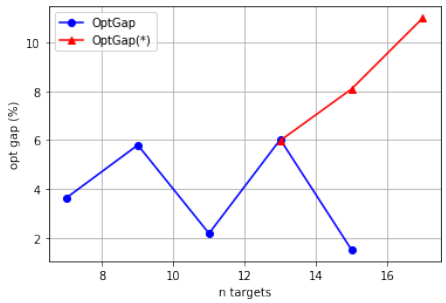


図 5　最適解と提案手法の目的関数値のギャップ

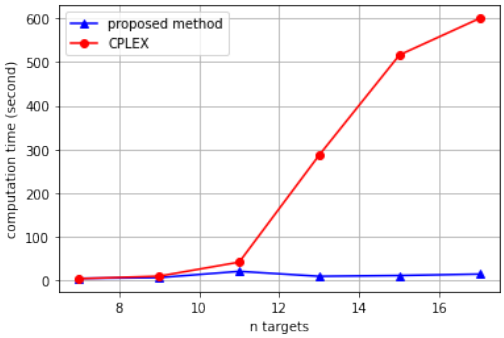


図 6　目的地数ごとの CPLEX と提案手法の計算時間の比較

5.3.2　実行不可能な場合の考察

数値実験において、提案手法が制限時間内に実行可能解を得られなかった場合について考察する。なお、今回の実験で用いるデータはデータ生成の方法から必ず一つ以上の実行可能解を持つことが保証されている。しかしながら、$n = 50, 70$ のそれぞれ 3 つのデータセットを 5 回実験した範囲では、すべて実行可能解が見つからなかった。

このような大きいデータセットにおいて実行可能解が見つからなかった理由として 2 つ挙げられる。1 つ目に小さいデータセットと比較して時間制約の数が多く、実行可能解の濃度が小さい点がある。2 つ目に、提案手法は近傍が実行不可能である場合は無条件に受け入れている。つまり、解の状態を考慮せずに探索しているため、反復回数や温度にかかわらずランダムな状態に遷移してしまう構造がある。

上記の問題を克服する方法を考察する。1 つ目の理由に関しては、問題の性質であるため改善が難しい。本研究では、初期点をランダムに生成するというような簡単な設定方法をしているが、これに対して VRPTW などで提案されているような手法で初期点を生成する方法なども検討の余地がある。2 つ目の理由に関して、実行不可能の場合には目的関数値が存在しないため、反復回数に応じて実行可能な状態に近づくスコア関数を設計する。具体的には、時間制約を逸脱している目的地の数の総和などが考えられる。

5.3.3　実行時間と目的関数値の関係

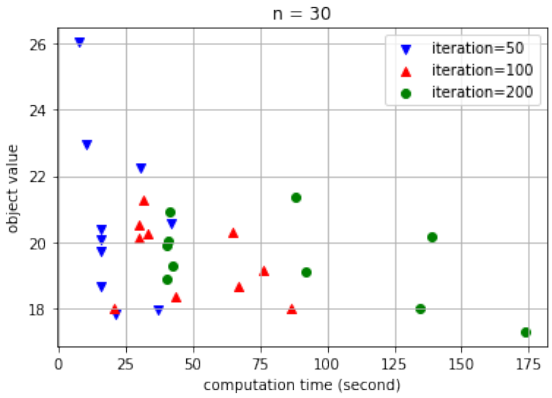
ここでは、提案手法のパラメータである反復回数を変化させた場合に、実行時間と目的関数値がどう変化するのかを検証する。本小節での反復回数は、アルゴリズム 3 における、$k_{\max}$ であることに注意する。実験内容は、$n = 30$ のデータセットを 4 つ用い、各データセットに対して反復回数を 50,100,200 と変化させ提案手法により得た結果が図 7 である。このデータの計算時間と目的関数値の平均と標準偏差をまとめたものが表 5 である。この実験においては、すべてのインスタンスで制限時間内に実行可能解を求められている。

図 7 の各グラフは表 5 の乱数のシードに対応しており、グラフの横軸は計算時間であり縦軸は出

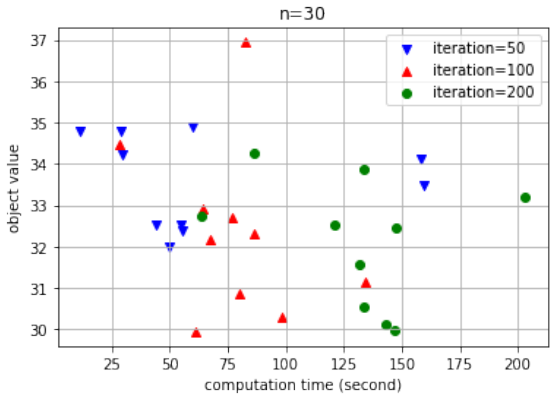
表 5　提案手法における反復回数を変更した場合の計算時間、目的関数値の変化

乱数のシード	設定反復回数	平均計算時間	標準偏差計算時間	平均目的関数値	標準偏差目的関数値
1055	50	21.306	11.547	20.644	2.520
	100	48.471	23.076	19.469	1.171
	200	83.379	50.209	19.504	1.247
1056	50	65.160	51.528	33.573	1.127
	100	77.909	27.302	32.379	2.101
	200	131.091	37.290	32.129	1.522
1057	50	39.355	22.235	32.124	1.202
	100	42.478	29.694	34.033	0.929
	200	62.091	20.348	33.745	0.866
1058	50	74.875	78.147	36.138	1.231
	100	46.787	28.616	35.671	0.835
	200	132.493	85.886	35.601	0.749

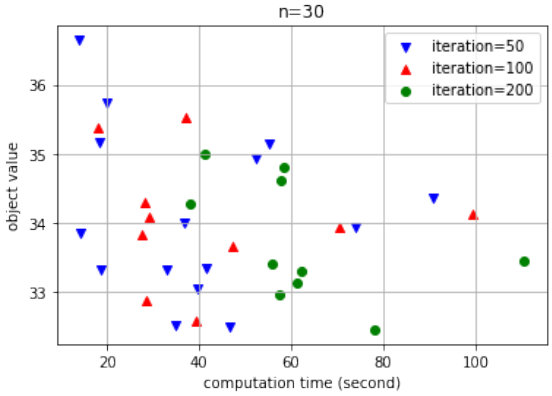
力された目的関数値である。表 5 と図 7 の結果から、反復回数を大きくするほど実行時間の平均も大きくなる傾向を見て取れる。提案手法のアルゴリズム 3 の継続条件の一つが前回までの反復の最良目的関数値より優れた目的関数値を得ることであるため、反復回数が大きいほど一回の反復で多くの近傍を探索するひつようがあり、継続する確率が上がるためと考えられる。また同様の理由から、反復回数が大きいほど近傍を探索できる確率が大きいため、目的関数値が小さくなる傾向が説明できる。



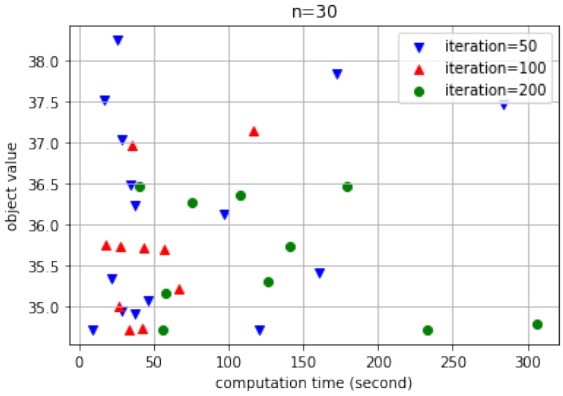
シードが 1055 の場合



シードが 1056 の場合



シードが 1057 の場合



シードが 1058 の場合

図 7　提案手法における実行時間と精度の関係

6　まとめと今後の課題

本研究では、時間制約付き carrier-vehicle 問題 (CVTSPTW) を定義し、問題の一部を SOCP として定式化し、焼きなまし法と組み合わせるヒューリスティクス解法を提案した。CVTSP に対して焼きなまし法を適用した Larasati らの研究 [12] と同様の結果が CVTSPTW に拡張した場合でも得られている。第 5 節の数値実験の結果、既存ソルバーである CPLEX を用いた方法と比較して短い時間で近似解が得られることがわかった。一方で、点の数が多い問題に対しては実行可能解を制限時間内に見つけることが困難であった。

本研究に対する今後の課題としては、点の数が多い問題に対しても制限時間内に実行可能解を見つけることが挙げられる。そのための方法として以下の方法を今後の研究で行いたい。

- 初期解をランダムではなく距離と時間枠を考慮した方法で作成する。
- 目的地を訪れる順番が実行不可能である時は目的関数値が得られないため一定の値を返しているが、時間枠を満たさない個数をペナルティとする目的関数値を構築する。
- 解空間が複雑な形をしているため、2-swap 以外の近傍関数を構成する。
- アルゴリズムのパラメータである初期温度、最終温度を問題の性質に応じて調整する。

謝辞

本論文を執筆するにあたり、たくさんの助言や添削をしてくださり、どの質問にもとても丁寧に回答をしてくださった指導教員の山下真教授に感謝いたします。また、ゼミを通して多くのことを教えてくださった山下研究室、澄田研究室の先生方や先輩方に感謝いたします。最後に、学士論文をともに取り組んだ同期の皆様や、支えてくださった家族に感謝いたします。

参考文献

[1] Christian Bliek1ú, Pierre Bonami, and Andrea Lodi. Solving mixed-integer quadratic programming problems with ibm-cplex: a progress report. In *Proceedings of the Twenty-Sixth RAMP symposium*, pp. 16–17, 2014.

[2] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[3] Sarah Drewes and Stefan Ulbrich. *Mixed integer second order cone programming*. Verlag Dr. Hut Germany, 2009.

[4] Claudio Gambella, Andrea Lodi, and Daniele Vigo. Exact solutions for the carrier–vehicle traveling salesman problem. *Transportation Science*, Vol. 52, No. 2, pp. 320–330, 2018.

[5] Claudio Gambella, Joe Naoum-Sawaya, and Bissan Ghaddar. The vehicle routing problem with floating targets: Formulation and solution approaches. *INFORMS Journal on Computing*, Vol. 30, No. 3, pp. 554–569, 2018.

[6] Emanuele Garone, Roberto Naldi, Alessandro Casavola, and Emilio Frazzoli. Planning algorithms for a class of heterogeneous multi-vehicle systems. *IFAC Proceedings*, Vol. 43, No. 14, pp. 969–974, 2010.

[7] Xiutang Geng, Zhihua Chen, Wei Yang, Deqian Shi, and Kai Zhao. Solving the traveling salesman problem based on an adaptive simulated annealing algorithm with greedy search. *Applied Soft Computing*, Vol. 11, No. 4, pp. 3680–3689, 2011.

[8] Amin Gholami and Xu Andy Sun. Towards resilient operation of multimicrogrids: An misocp-based frequency-constrained approach. *IEEE Transactions on Control of Network Systems*, Vol. 6, No. 3, pp. 925–936, 2018.

[9] Darrall Henderson, Sheldon H Jacobson, and Alan W Johnson. The theory and practice of simulated annealing. *Handbook of Metaheuristics*, pp. 287–319, 2003.

[10] Scott Kirkpatrick, C Daniel Gelatt Jr, and Mario P Vecchi. Optimization by simulated annealing. *Science*, Vol. 220, No. 4598, pp. 671–680, 1983.

[11] Martin Klau***v***co, Slavom´***i***r Bla***v***zek, Michal Kvasnica, and Miroslav Fikar. Mixed-integer socp formulation of the path planning problem for heterogeneous multi-vehicle systems. In *2014 European Control Conference (ECC)*, pp. 1474–1479. IEEE, 2014.

[12] Maharani Rizki Larasati and I-Lin Wang. An integrated integer programming model with a simulated annealing heuristic for the carrier vehicle traveling salesman problem. *Procedia Computer Science*, Vol. 197, pp. 301–308, 2022.

[13] Min Lin, Yuming Chen, Rui Han, and Yao Chen. Discrete optimization on truck-drone collaborative transportation system for delivering medical resources. *Discrete Dynamics in Nature and Society*, Vol. 2022, pp. 1–13, 2022.

[14] Miguel Sousa Lobo, Lieven Vandenberghe, Stephen Boyd, and Herve Lebret. Second-order cone programming. *Linear Algebra and Applications*, Vol. 284, pp. 193–228, 1998.

[15] Zhihao Luo, Zhong Liu, and Jianmai Shi. A two-echelon cooperated routing problem for a ground vehicle and its carried unmanned aerial vehicle. *Sensors*, Vol. 17, No. 5, p. 1144, 2017.

[16] Hyung Jin Park, Reza Mirjalili, Murray J Côté, and Gino J Lim. Scheduling diagnostic testing kit deliveries with the mothership and drone routing problem. *Journal of Intelligent & Robotic Systems*, Vol. 105, No. 2, p. 38, 2022.

[17] Stefan Poikonen and Bruce Golden. The mothership and drone routing problem. *INFORMS Journal on Computing*, Vol. 32, No. 2, pp. 249–262, 2020.

[18] Florian A Potra and Stephen J Wright. Interior-point methods. *Journal of Computational and Applied Mathematics*, Vol. 124, No. 1-2, pp. 281–302, 2000.

[19] William M Spears. Simulated annealing for hard satisfiability problems. *Cliques, Coloring, and Satisfiability*, Vol. 26, pp. 533–558, 1993.

[20] Felix Tamke and Udo Buscher. A branch-and-cut algorithm for the vehicle routing problem with drones. *Transportation Research Part B: Methodological*, Vol. 144, pp. 174–203, 2021.