# 時間枠付き巡回路問題に対する実験的解析

修士学位論文

# 時間枠付き巡回路問題に対する実験的解析

2021 年度
(2022 年 3 月)

東京海洋大学大学院
海洋科学技術研究科
海運ロジステイクス専攻

李雅婕

# 目 次

# 第1章　はじめに

## 1.1　研究背景

物流効率の向上、物流コストの削減、低炭素物流の実現は、現実社会においても非常に重要なことである。そのため、配達最短経路を求める組合せ最適化問題の研究が必要になる。巡回セールスマン問題（Traveling Salesman Problem、TSP）は、各都市間の距離が与えられたとき、出発地（デポ）から、複数の都市をちょうど 1 回ずつ訪問して出発地の都市に戻る最短経路を求める組合せ最適化問題である。巡回セールスマン問題を解決することは、現実に大きな影響を及ぼす。巡回セールスマン問題は多項式時間では最適解を得ることができないと予想されている問題であり、ある程度の精度を持った近似解を短時間に求めることが必要かつ重要とと述べている [4]。

### 1.1.1　時間枠付き巡回セールスマン問題とは

現実生活では様々な応用が考えられるため、巡回セールスマン問題について多くの研究が行われており、巡回セールスマン問題を拡張した様々な問題が研究している。時間枠付き巡回セールスマン問題 (TSP-TW) は，TSP の各都市に対する車両の出発時刻の制約を付けて拡張した問題である。TSP と同様な、短時間で最適解を見つけるさわめて難解である。

## 1.2　研究目的

ベンチマークである時間枠付き巡回セールスマン問題について、実験比較するために 3 つの定式化（Miller-Tucker-Zemlin(ポテンシャル) 定式化、ポテンシャル定式化基にして、持ち上げの操作定式化、2 添字ポテンシャル定式化）を述べる。3 つの定式化による時間枠付き巡回セールスマン問題を Gurobi で詳細な実験的解析を行った。Google OR-tools による時間枠付き巡回セールスマン問題を解決して、TSP に対する Google OR-tools と Gurobi Optimizer との比較をを実験的に行なった結果を示す。その実験で得られた結果と Gurobi ソルバーで得られた結果を比較し、実行時間と最適解の精度を分析して、2 つの解法のパフォーマンスを評価する。

## 1.3　論文構成

本論文の構成は以下の通りである。

- 第 2 章では、本研究で関連研究について述べる。
- 第 3 章では、本研究で用いている定式化について述べる。
- 第 4 章では、定式化評価実験について述べる。
- 第 5 章では、OR-tools による解法について述べる。

- 第 5 章では、Gurobi と OR-tools の比較実験について述べる。
- 第 6 章では、まとめ及び今後の課題について述べる。

# 第2章　関連研究

TSP の解法として、遺伝的アルゴリズム (GA)、シミュレーテッドアニーリング、アントコロニー最適化 (ACO)、粒子群最適化 (PSO)、タブーサーチ (TS)、ニューラルネットワーク (NN) 等のヒューリスティックアプローチが行われており、全体最適または妥当な準最適解を求める。GA は進化的アルゴリズムの中でも、離散的な組合せ最適化問題を解くのに有効なアルゴリズムとして注目されている [5]。

## 2.1　遺伝的アルゴリズム (GA) を用いた TSP 問題に関する研究

遺伝的アルゴリズム（GA）は、70 年代に提案されて以来、組合せ最適化問題を解くメタヒューリスティックとして最も成功した手法の一つとなっている。遺伝的アルゴリズム（GA）は、自然界における生物の進化の法則に基づき、アルゴリズムを設計・提案している。 ダーウィンの生物進化論の自然選択と遺伝のメカニズムをシミュレーションした生物進化過程の計算モデルであり、自然進化過程をシミュレーションして最適解を探索する手法である。

遺伝的アルゴリズムの性能は、初期解の生成、交差方法、突然変異方法によって決定される。巡回セールスマン問題 (TSP) における、筆者は 6 種類以上のクロスオーバー演算子を TSP に適用し、実験的に比較した結果を示す。その結果、OX 演算子が他の演算子よりも優れた解を得ることができることが示された [6]。

交差方法サイクル (Cycle,CX) 関数の使用は、GA に最適化の能力を与える他の関数（突然変異と生存者の選択）であるため、GA の最適化プロセスにいかなる改善も与えない。この種の関数を使用すると、実行時間が大幅に増加する。さらに、ノード数が増加するにつれて、実行時間は指数関数的に増加する [7]。

メタヒューリスティック関数の最適化能力を検証するためには、中立的な関数を使用しなければならず。CX 関数の主な目的は、母集団の個体の特徴を組み合わせて新しい個体を得ることである。したがって、中立的な交配から得られる子孫が親を改良することは非常にありえないので、CX はその目標を達成することはできないと説明している。このような理由から、CX 関数の主な効用は、アルゴリズムの探索過程の能力を向上させることにあると述べている [7]。

## 2.2　深層学習を用いた巡回セールスマン問題に関する研究

深層学習 (Deep Learning) とは、多層構造のニューラルネットワークを用いる、データ特徴の非線形変換や表現をディープニューラルネットワーク（DNN）を用いた学習し、様々な解析活動を行う手法である。近年、機械学習や深層学習を用いた技術が盛んに研究されており、これまで困難であった問題の解決可能性があるとして注目されている。

畳み込みニューラルネットワーク (CNN: Convolutional Neural Network) は畳み込み演算を行う処理層で構成されるニューラルネットワークであり、画像認識をはじめ画像を入力とする問題において高い性能を示している [8]。

巡回セールスマン問題における距離に代わる指標として、畳み込みニューラルネットワークにより最適経路画像を近似した優良エッジ値 (Good-Edge Distribution) を計算された。筆者は TSP の近傍探索法の 1 つである 2-opt 法において距離の代わりに優良エッジ値を用いる EV-2opt 法 (Edge Value 2-opt) を提案し，テスト用問題例について各解法により得られた解の平均誤差率を比較する。EV-2opt 法を利用した提案手法 (EV- 2opt+2opt) では，まず貪欲法ベースのアルゴリズムによって初期解を生成し，これに対して EV-2opt 法を適用した後，さらに通常の距離を用いた 2-opt 法を適用する。EV-2opt+2opt 法および 2opt 法を用いて，テスト用問題例に対して解を求める操作をそれぞれ 20 回試行する、これにより得られた解の平均誤差率を示す。この結果について，EV-2opt+2opt 法用いることでより最適解に近い解を求め，より質の良い解から 2-opt 法による探索を始めることができたため，誤差率が改善したと評価している [8]。

ビームサーチ (Beam Search) は探索アルゴリズムの一種、通常グラフの解空間が比較的大きい場合に用いられ、探索に占める空間と時間を減らすために、幅優先探索をする際、評価値が低い枝を切り捨て、評価値が高いノードを保持する。

畳込みニューラルネットワーク (convolutional neural networks) とビームサーチ (beam search) を用いた巡回セールスマン問題について、グラフサイズが固定されている場合、このフレームワークは、解の質、推論速度、サンプル効率の面で、これまでのすべての深層学習技術を凌駕している。小さな TSP インスタンスを高速に解くことができると優れた性能を発揮するが、汎用性に欠ける [9]。

# 第3章　時間枠付き巡回セールスマン問題の解析

本節では近来時間枠付き巡回セールスマン問題 (Traveling Salesman Problem With Time Windows) で用いられている定式化について述べる。3.2、3.3 節では巡回セールスマン問題に対する Miller-Tucker-Zemlin によって提案されたポテンシャル定式化の拡張と持ち上げ操作による強化定式化を示す [1]。3.4 節では 2 添字ポテンシャル定式化を示す [1]。

## 3.1　代数定義

この問題は、n 都市の時間枠付き巡回セールスマン問題の最適解を求めると説明している。単一車両をデポ (0) に出発すると仮定し、$c_{ij}$ は都市間の移動距離で、車両速度が一定しているため、移動時間とみなし、さらに点各都市 $i$ に対する車両の出発時刻が最早時刻 $e_i$ と最遅時刻 $\ell_i$ の間でなければならないという時間制約を課した問題である。ただし、車両が最早時刻 $e_i$ より早く都市 $i$ に到着した場合には、都市 $i$ 上で最早時刻 $e_i$ まで待つ必要がある。

以下に本定式化に使用する記号を示す。

$n$: 都市の数
$c_{ij}$: 都市間の移動時間
$e_i$: 都市 i に対する出発の最早時刻
$\ell_i$: 点 i に対する出発時刻が最遅時刻
$M$: 適当な大きい値の実数
$t_i$: 点 $i$ を出発する時刻を表す変数
$x_{ij}$:0-1 変数

## 3.2　Miller-Tucker-Zemlin(ポテンシャル) 定式化

$t_i$ は以下の制約を満たす必要がある:

$$e_i \leq t_i \leq \ell_i \quad \forall i = 1, 2, \ldots, n$$

ただし、 $e_1 = 0, \ell_1 = \infty$ と仮定する。

$x_{ij}$ は 0 、 1 変数である。$x_{ij} = 1$ の場合、車両は都市 $i$ 訪問した、次に都市 $j$ を訪問する。

都市 $i$ の次に都市 $j$ を訪問する ($x_{ij} = 1$) ときには、都市 $j$ を出発する時刻 $t_j$ は、都市 $i$ を出発する時刻に移動時間 $c_{ij}$ を加えた値以上であることから、以下の式を得る。

$$t_i + c_{ij} - M(1 - x_{ij}) \leq t_j \quad \forall i, j : j \neq 1, i \neq j$$

ここで，$M$ は大きな数を表す定数である。なお、移動時間 $c_{ij}$ は正の数と仮定する。$c_{ij}$ が 0 だと $t_i = t_j$ になる可能性があり、部分巡回路ができてしまう。これを避けるためには、巡回セール

スマン問題と同様の制約を付加する必要があるが、$c_{ij} > 0$ の仮定の下では、上の制約によって部分巡回路を除去することができる。

このような大きな数 Big M を含んだ定式化はあまり実用的ではないので、時間枠を用いて強化したものを示す。

M の値はなるべく小さい方が強い制約になる。$x_{ij}$=0 のときには、上の制約は

$$M \geq t_i + c_{ij} - t_j$$

と書き出すこどができる。

$$t_i \leq \ell_i$$

$$t_j \geq e_j$$

であるので、M の値は

$$\ell_i + c_{ij} - e_j$$

以上にすれば良いです。

$$\begin{array}{lll}
minimize & \sum_{i \neq j} c_{ij} x_{ij} & \\
s.t. & \sum_{j:j \neq i} x_{ij} = 1 & \forall i = 1, 2, \ldots, n \\
 & \sum_{j:j \neq i} x_{ji} = 1 & \forall i = 1, 2, \ldots, n \\
 & t_i + c_{ij} - [\ell_i + c_{ij} - e_j]^+ (1 - x_{ij}) \leq t_j & \forall i, j : j \neq 1, i \neq j \\
 & x_{ij} \in \{0, 1\} & \forall i, j : i \neq j \\
 & e_i \leq t_i \leq \ell_i & \forall i = 1, 2, \ldots, n
\end{array}$$

## 3.3 持ち上げ操作定式化

ポテンシャル制約と上下限制約は、持ち上げ操作によってさらに以下のように強化できる。

$$t_i + c_{ij} - [\ell_i + c_{ij} - e_j]^+ (1 - x_{ij}) + [\ell_i - e_j + \min\{-c_{ji}, e_j - e_i\}]^+ x_{ji} \leq t_j \quad \forall i, j : j \neq 1, i \neq j$$

$$e_i + \sum_{j \neq i} [e_j + c_{ji} - e_i]^+ x_{ji} \leq t_i \quad \forall i = 2, \cdots, n$$

$$t_i \leq \ell_i - \sum_{j \neq 1, i} [\ell_i - \ell_j + c_{ij}]^+ x_{ij} \quad \forall i = 2, \cdots, n$$

## 3.4 2 添字ポテンシャル定式化

上のポテンシャル定式化では、大きな（"Big M"）が使われていた。（"Big M"）を含んだ定式化はあまり実用化ではないので、これを除くたに以下の新しい変数を導入する。

$$y_{ij} = \begin{cases} \text{点 i の発時刻} & x_{ij} = 1 \text{ のとき}, \\ 0 \quad \text{それ以外のとき} & \end{cases}$$

$$
\begin{array}{lll}
minimize & \sum_{i \neq j} c_{ij} x_{ij} & \\
s.t. & \sum_{j:j \neq i} x_{ij} = 1 & \forall i = 1, 2, \ldots, n \\
 & \sum_{j:j \neq i} x_{ji} = 1 & \forall i = 1, 2, \ldots, n \\
 & \sum_{i:i \neq j} y_{ij} + \sum_{i:i \neq j} c_{ij} x_{ij} \leq \sum_{k:k \neq j} y_{jk} & \forall j = 1, 2, \ldots, n \\
e_i x_{ij} \leq y_{ij} \leq \ell_i x_{ij} & \forall i, j = 1 : i \neq j & \\
 & x_{ij} \in \{0, 1\} & \forall i, j : i \neq j
\end{array}
$$

最初の 2 つの制約は、次数制約である。3 番目の制約は、$(x_{ij} = 1)$ のとき、点 i の出発時刻 $(\sum_{i:i \neq j} y_{ij})$ に i から j への移動時間 $c_{ij}$ を加えたものより、点 j の出発時刻 $(\sum_{k:k \neq j} y_{jk})$ が遅いことを表したものである。4 番目の制約は、$(x_{ij} = 1)$ のときだけ $y_{ij}$ が正の値をとれることと、時間枠制約を一緒に表したものである。

# 第4章　定式化評価実験

本節では近来時間枠付き巡回セールスマン問題 (Traveling Salesman Problem With Time Windows) で用いられている 3 種類定式化の性能と時間枠の広さと定式化の強さの関係について比較実験を行うこととする。

## 4.1　Gurobi ソルバーによる評価実験

理論と実践において、Gurobi Optimizer は、卓越した性能対価格比を備えた世界をリードする大規模オプティマイザーであることが証明されており、開発と実装におけるコストを大幅に削減できる。Gurobi Optimizer による、ユーザーは最も困難なの問題を数学モデルとして定式化し、最適なソリューションを見つけることができる。ベンチマークについて、Gurobi が競合するソルバー（CPLEX および XPress を含む）よりも高速に実行可能なソリューションを見つけることができる。

### 4.1.1　実験方法

本研究では、Dumas et al. [2] によって提案されたインスタンス”Dumas”の解決を通して、3 種類の定式化を評価実験を行うこととする。Python と数理最適化ソルバーの Gurobi Optimizer による、ベンチマーク問題の実行時間を分析して、各定式化の性能解析と比較を行う。

### 4.1.2　実験環境

以下に、実験環境を示す。
OS 名：MacBook Pro
プロセッサ：CPU が 2.3GHz デュアルコア Intel Core i5
実装メモリ (RAM)：8.0 GB
使用ソフト：python3.7.5　Gurobi Optimizer ver.9.0.0

### 4.1.3　実験結果

実験結果は下の表 4.1、4.2、4.3 のようになった。4.1、4.2、4.3 表は、ベンチマーク問題 (n は都市の数、w は時間枠の幅、同じデータ型で、1 グループあたり 5 つのデータ例がある) に対する、3 つの定式化 (mtztw はポテンシャル定式化、mtz2tw は持ち上げの操作定式化、tsptw2 は 2 添字ポテンシャル定式化を定義する) によって実行時間を含んでいる。

表 4.1: 定式化実験結果 (1)

| 実例集 | mtztw | mtz2tw | tsptw2 |
|---|---|---|---|
| n20w20 | 求解時間 (s) | 求解時間 (s) | 求解時間 (s) |
| 1 | 0.09 | 0.07 | 0.28 |
| 2 | 0.04 | 0.02 | 0.34 |
| 3 | 0.03 | 0.02 | 0.32 |
| 4 | 0.06 | 0.05 | 0.64 |
| 5 | 0.07 | 0.02 | 0.35 |
| n20w40 | | | |
| 1 | 1.13 | 0.45 | 1.47 |
| 2 | 0.12 | 0.05 | 0.24 |
| 3 | 0.25 | 0.21 | 1.11 |
| 4 | 0.44 | 0.31 | 1.45 |
| 5 | 0.55 | 0.29 | 1.22 |
| n20w60 | | | |
| 1 | 1.26 | 0.7 | 1.81 |
| 2 | 0.24 | 0.11 | 1.22 |
| 3 | 0.42 | 0.4 | 1.51 |
| 4 | 0.75 | 0.6 | 3.59 |
| 5 | 0.71 | 0.53 | 2.92 |
| n40w20 | | | |
| 1 | 1.06 | 0.33 | 10.47 |
| 2 | 0.43 | 0.42 | 4.49 |
| 3 | 0.15 | 0.14 | 3.33 |
| 4 | 0.31 | 0.21 | 5.37 |
| 5 | 0.16 | 0.11 | 3.55 |
| n40w40 | | | |
| 1 | 0.61 | 0.25 | 10.14 |
| 2 | 1.37 | 0.91 | 14.08 |
| 3 | 1.46 | 1.19 | 8.31 |
| 4 | 5.51 | 1.47 | 28.86 |
| 5 | 1.65 | 1.31 | 30.07 |

### 4.1.4 結果分析

定式化評価実験の結果を分析すると、3 つ定式化の中で求解時間最も遅いのが 2 添字ポテンシャル定式化、最も速いのが持ち上げ操作定式化である。大規模な問題例 n200w20.001 について、 2 添字ポテンシャル定式化 67475 秒たっても解を得ることができなかった。

持ち上げ操作定式化によるポテンシャル定式化などと比べて、強化すると、高速化され、大規模な問題も解けるようになる。

ベンチマーク問題 n20w20、n40w20 を例をとして、3 つ定式化による求解時間を図 4.1,4.2 に示す。

表 4.2: 定式化実験結果 (2)

| 実例集 | mtztw | mtz2tw | tsptw2 |
|---|---|---|---|
| n60w20 | 求解時間 (s) | 求解時間 (s) | 求解時間 (s) |
| 1 | 1.66 | 0.95 | 72.77 |
| 2 | 1.82 | 0.73 | 7.5 |
| 3 | 1.75 | 0.99 | 60.33 |
| 4 | 0.45 | 0.43 | 20.82 |
| 5 | 1.51 | 1.15 | 12.28 |
| n60w40 | | | |
| 1 | 6.96 | 5.99 | 62.63 |
| 2 | 23.31 | 18.2 | 105.83 |
| 3 | 44.94 | 58.58 | 603.52 |
| 4 | 6.22 | 5.57 | 44.43 |
| 5 | 5.94 | 2.56 | 67.68 |
| n80w20 | | | |
| 1 | 9.46 | 4.16 | 88.11 |
| 2 | 6.55 | 1.92 | 195.21 |
| 3 | 4.92 | 2.91 | 317.14 |
| 4 | 5.77 | 1.27 | 69.54 |
| 5 | 1.48 | 0.73 | 81.36 |
| n80w40 | | | |
| 1 | 7.42 | 1.88 | 185.52 |
| 2 | 2232.54 | 1290.13 | 178762.68 |
| 3 | 63.45 | 24.12 | 8029.29 |
| 4 | 278.65 | 342.59 | 989.36 |
| 5 | 59.74 | 31.75 | 768.29 |

## 4.2 時間枠の広さと定式化の強さの関係

定式化評価実験の結果の分析に基づいて、時間枠の広さと定式化の強さの関係と考える。実験を通じて、求解時間に対する時間枠の広さの影響を分析する。

### 4.2.1 実験方法

実験データは、インスタンス (n20w20.001、n20w20.002、n40w20.001、n40w20.002、n60w20.001、n60w20.002) を使用した。持ち上げ操作定式化による、各都市間の移動時間は変化せず (原問題例の時間枠広さは 20 である)、時間枠の広さの変化 (40,60,80,100) のみを通して実行時間の変化を調べる。

### 4.2.2 実験結果

実験結果は下の表 4.4 のようになった。

表 4.3: 定式化実験結果 (3)

| 実例集 | mtztw | mtz2tw | tsptw2 |
|---|---|---|---|
| n100w20 | 求解時間 (s) | 求解時間 (s) | 求解時間 (s) |
| 1 | 47.99 | 5.98 | 742.37 |
| 2 | 11.04 | 7.86 | 579.91 |
| 3 | 22.97 | 5.04 | 653.24 |
| 4 | 53.42 | 111.6 | 909.78 |
| 5 | 64.81 | 27.79 | 477.71 |
| n150w20 | | | |
| 1 | 389.98 | 130.95 | 2497.94 |
| 2 | 144 | 39.69 | 2426.14 |
| 3 | 44.62 | 5.46 | 1145.22 |
| 4 | 27.31 | 6.81 | 1299.46 |
| 5 | 510.2 | 467.87 | 60229.5 |

### 4.2.3 結果分析

n60w20.002 を例をとして、求解時間に対する時間枠の広さの影響を図 4.3 に示す。

時間枠の幅が大きくなると、原実行時間と比較すると、実行時間は長くなった。

以上の実験的結果を考慮すると、時間枠が広い、大規模問題の場合、計算時間を短縮するために強い定式化の利用が推奨される。

図 4.1: 3 つ定式化による求解時間 (1)

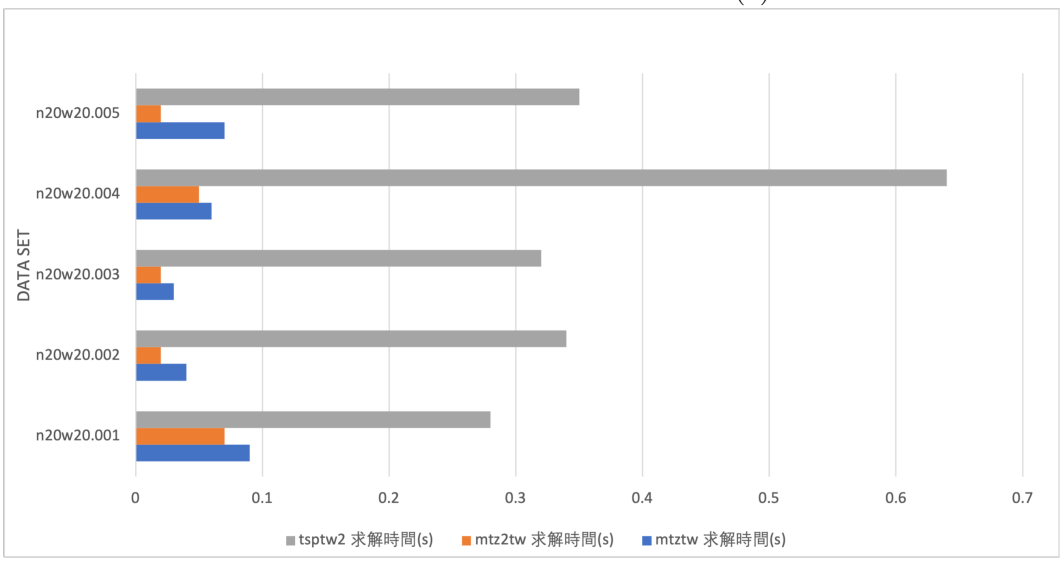



図 4.2: 3 つ定式化による求解時間 (2)

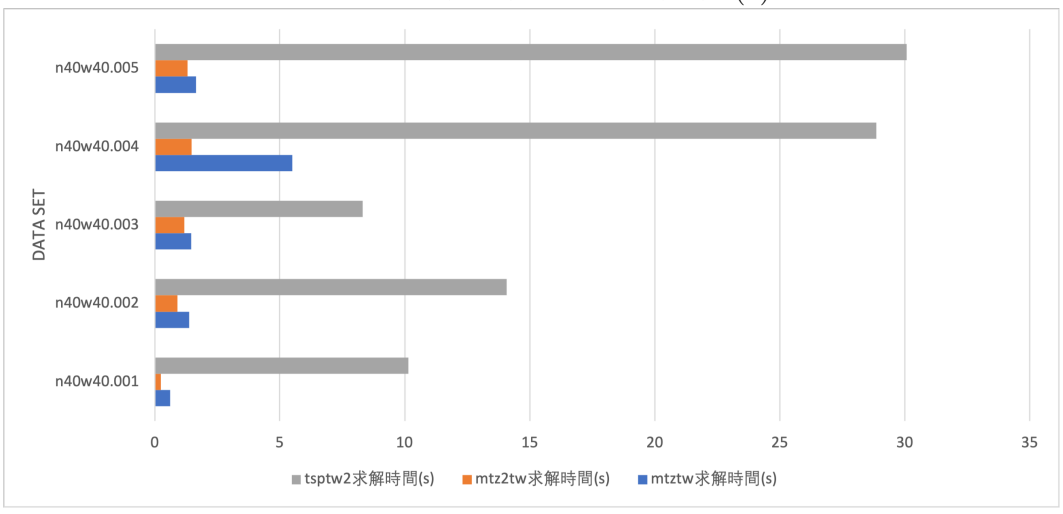
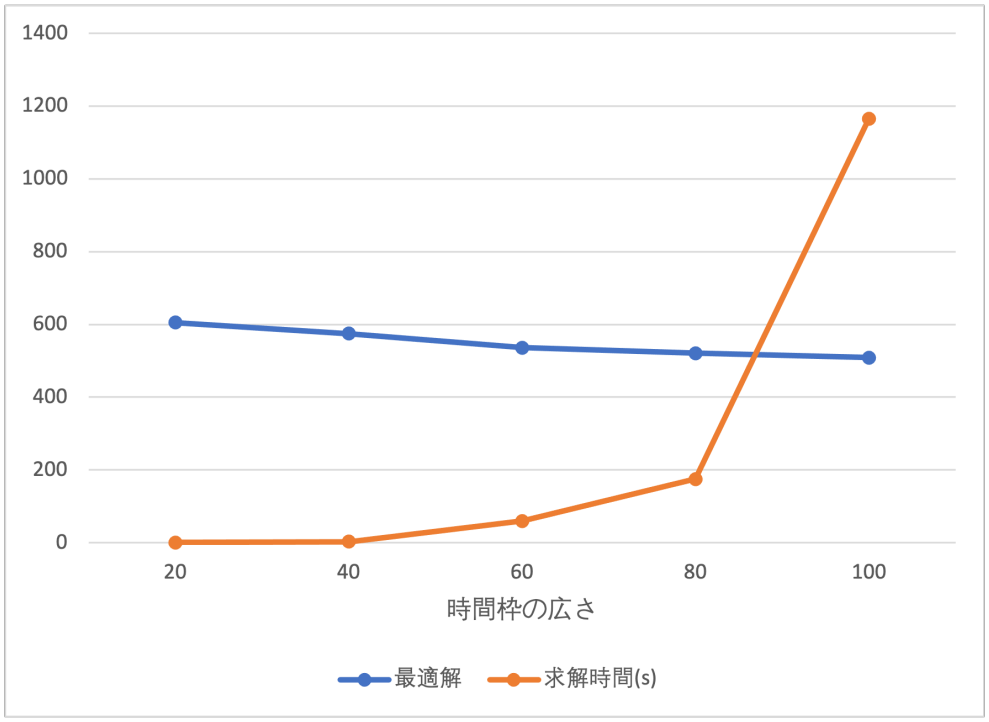


表 4.4: 時間枠の広さと定式化の強さの関係

| Data Set | Time Window Width | 最適解 | 求解時間 (s) |
|---|---|---|---|
| n20w20.001 | 20 | 378 | 0.07 |
| | 40 | 377 | 0.13 |
| | 60 | 352 | 0.1 |
| | 80 | 345 | 0.23 |
| | 100 | 313 | 2.97 |
| n20w20.002 | 20 | 286 | 0.05 |
| | 40 | 266 | 0.13 |
| | 60 | 264 | 0.46 |
| | 80 | 251 | 0.86 |
| | 100 | 223 | 0.59 |
| n40w20.001 | 20 | 500 | 0.35 |
| | 40 | 469 | 2.45 |
| | 60 | 457 | 1.9 |
| | 80 | 446 | 4.44 |
| | 100 | 446 | 433.45 |
| n40w20.002 | 20 | 552 | 0.47 |
| | 40 | 544 | 0.58 |
| | 60 | 544 | 5.18 |
| | 80 | 506 | 3.1 |
| | 100 | 463 | 5.47 |
| n60w20.001 | 20 | 551 | 0.92 |
| | 40 | 474 | 8.25 |
| | 60 | 471 | 79.15 |
| | 80 | 447 | 209.09 |
| | 100 | 418 | 208.78 |
| n60w20.002 | 20 | 605 | 0.71 |
| | 40 | 575 | 2.75 |
| | 60 | 536 | 60.24 |
| | 80 | 521 | 175.29 |
| | 100 | 509 | 1165.14 |

図 4.3: 時間枠の広さと定式化の強さの関係



# 第5章　OR-toolsによる解法

本節では、Google OR-toolsの紹介、および時間枠付き巡回セールスマン問題について、OR-toolsによる解法を述べる。

## 5.1　OR-toolsとは

Google OR-tools とは、Google のオープンソース数理最適化ツールである。車両ルーティング、フロー、整数および線形計画法、制約計画法など解決できる [3]。解決策の非常に多くの問題に対する最良の解決策を見つけようとする。

車両ルーティング問題、スケジューリング問題、ビンパッキング問題など、このような問題には非常に多くの解決策がある、検索が非常に困難になる。最適解を見つけるために、OR-Tools は車両ルーティングライブラリを使用して検索セットを絞り込み、最適な（または最適に近い）ソリューションを見つける。

OR-tools は、次のソルバーが含まれている。

- 制約プログラミング (Constraint Programming)：制約条件として表現された問題に対して、実現可能な解決策を見つける。
- 線形および混合整数計画法 (Linear and Mixed-Integer Programming)：Glop 線形オプティマイザは、制約条件として一連の線形不等式を与え、線形目的関数の最適値を求める。（例えば、仕事割当問題の最適値、コストを最小化するため、リソースの最適配分を見つける。）
- 車両ルーティング (Vehicle Routing)：制約条件下で最適な車両ルートを特定するための専用ライブラリ。
- グラフアルゴリズム (Graph Algorithms)：グラフの最短経路、最小コストフロー、最大フロー、線形和の割り当てを求めるためのソルバー。

## 5.2　ファーストソリューション戦略

ファーストソリューション戦略は、ソルバーが初期解を求めるために用いる方法である。
ルーティングの問題を解決するには、OR-Tools ソルバーには、14 種類のストラテジーが含まれている。

- AUTOMATIC: 解いているモデルに基づいて、ソルバーに使用するストラテジーを検出させる。
- PATH_CHEAPEST_ARC : ルートの「開始」ノードから始めて、最も安いルートセグメントを生成するノードに接続し、ルートに追加された最後のノードを反復してルートを拡張する。

- PATH_MOST_CONSTRAINED_ARC: 制約条件を満たすの弧を優先する。
- EVALUATOR_STRATEGY: 関数を用いてアークコストを評価する。
- SAVINGS: セービングアルゴリズム (Clarke Wright)。
- SWEEP: スイープアルゴリズム (Wren Hollida)。
- CHRISTOFIDES: クリストファイドアルゴリズム
- ALL_UNPERFORMED: 全てのノードを非アクティブにする。ノードがオプションの場合のみ、解を見つける。
- BEST_INSERTION: 最も安価なノードを最も安価な位置に挿入することにより、ソリューションを繰り返し構築する。挿入のコストは、ルーティングモデルのグローバルコスト関数に基づく。
- PARALLEL_CHEAPEST_INSERTION: 挿入のコストはアークコスト関数に基づく。BEST_INSERTION より速い。
- LOCAL_CHEAPEST_INSERTION: 挿入のコストは、アークコスト関数に基づく。挿入のために選択されたノードによって異なる。ここでは、ノードは作成順に考慮される。
- GLOBAL_CHEAPEST_ARC: 最も安いルートセグメントを生成する 2 つのノードを繰り返し接続する。
- LOCAL_CHEAPEST_ARC: バインドされていないサクセサを持つ最初のノードを選択し、それを最も安価なルートセグメントを生成するノードに接続する。
- FIRST_UNBOUND_MIN_VALUE: バインドされていないサクセサを持つ最初のノードを選択し、それを最初に使用可能なノードに接続する。

## 5.3 時間枠付き巡回セールスマン問題解決の主な手順

時間枠付き巡回セールスマン問題解決する。

- まずモジュールをインポートする。
- データモデルを作成する。データモデルには、都市間の距離行列（distance_matrix）、車両の数（num_vehicles）、場所の時間枠の配列 (time_windows)、出発地のインデックス（depot）から構成される。
  - data['distance_matrix']: 距離行列。21 の都市間の距離を表す 21 × 21 の行列です (depot(0) が含む)。配列インデックスはさまざまな都市に対応する。
  - data['time_windows']：場所の時間枠の配列。車両は、時間枠内の場所を訪問する必要がある。
  - data[‘num_vehicles’]: 車両の数。これらの 21 の都市を移動するために一つだけの運搬車を運転するとして理解できる (巡回セールスマン問題)。

- data[‘depot’] : 出発地と帰るの場所。出発地と帰るの場所インデックスを表す。ここでは 0 と定義されている。これは、最初の都市から始まり、最後に最初の都市に戻ることを定義する。

コードは：

```
def create_data_model():
    data= {}
    df=pd.read_csv('n20w20.001.csv',header=None)
    df1=np.array(df)
    data['time_matrix']=df1.tolist()
    d= {}
    for i in range(21):
        d[i]=(e1[i],l1[i])
        data['time_windows']=list(d.values())
    data['num_vehicles'] = 1
    data['depot'] = 0
    return data
```

- ソリューションプリンターを追加する。
- インデックスマネージャー（manager）とルーティングモデル（routing）を作成する。
- 時間コールバックを作成する。移動コストを定義するアークコストには、都市間の移動時間に設定する。
- 各アークのコストを定義する。車両の速度が異なる場合、場所間の移動コストを、距離を車両の速度で割った値、つまり移動時間として定義する。
- 車両の移動時間のディメンションを作成する。車両は、時間枠の間に顧客訪問する必要がある。時間枠の最小値と最大値が同じである場合、ソリューションウィンドウは単一の時点である。つまり、車両は到着したらすぐにその場所から出発する必要がある。 一方、最小値が最大値よりも小さい場合、車両は出発する前に待機することができる。
- 検索パラメータを設定する。ストラテジーを PATH_CHEAPEST_ARC に設定。未訪問ノードの重みが最小のエッジを繰り返し追加することにより、ソルバーの初期パスを作成する。

  コードは：

  ```
  search_parameters=pywrapcp.DefaultRoutingSearchParameters()
  search_parameters.first_solution_strategy = ( routing_enums_pb2.FirstSolutionStrategy.PATH_-
  CHEAPEST_ARC)
  ```

- 最短経路を解く。最短経路を解決するために routing.SolveWithParameters メソッドを利用する。

# 第6章　評価実験1

本節では、時間枠付き巡回セールスマン問題について、Gurobi と OR-tools に対して、性能を比較するために実験を行った。

## 6.1　実験方法

実験環境は CPU が 2.3GHz デュアルコア インテル Core i5，メモリが 8.00GB，実装言語には python3.7.5 を用いた。Google OR-Tools v9.0.9048 パッケージを使用した。問題インスタンスとしては、Dumas et al. によって提案されたベンチマーク問題集 (Dumas) を用いた。

上記定式化評価実験の結果の分析に基づいて、Gurobi による、持ち上げ操作定式化を利用する。OR-tools について、本実験では PATH_CHEAPEST_ARC ストラテジーを使用する。

巡回セールスマン問題について、n 顧客を訪問する可能な巡回路の総数は、称巡回セールスマン問題の場合 (n-1)!/2 個、非対称の場合 (n-1)!個となり、最適解を見つける非常に難しい、短い時間で解くことはできない。しかし、その場合、巡回路の時間制約を増やすと、解決はより困難になっている。そこで、Gurobi と OR-tools の性能を比較するために実験を行った。

## 6.2　実験結果

実験結果は下の表 6.1 のようになった。
多くの実験が行われているが、OR-tools については、小さなデータに対してのみ実行可能な解が存在し、大規模な問題については、最適解を得ることが難しい、プログラムは最後まで実行すると、実行可能な解が存在しないといった状況である。

表 6.1: 評価実験 1 結果

| DATA SET | gurobi | 時間 (s) | or-tools | 時間 (s) | Exact[ ] OptimalValue |
|---|---|---|---|---|---|
| n20w20.001 | 378 | 0.07 | 378 | 0.098 | |
| n20w20.002 | 286 | 0.02 | 286 | 0.0405 | |
| n20w20.003 | 394 | 0.02 | 394 | 0.498 | |
| n20w20.004 | 396 | 0.05 | 396 | 3.59 | |
| n20w20.005 | 352 | 0.02 | 352 | 79 | |
| 平均値 | 361.2 | | 361.2 | | 361.2 |
| n20w40.001 | 254 | 0.45 | 254 | 109 | |
| n20w40.003 | 317 | 0.21 | 317 | 196 | |
| n20w40.005 | 288 | 0.29 | 288 | 0.476 | |

# 第7章　評価実験2

本節では、OR-tools による、時間枠付き巡回セールスマン問題の初期解を求めさせるのではなく、初期経路を指定して、ベンチマーク問題に対する Gurobi ソルバーで既に良い解を見つけたので、それを出発点として修正した問題を解く、実行時間を比較するために実験を行った。

## 7.1　実験方法

インスタンスとしては、Dumas et al. によって提案されたインスタンス (Dumas) を用いた。OR-tools による、検索時の初期経路の設定の方法を利用する。OR-tools による実行時間と持ち上げ操作定式化を利用した Gurobi ソルバーによる実行時間を比較する。
初期経路を作成するには、次の通りである。

- 初期経路を含む配列を定義する。
- ReadAssignmentFromRoutes メソッドを使用して初期解を作成する。

注：初期経路には、デポは含まれていない。

## 7.2　実験結果

実験結果は下の表 7.1 のようになった。

表 7.1: 評価実験 2 結果

| DATA SET | gurobi | 実行時間 (s) | ot-tools | 実行時間 (s) |
|---|---|---|---|---|
| n60w20.001 | 551 | 0.95 | 551 | 0.126 |
| n60w20.002 | 605 | 0.73 | 605 | 0.128 |
| n60w20.003 | 533 | 0.99 | 533 | 0.101 |
| n60w20.004 | 616 | 0.53 | 616 | 0.114 |
| n60w40.002 | 621 | 18.2 | 621 | 0.154 |
| n60w40.003 | 603 | 58.58 | 603 | 0.145 |
| n60w40.005 | 539 | 2.56 | 539 | 0.107 |
| n80w20.001 | 616 | 4.16 | 616 | 0.223 |
| n80w20.002 | 737 | 1.92 | 737 | 0.227 |
| n80w20.003 | 667 | 2.91 | 667 | 0.219 |
| n80w20.004 | 615 | 1.27 | 615 | 0.204 |
| n80w20.005 | 748 | 0.73 | 748 | 0.221 |

### 7.2.1 結果分析

表 6.1,6.2 を見ると、時間枠付き巡回セールスマン問題に対して、Gurobi Optimizer と OR-tools を行ったときの計算時間と最適解をそれぞれ示す。

- Gurobi Optimizer による実験で、短時間で最適値を取ることが確認し、大規模なデータに対して解くことができる。
- OR-tools は Gurobi Optimizer と比べ、解の精度は同じが, 計算時間が長く時間がかかった。
- Google OR-tools ソルバーについて、大規模データの時間枠付き巡回路問題の最適解を見つけるさわめて難解である。
- Google OR-tools ソルバーについて、初期解がわかっている場合に初期解を出発点として問題を解くことで、実行時間を大幅に短縮することができる。

# 第8章　まとめ

本論文では，時間枠付き巡回路問題に対し、解の精度の向上、短時間で最適解を求める効率的な方法を見つけるために、さまざまな実験を行った。ベンチマークである時間枠付き巡回セールスマン問題について、実験比較するために 3 つの定式化（Miller-Tucker-Zemlin(ポテンシャル) 定式化、ポテンシャル定式化基にして、持ち上げの操作定式化、2 添字ポテンシャル定式化）の定式化の比較実験を行うこととする。比較実験によって，これらの 3 つの定式化の有効性を確かめ，ポテンシャル定式化基にして、持ち上げの操作定式化の高速化が確認した。

また、近年、組合せ最適化問題について注目されたソルバー Gurobi と OR-tools に対して、性能を比較するために実験を行った。Gurobi Optimizer ソルバーの有効性と探索効率高速化が確認した、および、OR-tools ソルバーは、時間枠の制約がある大規模な巡回セールスマン問題を解くことが困難である。

大規模な問題の最適解を求めるには計算量が多く、解の速度と精度を向上させるため、時間枠付き巡回セールスマン問題に対する他のアルゴリズムやソルバーの研究を続け、最も高速化や厳密手法を見つけることが今後の課題として考えられる。

# 謝辞

本論文を作成するにあたり、主指導教員である久保幹雄教授には、研究の着想から、実験、論文執筆まで熱心なご指導をいただきました。心から感謝申し上げます。また、所属する研究室のみなさまには多くのご協力をいただきました。誠にありがとうございました。

# 参考文献


[1] あたらしい数理最適化: Python 言語と Gurobi で解く、久保 幹雄、 ペドロソ ジョア・ペドロ、 村松 正和、 レイス アブドル (2012).

[2] Y. Dumas, J. Desrosiers, E. Gelinas, M. M. Solomon, An optimal algorithmfor the traveling salesman problem with time windows, Operations Research43 (2) (1995) 367-371.

[3] ”AboutOR-Tools.”[Online].Available:https://developers.google.com/optimization/introduction/overview

[4] 坂上知英, 吉澤慎, 太田義勝, 大山口通夫, 巡回セールスマン問題の近似アルゴリズムについて,vol.25,pp.81-96(2000)

[5] Helsgaun, K.: General k-opt submoves for the Lin - Kernighan TSP heuristic, Mathematical Programming. Computation, Vol.1, No.2, pp.119 - 163 (online), DOI: 10.1007/s12532-009-0004-6 (2009).

[6] Abdoun Otman,Jaafar Abouchabaka,A comparative study of adaptive crossover operators for genetic algorithms to resolve the traveling salesman problem,International Journal of Computer Applications 31(11):49-57(2011)

[7] Osaba, E., Carballedo, R., Diaz, F. and Perallos, A. (2013) Analysis of the Suitability of Using Blind Crossover Operators in Genetic Algorithms for Solving Rouning Problems. IEEE International Symposium on Applied Computational Intelligence and Informatics, 17-22.

[8] 三木 彰馬, 榎原 博之, 深層学習を用いた巡回セールスマン問題の解法, 情報処理学会論文誌 Vol.60 No.2 651 - 659 (Feb. 2019).

[9] Chaitanya K. Joshi, Thomas Laurent, and Xavier Bresson. “An Efficient Graph Convolutional Network Technique for the Travelling Salesman Problem”. In: CoRR(2019).