修 士 論 文 の 和 文 要 旨

研究科・専攻	大学院　情報理工学研究科　　情報・ネットワーク工学専攻　博士前期課程		
氏　　名	福田優真	学籍番号	1731132
論 文 題 目	定期航路の設計とその航路への船舶割当問題について		

要　旨

海運とは海上を利用して船によって行う輸送方法のことをいう．近年，大量の物資を一度に運べることから注目され，貿易の中心を担っている．EU では，2016 年度において価格ベースで半分程度を海運による運搬が占めている．

海運は運用形態によって,定期船と不定期船という 2 種類に大別することができる．定期船とは出発港や通る港，港を通る時間などが決められている運用形態のことを指す．主に，貨物を積み込んでいるトラックなどを直接積み込む RO-RO 船や電化製品や衣料などといった一般貨物など，定期的に必要とされるような物資の運搬に利用されることが多い．対して，不定期船は荷物を運びたい側の要求に応じて，船を所有している側が船を出す(これを傭船という)という運用形態になる．この運用形態は，鉱石や穀物など，需要が一定しないような貨物を運搬されるのに利用されることが多い．

本研究では特に定期船に着目する．つまり，船には航路が設定され，船はその航路を何度も巡回することになる．先述の，定期船は定期的に必要となる物資の運搬などに利用されるため，

なるべく移動するのにかかるコストが低いほうが嬉しい．さらに，近年は環境的な観点から，なるべく移動にコストがかからないほうがより望まれる.従って，本研究では定期船に対し，なるべくコストのかからないような航路の割当と船拍の割当を行うような最適化問題を提案する．

さらに，提案したモデルに対して，どれくらいの規模の問題をどれくらいの速度で解くことができるかや，モデルがどういったインスタンスに対してどのような挙動をするかなどを数値実験によって明らかにする．得られた実験結果から 1 期間だけ通るような航路を考えるだけでなく，複数期間を跨ぐような航路も用意した方がよりコストの低い航路を割り当てることができることがわかった．

電気通信大学大学院
情報・ネットワーク工学専攻情報数理工学プログラム修士論文

定期航路の設計とその航路への船舶割当問題について

2019 年 3 月 4 日

情報数理工学プログラム

学籍番号 1731132

福田優真

指導教員　村松正和
山本有作

目次

1　導入

1.1　定期船に対する最適化問題

海運とは海上を利用して船によって行う輸送方法のことをいう．近年, 大量の物資を一度に運べることから注目され, 貿易の中心を担っている．EU[10] では, 2016 年度において価格ベースで半分程度を海運による運搬が占めている．

海運は運用形態によって, **定期船**と**不定期船**という 2 種類に大別することができる．定期船とは出発港や通る港, 港を通る時間などが決められている運用形態のことを指す．主に, 貨物を積み込んでいるトラックなどを直接積み込む RO-RO 船 [13] や電化製品や衣料品などといった一般貨物など, 定期的に必要とされるような物資の運搬に利用されることが多い．対して, 不定期船は荷物を運びたい側の要求に応じて, 船を所有している側が船を出す (これを傭船という) という運用形態になる．この運用形態は, 鉱石や穀物など, 需要が一定しないような貨物を運搬されるのに利用されることが多い．

本研究では特に定期船に着目する．つまり, 船には航路が設定され, 船はその航路を何度も巡回することになる．先述の, 定期船は定期的に必要となる物資の運搬などに利用されるため, 移動コストは低いほうが望ましい．さらに, 近年は環境的な観点から, 移動にコストがかからないような運用が望まれている [17]．従って, 本研究では定期船に対し, なるべくコストのかからないような航路割当と船舶割当を行う最適化問題を提案する．

さらに, 提案したモデルに対して, どれくらいの規模の問題をどれくらいの速度で解くことができるかや, モデルがどういったインスタンスに対してどのような挙動をするかなどを数値実験によって明らかにする．

1.2　論文の構成

本論文では, まず, 第 2 章で読むにあたって必要な知識を記す．そして, 第 3 章で参考にした不定期船に対して提案されている最適化問題について記し, 第 4 章でその最適化問題を定期船へ拡張した結果を示す．第 5 章では提案モデルの数値実験を行い, 最後に第 6 章で本論文のまとめと今後の課題について記す．

2　準備

本章では本論で必要となる定義などを記す．

2.1　記号の定義

いくつかの記号を定義する.

まず, 実数全体の集合を $\mathbb{R}$ として, 実数全体の中から正の数だけを取り出して集めた集合を $\mathbb{R}_{>0}$ とする. すなわち,

$$\mathbb{R}_{>0} = \{\, x \in \mathbb{R} \mid x > 0 \,\}$$

である. また, $\mathbb{R}_{>0}$ に 0 を加えた集合, すなわち $\mathbb{R}_{>0} \cup \{\, 0 \,\}$ を $\mathbb{R}_{\geq 0}$ とする. すなわち,

$$\mathbb{R}_{\geq 0} = \{\, x \in \mathbb{R} \mid x \geq 0 \,\}.$$

n 次元のユークリッド空間は $\mathbb{R}^n$ と書く. ここでは, ベクトル x は全て縦ベクトルと考え, 表記する時は転置記号 T を用いて, $x = (x_1, \cdots, x_n)^T$ と記す. さらに, 自然数全体の集合を

$$\mathbb{N} = \{\, 1, 2, \cdots \}$$

とし, 有理数全体の集合を

$$\mathbb{Q} = \left\{ \frac{p}{q} \,\middle|\, p, q \in \mathbb{N} \right\} \cup \left\{ -\frac{p}{q} \,\middle|\, p, q \in \mathbb{N} \right\} \cup \{\, 0 \,\}$$

とする.

さらに n 次実正方行列全体の集合 $\mathbb{R}^{n \times n}$ とし, n 次元正方行列 $A \in \mathbb{R}^{n \times n}$ はベクトル $a_1, \cdots, a_n$ を用いて

$$A = (a_1, \cdots, a_n) = \begin{pmatrix} a_{11} & \cdots & a_{n1} \\ \vdots & \ddots & \vdots \\ a_{1n} & \cdots & a_{nn} \end{pmatrix}$$

のように表す.

2.2　行列の正定値性

行列 $A \in \mathbb{R}^{n \times n}$ が任意の 0 でないベクトル $x \in \mathbb{R}^n$ に対し,

$$x^T A x > 0$$

を満たすとき, A **は正定値である**という.

n 次実正方行列 A において,

$$Ax = \alpha x$$

を満たすような $\alpha \in \mathbb{R}$ を**固有値**, $x \in \mathbb{R}^n$ を**固有ベクトル**というが, A が正定値であることはこの固有値が全て正であることと同値であることが知られている [20].

2.3 順序

集合 X と Y に対して, その集合の直積 $X \times Y = \{ (x, y) \mid x \in X, y \in Y \}$ の任意の部分集合を X と Y **の間の関係**という. $Y = X$ のとき, すなわち $X \times X$ の時, その関係を X **上の関係**という. 組 $(x, y) \in X \times Y$ が $(x, y) \in R \subseteq X \times Y$ となるとき, x と y は R によって関係するといい, xRy と書く.

例 1. 実数の集合 $\mathbb{R}$ の部分集合 $[0, 1] \subset \mathbb{R}$ 上の関係には $\leq$ や $=$ などが上げられる.

集合 X 上の関係 R の中でも,

反射律

$$\forall x \in X; xRx$$

非対称律

$$\forall x, y \in X; xRy \land yRx \implies x = y$$

推移律

$$\forall x, y, z \in X; xRy \land yRz \implies xRz$$

を満たすようなものを, X **上の順序**という. また, 集合 X と X 上の順序 R の組 (X, R) を順序集合という.

例 2. 有理数の集合 $\mathbb{Q}$ における関係 $\leq$ などはこれらを満たす. 確認してみると,

反射律

$$\forall p \in \mathbb{Q}; q \leq q$$

非対称律

$$\forall p, q \in \mathbb{Q}; p \leq q \land q \leq p \implies q = p$$

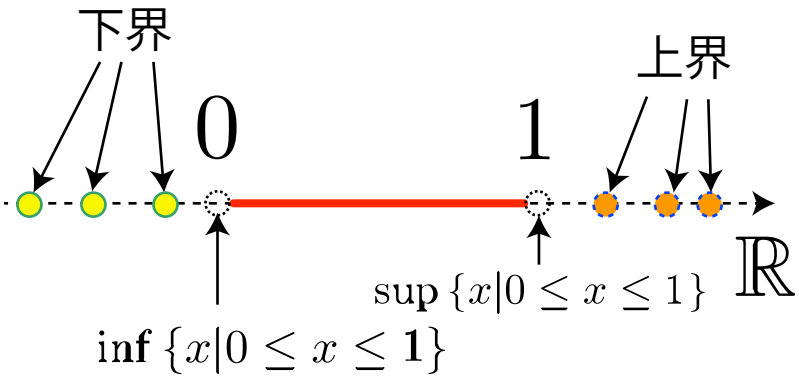

図 2.1　$(0,1)$ における上限・下限・上界・
下界の例

推移律

$$\forall p,q,r \in X; p \leq q \wedge q \leq r \implies p \leq r$$

となる．よって，$\mathbb{Q}$ 上の関係 $\leq$ は順序となることが確認できる．

2.4　上界・下界・上限・下限・最小元・最大元

集合 X 上に，順序 $\leq$ が定義されているとする．集合 X の部分集合 S に対して，$x \in X$ とした時，$\forall y \in S; y \leq x$ を満たせば x を S **の上界**といい，S **は上に有界である**という．また，$\forall y \in S; y \geq x$ を満たせば x を S **の下界**といい，S **は下に有界である**という．

例 3. 開区間 $(0,1)$ 上の順序 $\leq$ に対して，-1 は下界となり，1 は上界となる．

また，$\forall y \in S; y \leq x$ となる元 $x \in S$ のことを S **の最大元**，$\forall y \in S; y \geq x$ となる $x \in S$ のことを S **の最小元**といい，それぞれ $\max S$，$\min S$ と書く．さらに，$S' = \{\, y \in X \mid y\text{は}S\text{の上界} \,\}$ における最小元 $x \in S'$ を S **の上限**といい，$S'' = \{\, y \in X \mid y\text{は}S\text{の下界} \,\}$ における最大元 $x \in S''$ を S **の下限**といい，それぞれ $\sup S$, $\inf S$ と書く．なお，$\inf \emptyset = -\infty$, $\sup \emptyset = \infty$ と約束する．

例 4. 閉区間 $[0,1]$ 上の順序 $\leq$ に対して，最小元及び下限は 0，最大元及び上限は 1 となる．

なお，一般には最大元や最小元が存在するとは限らない．

例 5. 開区間 $(0,1)$ では，上限は 1，下限は 0 となるが，最小元と最大元は存在しない．

もし, 集合 X の部分集合 S に最大元が存在すれば, 最大元と上限は一致する.

証明. S の上界の中から任意に $x \in X$ をとってくる. すなわち, $x \in X$ は

$$\forall y \in S, y \leq x$$

を満たす. $\max S \in S$ であるので,$\max S \leq x$. また, $\max S$ は S の最大元であるので, 定義から

$$\forall y \in S, y \leq \max S$$

であるので, $\max S$ は S の上界となる. 以上から $\max S$ は S の上界全体の集合の最小元となる. 故に, $\max S$ は上限となる. □

下限も同様にして示せる.

2.5 内積

$x = (x_1, \cdots, x_n)^T, y = (y_1, \cdots, y_n)^T \in \mathbb{R}^n$ に対し,

$$\langle x, y \rangle = \sum_{i=1}^{n} x_i y_i$$

を**内積**という.

内積には性質

(1) $\forall x, y \in \mathbb{R}^n; \langle x, y \rangle = \langle y, x \rangle$
(2) $\forall x, y, z \in \mathbb{R}^n; \langle x + y, z \rangle = \langle x, z \rangle + \langle y, z \rangle$
(3) $\forall \alpha \in \mathbb{R}, \forall x, y \in \mathbb{R}^n; \langle \alpha x, y \rangle = \alpha \langle x, y \rangle$
(4) $\forall x \in \mathbb{R}^n; \langle x, x \rangle \geq 0 \wedge \langle x, x \rangle = 0 \iff x = 0$

が成立する. 実際, 次のように確認できる.

(1) $\forall x, y \in \mathbb{R}^n; \langle x, y \rangle = \sum_{i=1}^{n} x_i y_i = \sum_{i=1}^{n} y_i x_i = \langle y, x \rangle$
(2) $\forall x, y, z \in \mathbb{R}^n; \langle x + y, z \rangle = \sum_{i=1}^{n} (x_i + y_i) z_i = \sum_{i=1}^{n} x_i z_i + \sum_{i=1}^{n} y_i z_i = \langle x, z \rangle + \langle y, z \rangle$
(3) $\forall \alpha \in \mathbb{R}, \forall x, y \in \mathbb{R}^n; \langle \alpha x, y \rangle = \sum_{i=1}^{n} (\alpha x_i) y_i = \sum_{i=1}^{n} \alpha x_i y_i = \alpha \sum_{i=1}^{n} x_i y_i = \langle x, y \rangle$

(4) まず, 任意の $\forall x \in \mathbb{R}^n$ に対し, $\langle x, x \rangle \geq 0$ を示す.

$$\langle x, x \rangle = \sum_{i=1}^{n} x_i x_i = \sum_{i=1}^{n} x_i^2$$

全ての $x_i \ (i = 1, \cdots, n)$ に対し, $x_i^2 \geq 0$ であるので,

$$\langle x, x \rangle = \sum_{i=1}^{n} x_i^2 \geq 0$$

となる. 次に, $\langle x, x \rangle = 0 \iff x = 0$ を示すが, $x = 0$ ならば, 明らかに $\langle x, x \rangle = 0$ なため, $\langle x, x \rangle = 0$ ならば $x = 0$ であることを示す.

$$\langle x, x \rangle = \sum_{i=1}^{n} x_i^2$$

であるが, $x_i \ (i = 1, \cdots, n)$ に対し, $x_i^2 \geq 0$ であるため,

$$\sum_{i=1}^{n} x_i^2 = 0$$

となるためには明らかに左辺の全ての項 x_i^2 が $x_i^2 = 0$ でなければならない. 従って, $x_i = 0 \ (i = 1, \cdots, n)$.

2.6 ユークリッドノルム

$x = (x_1, \cdots, x_n)^T \in \mathbb{R}^n$ に対し,

$$\|x\| = \sqrt{\langle x, x \rangle} = \sqrt{\sum_{i=1}^{n} x_i^2}$$

をユークリッドノルムという.

ユークリッドノルムには性質

(1) $\|x\| = 0 \iff x = 0$
(2) $\forall \alpha \in \mathbb{R}, \forall x \in \mathbb{R}^n; \|\alpha x\| = |\alpha| \|x\|$
(3) $\forall x, y \in \mathbb{R}^n; \|x\|^2 \|y\|^2 \geq \langle x, y \rangle^2$
(4) $\forall x, y \in \mathbb{R}^n; \|x + y\| \leq \|x\| + \|y\|$

がある. 実際, 次のように確認できる.

(1) $\|x\| = \sqrt{\langle x, x\rangle}$ であって, $\|x\| = 0$ となるには明らかに $\langle x, x\rangle = 0$ であるが, 内積の性質から $x = 0$. また, $x = 0$ の時, 明らかに $\sqrt{\langle x, x\rangle} = \sqrt{\langle 0, 0\rangle} = 0$.

(2) 任意の $\alpha \in \mathbb{R}, \forall x \in \mathbb{R}^n$ に対し,

$$\begin{aligned}
\|\alpha x\| &= \sqrt{\langle \alpha x, \alpha x\rangle} \\
&= \sqrt{\sum_{i=1}^{n} (\alpha x_i)(\alpha x_i)} \\
&= \sqrt{\sum_{i=1}^{n} \alpha^2 x_i^2} \\
&= \sqrt{\alpha^2 \sum_{i=1}^{n} x_i^2} \\
&= |\alpha| \sqrt{\sum_{i=1}^{n} x_i^2} \\
&= |\alpha| \|x\|.
\end{aligned}$$

(3) z に関する 2 次方程式

$$\sum_{i=1}^{n} (x_i z - y_i)^2 = 0$$

を考える. 明らかに

$$\sum_{i=1}^{n} (x_i z - y_i)^2 \geq 0$$

であるため, 方程式の解の個数は 1 つ以下である. 式を展開すると,

$$\sum_{i=1}^{n} (x_i z - y_i)^2 = \sum_{i=1}^{n} (x_i^2 z^2 - 2 x_i y_i z + y_i^2) = \left(\sum_{i=1}^{n} x_i^2 \right) z - 2 \left(\sum_{i=1}^{n} x_i y_i \right) z + \sum_{i=1}^{n} y_i^2$$

となるので, 判別式 $\frac{D}{4}$ は,

$$\frac{D}{4} = \left(\sum_{i=1}^{n} x_i y_i \right)^2 - \left(\sum_{i=1}^{n} x_i^2 \right) \left(\sum_{i=1}^{n} y_i^2 \right)$$

となる. 解の個数が 1 つ以下なので,

$$\left(\sum_{i=1}^{n} x_i y_i\right)^2 - \left(\sum_{i=1}^{n} x_i^2\right)\left(\sum_{i=1}^{n} y_i^2\right) \leq 0$$
$$\iff \left(\sum_{i=1}^{n} x_i y_i\right)^2 \leq \left(\sum_{i=1}^{n} x_i^2\right)\left(\sum_{i=1}^{n} y_i^2\right)$$

すなわち, $\langle x, y\rangle^2 \leq \|x\|^2\|y\|^2$.

(4) 任意の $x, y \in \mathbb{R}^n$ に対し, $(\|x\| + \|y\|)^2 - \|x + y\|^2$ を考える. 展開すると

$$\begin{aligned}(\|x\| + \|y\|)^2 - \|x + y\|^2 &= \left(\|x\|^2 + 2\|x\|\|y\| + \|y\|^2\right) - (\langle x + y, x + y\rangle) \\ &= \left(\|x\|^2 + 2\|x\|\|y\| + \|y\|^2\right) - (\langle x, x\rangle + \langle x, y\rangle + \langle y, x\rangle + \langle y, y\rangle) \\ &= \left(\|x\|^2 - 2\|x\|\|y\| + \|y\|^2\right) - \left(\|x\|^2 + 2\langle x, y\rangle + \|y\|^2\right) \\ &= 2\|x\|\|y\| - 2\langle x, y\rangle.\end{aligned}$$

ここで, ユークリッドノルムの性質

$$\forall x, y \in \mathbb{R}^n; \|x\|^2\|y\|^2 \geq \langle x, y\rangle^2$$

を変形したもの

$$\|x\|\|y\| \geq |\langle x, y\rangle|$$

を用いて,

$$\begin{aligned}(\|x\| + \|y\|)^2 - \|x + y\|^2 &= 2\|x\|\|y\| + 2\langle x, y\rangle \\ &\geq 2|\langle x, y\rangle| + 2\langle x, y\rangle\end{aligned}$$

となる. ここで,
$\langle x, y\rangle \geq 0$ のとき　$(\|x\| + \|y\|)^2 - \|x + y\|^2 \geq 2\langle x, y\rangle + 2\langle x, y\rangle = 4\langle x, y\rangle \geq 0$
$\langle x, y\rangle < 0$ のとき　$(\|x\| + \|y\|)^2 - \|x + y\|^2 \geq -2\langle x, y\rangle + 2\langle x, y\rangle = 0$
以上から, $(\|x\| + \|y\|)^2 - \|x + y\|^2 \geq 0 \iff (\|x\| + \|y\|)^2 \geq \|x + y\|^2$. すなわち, $\|x\| + \|y\| \geq \|x + y\|$.

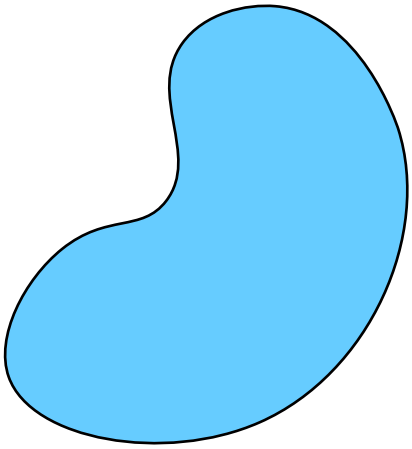

図 2.2　凸でない集合

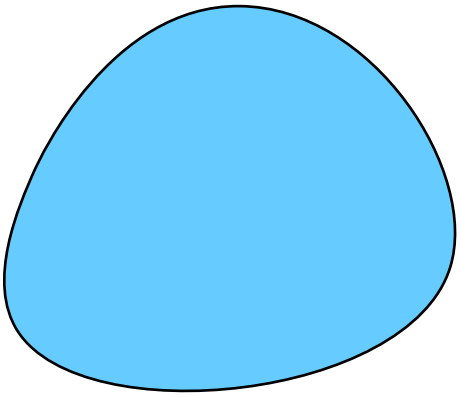

図 2.3　凸集合

2.7　凸

n 次元ユークリッド空間 $\mathbb{R}^n$ の部分集合 S が, S の任意の 2 点を結ぶ線分を含むようなとき, S を**凸集合**であるという. すなわち,

$$\forall x, y \in S, \forall \alpha \in [0,1]; \alpha x + (1-\alpha) y \in S$$

を満たすような集合 $S \subseteq \mathbb{R}^n$ のことを凸集合という. 図 2.3 を見ればわかるように, 凸集合とは凹んだ部分がないような集合のことをいう.

関数 $f : \mathbb{R}^n \to \mathbb{R}$ に対して,

$$\operatorname{graph} f = \left\{ \begin{pmatrix} x \\ \beta \end{pmatrix} \in \mathbb{R}^{n+1} \;\middle|\; x \in \mathbb{R}^n, \beta \in \mathbb{R}, f(x) = \beta \right\}$$

を関数 f の**グラフ**といい,

$$\operatorname{epi} f = \left\{ \begin{pmatrix} x \\ \beta \end{pmatrix} \in \mathbb{R}^{n+1} \;\middle|\; x \in \mathbb{R}^n, \beta \in \mathbb{R}, f(x) \leq \beta \right\}$$

を関数 f の**エピグラフ**という. エピグラフが凸である関数 $f : \mathbb{R}^n \to \mathbb{R}$ のことを凸関数という. 明らかに凸関数とは 図 2.5 のように凹んだ部分がないような関数のことをいう.

例 6. $f(x) = x^2$ や $f(x) = e^x$ は凸関数.

また, 凸関数は 図 2.5 を見ればわかるように, 凸関数 f のグラフに属す任意の 2 点を結ぶ線分がエピグラフに含まれるという性質でも特徴付けることができる. 実際, 以下の定理が成り立つ.

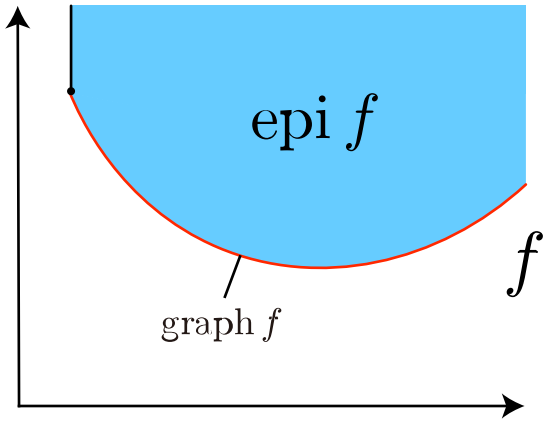

図 2.4　関数 $f(x) \geq \beta$ となる部分をエピグラフ, $f(x) = \beta$ となる部分をグラフという.

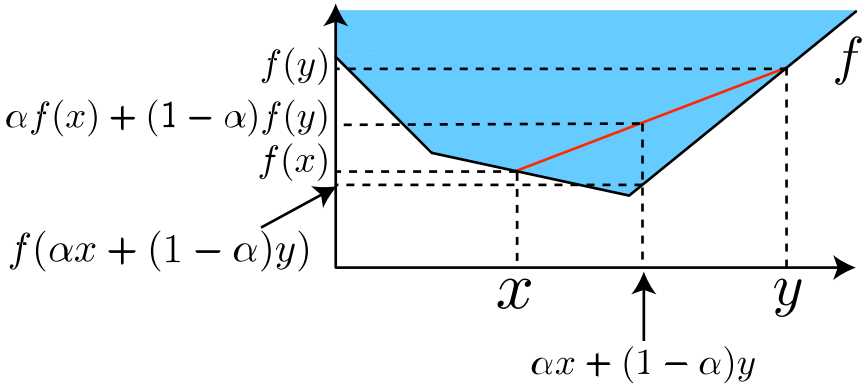

図 2.5　凸関数

定理 1. 関数 $f : \mathbb{R}^n \to \mathbb{R}$ が凸関数であることは

$$\forall x, y \in \mathbb{R}^n, \forall \alpha \in [0,1]; f(\alpha x + (1-\alpha) y) \leq \alpha f(x) + (1-\alpha) f(y) \tag{2.1}$$

と同値である.

証明. まず, 十分条件を示す. 凸関数の定義から, f が凸関数であることは, 任意の $(x, \beta)^T \in \operatorname{epi} f$, $(y, \gamma)^T \in \operatorname{epi} f$, $\alpha \in [0,1]$ に対し,

$$\alpha \begin{pmatrix} x \\ \beta \end{pmatrix} + (1-\alpha) \begin{pmatrix} y \\ \gamma \end{pmatrix} = \begin{pmatrix} \alpha x + (1-\alpha) y \\ \alpha \beta + (1-\alpha) \gamma \end{pmatrix} \in \operatorname{epi} f$$

となることである. これを書き直すと,

$$f(\alpha x + (1-\alpha) y) \leq \alpha \beta + (1-\alpha) \gamma$$

となる. 故に, この不等式が任意の $f(x) \leq \beta$, $f(y) \leq \gamma$ となる $x, y \in \mathbb{R}^n$ 及び $\beta, \gamma \in \mathbb{R}$, $\alpha \in [0,1]$ で成り立たねばならないため, $\beta = f(x)$, $\gamma = f(y)$ でも成り立たなければならない.

次に, 必要条件を示す. 任意の $(x,\beta)^T, (y,\gamma)^T \in \operatorname{epi} f$ と $\alpha \in [0,1]$ に対して,

$$\alpha \begin{pmatrix} x \\ \beta \end{pmatrix} + (1-\alpha) \begin{pmatrix} y \\ \gamma \end{pmatrix} = \begin{pmatrix} \alpha x + (1-\alpha) y \\ \alpha\beta + (1-\alpha)\gamma \end{pmatrix}$$

となる. $(x,\beta)^T$ と $(y,\gamma)^T$ は $\operatorname{epi} f$ の点なので,

$$\begin{aligned} f(x) \leq \beta &\implies \alpha f(x) \leq \alpha\beta \\ f(y) \leq \gamma &\implies (1-\alpha) f(y) \leq (1-\alpha)\gamma \end{aligned}$$

を満たす. 従って,

$$\alpha f(x) + (1-\alpha) f(y) \leq \alpha\beta + (1-\alpha)\gamma$$

である. f の性質

$$f(\alpha x + (1-\alpha) y) \leq \alpha f(x) + (1-\alpha) f(y)$$

から,

$$f(\alpha x + (1-\alpha) y) \leq \alpha f(x) + (1-\alpha) f(y) \leq \alpha\beta + (1-\alpha)\gamma$$

となる. よって, $(\alpha x + (1-\alpha) y, \alpha\beta + (1-\alpha)\gamma)^T \in \operatorname{epi} f$.

以上から, 題意は示された. □

凸関数の性質 (2.1) のうち, 等号条件を外した

$$\forall x, y \in \mathbb{R}^n, \forall \alpha \in (0,1); x \neq y \implies f(\alpha x + (1-\alpha) y) < \alpha f(x) + (1-\alpha) f(y)$$

を満たすような関数 $f : \mathbb{R}^n \to \mathbb{R}$ のことを**狭義凸関数**という.

例 7. $f(x) = x^4$ は狭義凸関数.

狭義凸関数は平坦な部分がない凸関数のことをいう.

2.8 錐

n 次元ユークリッド空間 $\mathbb{R}^n$ の部分集合 S が原点 0 を始点として, 任意の $x \in S$ を通るような半直線を含むとき, S を**錐**であるという. すなわち,

$$\forall x \in S, \forall c \geq 0; cx \in S$$

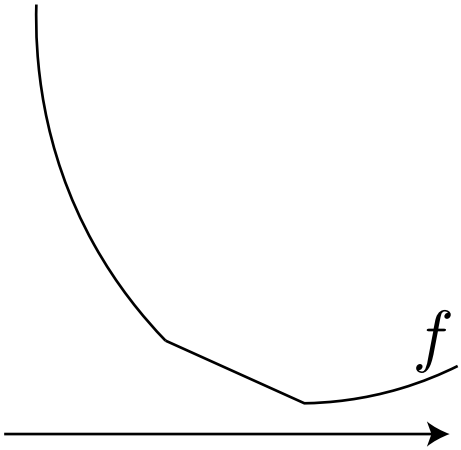

図 2.6　狭義凸でない関数

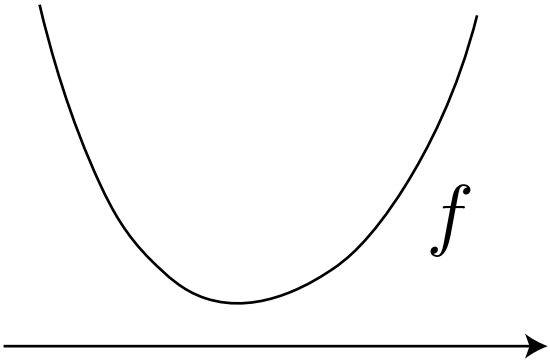

図 2.7　狭義凸関数

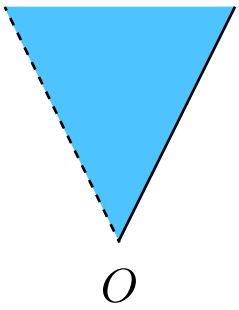

図 2.8　錐 (点線部分は含まない)

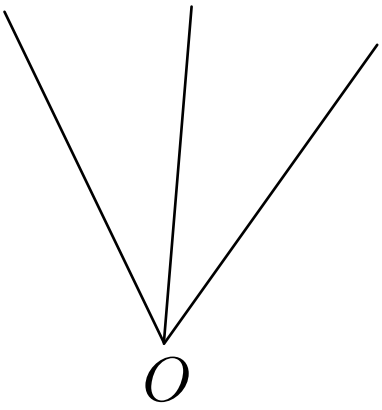

図 2.9　3 本の半直線からなる集合も錐となるが, 凸錐とはならない

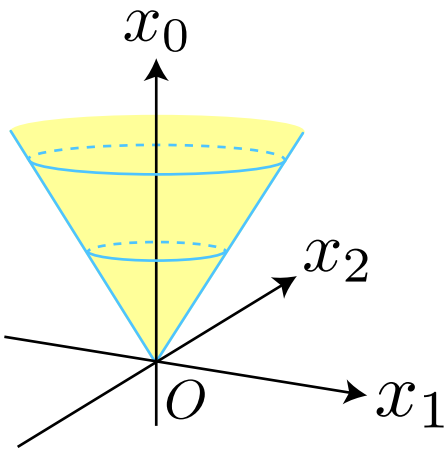

図 2.10　2 次錐

を満たすような集合 S のことを言う. つまり, 錐とは原点から放射状に伸びているような集合のことをいう.

例 8. $\{\, x \in \mathbb{R} \mid x \geq 0 \,\}$ は錐.

特に, 凸集合である錐のことを**凸錐**であるという.

例 9.

$$L = \left\{ \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{pmatrix} \in \mathbb{R}^{n+1} \;\middle|\; x_0 \geq \left\| (x_1, \cdots, x_n)^T \right\| \right\} \tag{2.2}$$

という集合は **2 次錐**と呼ばれる. この集合は凸錐となる.

実際, 錐であることは任意の $(x_0, \cdots, x_n)^T \in L$, $c \geq 0$ に対して,

$$\begin{aligned} cx_0 - \left\|(cx_1, \cdots, cx_n)^T\right\| &= cx_0 - c\left\|(x_1, \cdots, x_n)^T\right\| \\ &= c\left(x_0 - \left\|(x_1, \cdots, x_n)^T\right\|\right) \\ &\geq 0 \ \left(\because c \geq 0, (x_0, \cdots, x_n)^T \in L\right) \end{aligned}$$

となることから簡単に確認できる.

また, 2 次錐が凸となることは, 任意の $(x_0, \cdots, x_n)^T, (y_0, \cdots, y_n) \in L$, $\alpha \in [0, 1]$ に対し,

$$\alpha \begin{pmatrix} x_0 \\ \vdots \\ x_n \end{pmatrix} + (1-\alpha) \begin{pmatrix} y_0 \\ \vdots \\ y_n \end{pmatrix} = \begin{pmatrix} \alpha x_0 + (1-\alpha) y_0 \\ \vdots \\ \alpha x_n + (1-\alpha) y_n \end{pmatrix}$$

となることから, $x = (x_1, \cdots, x_n)^T, y = (y_1, \cdots, y_n)^T$ として

$$\begin{aligned} &(\alpha x_0 + (1-\alpha) y_0)^2 - \|\alpha x + (1-\alpha) y\|^2 \\ &= \left(\alpha^2 x_0^2 + 2\alpha(1-\alpha) x_0 y_0 + (1-\alpha)^2 y_0^2\right) - \langle \alpha x + (1-\alpha) y, \alpha x + (1-\alpha) y \rangle \\ &= \left(\alpha^2 x_0^2 + 2\alpha(1-\alpha) x_0 y_0 + (1-\alpha)^2 y_0^2\right) - \left(\alpha^2 \|x\|^2 + 2\alpha(1-\alpha)\langle x, y\rangle + (1-\alpha)^2 \|y\|^2\right) \\ &= \alpha^2 \left(x_0^2 - \|x\|^2\right) + (1-\alpha)^2 \left(y_0^2 - \|y\|^2\right) + 2\alpha(1-\alpha)(x_0 y_0 - \langle x, y\rangle) \\ &\geq \alpha^2 \left(x_0^2 - \|x\|^2\right) + (1-\alpha)^2 \left(y_0^2 - \|y\|^2\right) + 2\alpha(1-\alpha)(x_0 y_0 - \|x\|\|y\|) \end{aligned}$$

となる. 最後の不等号はユークリッドノルムの性質から言える. ここで,

- $x_0 \geq \|x\| \implies x_0^2 \geq \|x\|^2 \iff x_0^2 - \|x\|^2 \geq 0$
- $y_0 \geq \|y\| \implies y_0^2 \geq \|y\|^2 \iff y_0^2 - \|y\|^2 \geq 0$
- $x_0 \geq \|x\| \wedge y_0 \geq \|y\| \implies x_0 y_0 \geq \|x\|\|y\| \iff x_0 y_0 - \|x\|\|y\| \geq 0$

から, $\alpha \geq 0$ であるので,

$$(\alpha x_0 + (1-\alpha) y_0)^2 - \|\alpha x + (1-\alpha) y\|^2 \geq 0 \implies (\alpha x_0 + (1-\alpha) y_0) \geq \|\alpha x + (1-\alpha) y\|$$

となる. よって凸であることがわかる.

2.9 最適化問題

n 次元ユークリッド空間 $\mathbb{R}^n$ の部分集合 S と n 次元ユークリッド空間上の関数 $f : \mathbb{R}^n \to \mathbb{R}$ に対し, f を最小とするような解 $x \in S$ を求める問題を**最適化問題**といい,

$$\begin{array}{ll} \min & f(x) \\ \text{s.t.} & x \in S \end{array} \tag{2.3}$$

と書く. f を**目的関数**, S を**許容領域**という. S を定める条件を**制約**といい, 関数 $g_i : \mathbb{R}^n \to \mathbb{R}$ $(i = 1, \cdots, n)$, $h_j : \mathbb{R}^n \to \mathbb{R}$ $(i = j, \cdots, m)$ を用いて,

$$S = \{\, x \mid g_i(x) \leq 0, h_j(x) = 0 \ \ (i = 1, \cdots, n; j = 1, \cdots, m) \,\}$$

という不等式や等式で表される. 最適化問題は“$x \in S$”の部分に制約のみを用いて,

$$\begin{array}{ll} \min & f(x) \\ \text{s.t.} & g_i(x) \leq 0 \ \ (i = 1, \cdots, n) \\ & h_j(x) = 0 \ \ (j = 1, \cdots, m) \end{array}$$

のように書くこともある.

S の元は**許容解**という. $S = \emptyset$ であれば, 最適化問題 (2.3) は**実行不能**であるといい, $S \neq \emptyset$ であれば**実行可能**であるという.

最適化問題 (2.3) の許容解の中で, 目的関数 f を最小化する許容解 $x^* \in S$, すなわち

$$\forall x \in S, f(x^*) \leq f(x)$$

となる $x^* \in S$ のことを**最適解**という. 最適化問題 (2.3) の min の部分が max となったものを最大化問題というが, 最大化問題の時は

$$\forall x \in S, f(x^*) \geq f(x)$$

となる $x^* \in S$ のことを最適解という. ここで定義している最適解は, 特に大域的最適解と呼ばれる. 対して, ある $\epsilon > 0$ に対し, 許容解 $x' \in S$ を中心として近傍

$$B(x', \epsilon) = \{\, x \mid \|x - x'\| < \epsilon \,\}$$

と S の積 $S \cap B(x', \epsilon)$ において, $x' \in S$ が目的関数を最小とするようなもの, すなわち

$$\forall x \in B(x', \epsilon); f(x') \leq f(x)$$

を, 局所最適解というが, 一般に大域的最適解ならば局所最適解となるが, その逆は必ずしも一致するとは限らない.

,

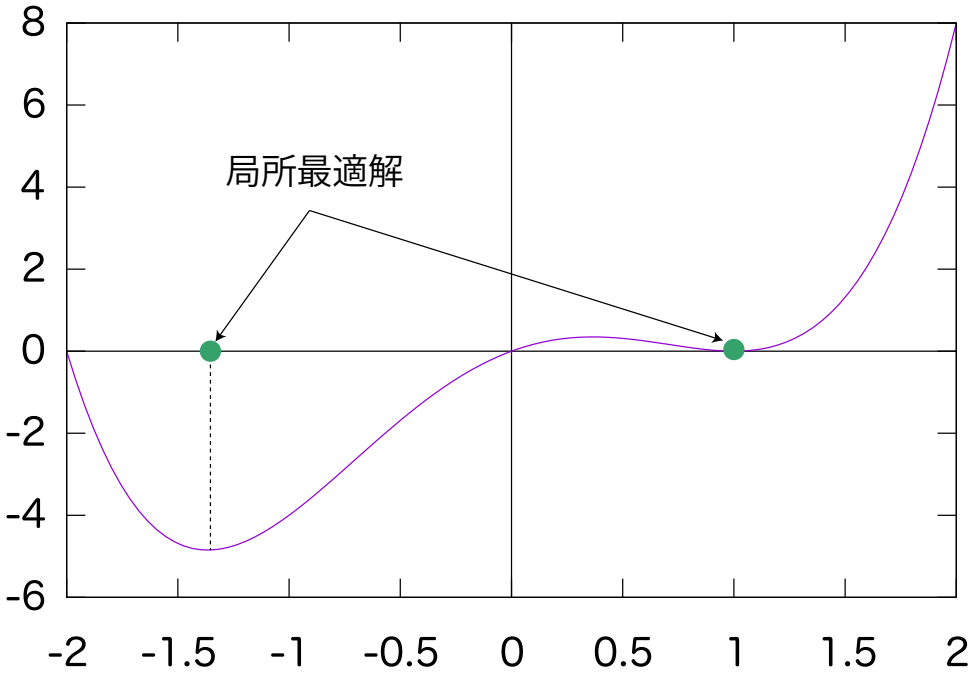

図 2.11　円の部分が例の $f(x)$ の局所最適解

例 10. 最適化問題

$$\begin{array}{ll} \min & x^4 - 3x + 2x \\ \text{s.t.} & x \in \mathbb{R} \end{array}$$

の局所最適解は $x = -\frac{1+\sqrt{3}}{2}, 1$ であり, その目的関数値は $f\left(-\frac{1+\sqrt{3}}{2}\right) = -\frac{15+6\sqrt{3}}{4}, f(1) = 0$ である.

もし, いくらでも目的関数値を下げられる場合, すなわち任意の $M \in \mathbb{R}$ に対して $f(x) < M$ となる許容解 $x \in S$ が存在する場合は, 問題が**非有界**であるといい, それ以外の場合は**有界**であるという. 最大化問題においてはいくらでも目的関数値を大きくできる場合を問題が非有界であるといい, そうでない場合を有界であるという.

例 11. 目的関数 $f : \mathbb{R} \to \mathbb{R}$ を $f(x) = x$ とした最適化問題

$$\begin{array}{ll} \min & x \\ \text{s.t.} & x \leq 0 \end{array}$$

は明らかに目的関数値をいくらでも下げられるので非有界となる.

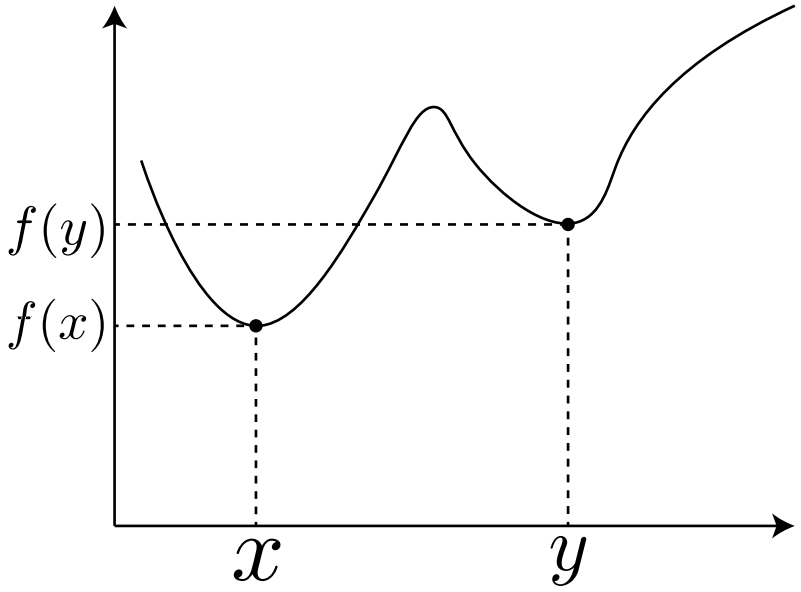

図 2.12　y は局所最適解だが, 大域最適解ではない. しかし, x は局所最適解であり, 大域最適解である

一般に, 最適化問題 (2.3) には目的関数値の下界が有限の値であっても最適解が存在するとは限らないが, 目的関数値の下限値, すなわち

$$\inf \{ f(x) \mid x \in S \} \tag{2.4}$$

を最適化問題の**最適値**であるという. 上で示したように, 最適解が存在すれば, 下限である (2.4) と最小元 $\min \{ f(x) \mid x \in S \}$ は一致する.

2.10　凸計画問題

最適化問題 (2.3) において

(1) S の不等式制約に対応する関数 $g_i : \mathbb{R}^n \to \mathbb{R}$ $(i = 1, \cdots, n)$ が凸関数, 等式制約に対応する関数 $h_j : \mathbb{R}^n \to \mathbb{R}$ $(j = 1, \cdots, n)$ が 1 次関数
(2) 目的関数 f が凸関数

であるようなものを**凸計画問題**という. 一般の最適化問題は局所最適解と大域的最適解が一致するとは限らないことを上で述べたが, 凸計画問題では局所最適解であれば必ず大域的最適解であることが保証される [1]. このクラスは効率的に解くことができる

- **錐線形計画問題**

- **2 次錐計画問題**
- **線形計画問題**

などの様々なクラスを含む.

2.11 線形計画問題

最適化問題 (2.3) の中でも,

(1) 目的関数が 1 次関数
(2) 許容領域 S の制約本数が有限
(3) 不等式制約には等号付きの 1 次不等式しか表れない
(4) 等式制約には 1 次等式しか表れない

という最適化問題, すなわち,

$$\begin{aligned} \min \quad & \sum_{i=1}^{n} c_i x_i \\ \text{s.t.} \quad & \sum_{j=1}^{n} a_{ij} x_k \leq b_i \ \ (i = 1, \cdots, m) \\ & \sum_{j=1}^{n} a_{ij} x_j = b_i \ \ (i = m+1, \cdots, l) \\ & x_i \geq 0 \ \ (i = 1, \cdots, n) \end{aligned} \tag{2.5}$$

を**線形計画問題**という. 明らかに線形計画問題は凸計画問題であるため, その取り扱いは易しい.

この線形計画問題は適用範囲の広さと, 効率的な解法の存在によって非常に重要な問題である. 最初の実用的な解法は 1947 年の Danzig により提案された単体法 [3] である. この解法は, 改良によって, 実用的には十分な大きさの問題も解くことができるようになった. しかし, 単体法には入力データサイズに対して指数的な時間が必要となる場合があることが知られており, 単体法が多項式時間解法となるかはわかっていない.

1979 年に L. G. Khachian[12] が始めての多項式時間解法である楕円体法を提案した. この解法は線形計画問題に対する多項式時間解法の提案という理論的側面では大きな貢献を果たしたが, この解法は実用には耐えなかった. この問題点を克服した多項式時間解法として, 1984 年に N. Karmarker[11] が新しい解法を提案した. 以降, 内点法と呼称される解法の研究が活発に行われるようになり, 多くの多項式時間解法が線形計画問題に対して提案されている.

2.12　2 次錐計画問題

線形計画問題 (2.5) に対し, $x_i \geq 0 \ (i = 1, \cdots, n)$ という非負性を表す制約を 2 次錐 (2.2) に表れる制約

$$x_0 \geq \left\| (x_1, \cdots, x_n)^T \right\|$$

に置き換えた問題, すなわち,

$$\begin{aligned} \min \quad & \sum_{i=0}^{n} c_i x_i \\ \text{s.t.} \quad & \sum_{j=0}^{n} a_{ij} x_j \leq b_i \ \ (i = 1, \cdots, l) \\ & \sum_{j=0}^{n} a_{ij} x_j = b_i \ \ (i = l+1, \cdots, m) \\ & x_0 \geq \left\| (x_1, \cdots, x_n)^T \right\| \end{aligned}$$

を, より一般には,

$$\begin{aligned} \min \quad & \sum_{i=1}^{m} \sum_{j=0}^{n} c_j^i x_j^i \\ \text{s.t.} \quad & \sum_{i=1}^{m} \sum_{j=0}^{n} a_{kj}^i x_j^i \leq b_k \ \ (k = 1, \cdots, l) \\ & \sum_{i=1}^{m} \sum_{j=0}^{n} a_{kj}^i x_j^i = b_k \ \ (k = l+1, \cdots h) \\ & x_0^i \geq \left\| (x_1^i, \cdots, x_n^i)^T \right\| \ \ (i = 1, \cdots, m) \end{aligned}$$

を **2 次錐計画問題**という.

この問題に対しても効率的な解法が知られている [21].

2.13　整数計画問題

線形計画問題 (2.5) に「許容解は整数値をとる」という制約を加えたものを**整数計画問題**という.

例 **12.** 最適化問題

$$\begin{aligned} \min \quad & \sum_{i=1}^{n} c_i x_i \\ \text{s.t.} \quad & \sum_{i=1}^{n} a_i x_i = b \\ & x_i \in \{\, 0, 1 \,\} \quad (i = 1, \cdots, n) \end{aligned}$$

は整数計画問題である.

整数計画問題は非常に実用的な問題であるが, その反面, 解くことは一般には難しい [19].

したがって, 厳密な最適解を得る分枝限定法などの列挙解法を利用しなければならない. しかし, 実用的な観点からより短時間でできる限り良い解を求める方法としてメタヒューリスティクスと総称される解法が多く提案されている [9].

2.14 混合整数計画問題

整数計画問題に連続値をとる変数が入っている時, その問題を**混合整数計画問題**という.

例 **13.** 最適化問題

$$\begin{aligned} \min \quad & \sum_{i=1}^{n} c_i x_i \\ \text{s.t.} \quad & \sum_{j=1}^{n} a_{ij} x_j = b_i y_i \quad (i = 1, \cdots, m) \\ & x_i \geq 0 \quad (i = 1, \cdots, n) \\ & y_i \in \{\, 0, 1 \,\} \quad (i = 1, \cdots, m) \end{aligned}$$

は混合整数計画問題である.

2.15 混合整数 2 次制約計画問題

混合整数計画問題の制約に, ベクトル $x, c \in \mathbb{R}^n$ と正定値行列 $Q \in \mathbb{R}^{n \times n}$, 定数 $c_0, b_i \in \mathbb{R}$ を用いて

$$\frac{1}{2} x^T Q x + c^T x + c_0 \geq b_i \quad (i = 1, \cdots, l)$$

と表される **2 次制約**が表れる問題を**混合整数 2 次制約問題**という.

例 **14.** 最適化問題

$$\begin{aligned}
\min \quad & \sum_{i=1}^{n} c_i x_i \\
\text{s.t.} \quad & \sum_{j=1}^{n} a_{ij} x_j = b_i y_i \ \ (i = 1, \cdots, m) \\
& \sum_{j=1}^{n} d_{ij} x_j^2 = e_i \ \ (i = 1, \cdots, l) \\
& x_i \geq 0 \ \ (i = 1, \cdots, n) \\
& y_i \in \{ 0, 1 \} \ \ (i = 1, \cdots, m)
\end{aligned}$$

は混合整数 2 次制約計画問題である.

この問題はソルバーで扱うことはできるが, 解くのが難しく, 解くまでに時間がかかってしまうことがある.

2.16 分数制約

分数制約

$$\frac{1}{x} \leq y \ \ (y > 0)$$

という制約は 2 次錐制約

$$x_0 \geq \left\| (x_1, \cdots, x_n)^T \right\|$$

に直すことができる [16]. 実際,

$$\frac{1}{x} \leq y \iff 1 \leq xy$$

となることと,

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \frac{1}{2} \begin{pmatrix} -1 & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \frac{1}{2} \begin{pmatrix} y - x \\ y + x \end{pmatrix} \iff x = y' - x', y = y' + x'$$

から

$$1 \leq xy \iff 1 \leq (y' + x')(y' - x') \iff 1 + (x')^2 \leq (y')^2$$

となるからである.

3　先行研究

Ricardo ら [6] は不定期船に対して,

- 荷物を降ろしたい港の集合
- 使用する船の数
- 出発港 (デポ)
- デポから出て港をいくつか通ってデポへ戻ってるような“航路”の集合
- 港毎に存在する, ある時刻からある時刻までに来てほしいという制約を表す“time-window”の集合
- 港毎に存在する荷物を降ろしたり, 船の清掃を行ったりなどのサービスにかかる“サービス時間”
- 港間の距離

を入力として,

(1) 航路のコストを計算し,
(2) その計算したコストを利用して船が通る航路を決定する

最適化問題を提案している [6].

本節では, 先行研究 [6] が扱っている不定期船における最適化問題とその定式化について述べる.

3.1　航路のコスト

港の集合を V, デポを $i_0, i_{h+1} \in V$(ただし, $i_0 = i_{h+1}$) とする. 途中の j 番目に訪れる港を $i_j \in V$ として, デポ i_{h+1} へ戻ってくるようなものを航路 $r = (i_0, \cdots, i_{h+1})$ という. まずは, この航路のコストを求める問題を考え, 定式化していく.

ここでは, コストの算出方法に

- 移動時にかかる燃料代など移動速度に関わるものから算出できる
- 計算する関数は狭義凸で微分可能な関数
- 計算する関数が返すのは単位距離あたりのコスト

という仮定をおいている. 実際, Fagerholt ら [4] によって, 船にかかるコストは速度 v[kn]

に応じて

$$f(v) = 0.0036v^2 - 0.1015v + 0.8848 \tag{3.1}$$

と近似できることが確認されている. なお

$$1\,\mathrm{kn} = 1\,\mathrm{M/h} = 1.852\,\mathrm{km/h}$$

である. 単位 M は距離を表す単位であり, 日本語では海里などと言ったりする. $1\,\mathrm{M} = 1.852\,\mathrm{km}$ である.

以上から, 港間 (i_j, i_{j+1}) を移動する際の移動速度を $v_{i_j,i_{j+1}} \in \mathbb{R}_{>0}[\mathrm{kn}]$ とし, 速度からコストを算出する微分可能な狭義凸関数を $c : \mathbb{R}_{>0} \to \mathbb{R}_{>0}$, 港間 (i_j, i_{j+1}) の距離を $d_{i_j,i_{j+1}} \in \mathbb{R}_{>0}[\mathrm{M}]$ とすると, 関数 c は単位距離あたりのコストを返すので, 港間の移動にかかるコストは

$$d_{i_j,i_{j+1}} c(v_{i_j,i_{j+1}})$$

となる. よって航路全体にかかるコストは

$$\sum_{j=0}^{h} d_{i_j,i_{j+1}} c(v_{i_j,i_{j+1}})$$

と計算できる.

さて, 先程書いたように j 番目に訪れる港 $i_j \in V$ には

訪問期間の制約を表す time-window: $[a_{i_j}, b_{i_j}]$

が存在している. すなわち, 港 $i_j \in V$ へ付く時刻を $t_{i_j} \in \mathbb{R}_{>0}[\mathrm{h}]$ としたとき, t_{i_j} は

$$a_{i_j} \leq t_{i_j} \leq b_{i_j} \ \ (j = 1, \cdots, h+1)$$

を満たさなければならない. 特にデポから出発するときは $t_{i_0} \in [0, 0][\mathrm{h}]$, つまり

$$t_{i_0} = 0$$

である. さらに, 当然ながら移動先があるにも関わらず移動しない船などは現実には存在しないし, 飛行機などのように高速に移動するような船も存在しないため, 船の港間 (i_j, i_{j+1}) を移動する速度 $v_{i_j,i_{j+1}} \in \mathbb{R}_{>0}[\mathrm{kn}]$ には下限値 $v_{\min}(> 0)[\mathrm{kn}]$ と上限値 $v_{\max}(> v_{\min})[\mathrm{kn}]$ が設けられている. つまり, 船の移動速度には

$$v_{\min} \leq v_{i_j,i_{j+1}} \leq v_{\max} \ \ (j = 0, \cdots, h)$$

という制約がかけられる. また, 船が港 $i_{j+1} \in V$ に着く時刻 $t_{i_{j+1}} \in \mathbb{R}_{>0}[\mathrm{h}]$ は,

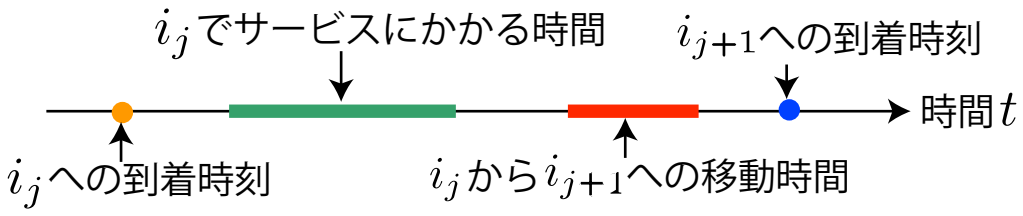

図 3.1　t_{i_j} と $t_{i_{j+1}}$ の関係

表 3.1　航路のコストを計算する時に使用する定数

記号	意味
$(i_0, \cdots, i_{h+1})$	航路
$\tau_{i_j} \in \mathbb{R}_{>0}$	港 i_j におけるサービス時間 [h]
$[a_{i_j}, b_{i_j}] \subseteq \mathbb{R}_{>0}$	港 i_j における time-window [h]
$v_{\min} \in \mathbb{R}_{>0}$	速度の下限値 [kn]
$v_{\max} > v_{\min}$	速度の上限値 [kn]
$d_{i_j,i_{j+1}} \in \mathbb{R}_{>0}$	港 i_j と i_{j+1} の距離 [M]
$c : [v_{\min}, v_{\max}] \to \mathbb{R}_{>0}$	速度からコストを算出する関数

表 3.2　航路のコストを計算する時に使用する変数

記号	意味
$t_{i_j} \in [a_{i_j}, b_{i_j}]$	港 i_j へ着く時刻 [h]
$v_{i_j,i_{j+1}} \in [v_{\min}, v_{\max}]$	港 i_j から i_{j+1} への移動速度 [kn]

- 1 つ前の港に着いた時刻 $t_{i_j} \in \mathbb{R}_{>0}$[h]
- 移動時間 $d_{i_j,i_{j+1}}/v_{i_j,i_{j+1}}$[h]
- 1 つ前の港でサービスにかかった時間 $\tau_{i_j} \in \mathbb{R}_{>0}$[h]

の和以上となる必要がある．すなわち，

$$t_{i_{j+1}} \geq t_{i_j} + \tau_{i_j} + \frac{d_{i_j,i_{j+1}}}{v_{i_j,i_{j+1}}} \quad (j = 0, \cdots, h)$$

を満たしている必要がある．

まとめると，まず，入力は 表 3.1 と 表 3.2 の記号を利用して，

- デポ i_0 から出ていくつか港 i_j を通ってデポ $i_{h+1}(= i_0)$ へ戻ってくるような航路 $r = (i_0, \cdots, i_{h+1})$

- 港 i_j 毎に存在する, ある時刻からある時刻までに来てほしいという制約を表す“time-window”の集合
- 港 i_j 毎に存在する荷物を降ろしたり, 船の清掃を行ったりなどのサービスにかかる“サービス時間”
- 港間 (i_j, i_{j+1}) の距離

となる. また, 制約は

(1) time-window 制約

$$a_{i_j} \leq t_{i_j} \leq b_{i_j} \ \ (j = 1, \cdots, h+1)$$

(2) デポの出発時間に関する制約

$$t_{i_0} = 0$$

(3) 速度制約

$$v_{\min} \leq v_{i_j, i_{j+1}} \leq v_{\max} \ \ (j = 0, \cdots, h)$$

(4) 移動先の港に着く時刻の制約

$$t_{i_{j+1}} \geq t_{i_j} + \tau_{i_j} + \frac{d_{i_j, i_{j+1}}}{v_{i_j, i_{j+1}}} \ \ (j = 0, \cdots, h)$$

の 4 つを守らなければならない. 以上のもと, 移動コスト

$$\sum_{j=0}^{h} d_{i_j, i_{j+1}} c(v_{i_j, i_{j+1}})$$

を最小にするような最適化問題は

$$\begin{aligned} \min \quad & \sum_{j=0}^{h} d_{i_j, i_{j+1}} c(v_{i_j, i_{j+1}}) \\ \text{s.t.} \quad & t_{i_{j+1}} \geq t_{i_j} + \tau_{i_j} + \frac{d_{i_j, i_{j+1}}}{v_{i_j, i_{j+1}}} \ \ (j = 0, \cdots, h) \\ & a_{i_j} \leq t_{i_j} \leq b_{i_j} \ \ (j = 1, \cdots, h+1) \\ & t_{i_0} = 0 \\ & v_{\min} \leq v_{i_j, i_{j+1}} \leq v_{\max} \ \ (j = 0, \cdots, h) \end{aligned} \tag{3.2}$$

と定式化される. この最小値を航路 $r = (i_0, \cdots, i_{h+1})$ のコストと定義する.

3.2 航路の決定

与えられた航路 $r \in R$ 全てに対して, 各々のコスト $c_r \in \mathbb{R}_{>0}$ が計算できたら, 次に使用する航路を決定する. 航路を船が使用するか否かを表す変数を $z_r \in \{0, 1\}$ とする. すなわち, 航路を通過する船があるとき $z_r = 1$ であり, ないときは $z_r = 0$ である. この変数を用いて, 全体としてのコストは, 使用する航路のコストの総和, すなわち,

$$\sum_{r \in R} c_r z_r$$

と表すことができる. なお, 航路 $r \in R$ のコストが計算できない, すなわち, 航路 $r \in R$ に対して, (3.2) の実行可能解がない場合, そのコストは $c_r = \infty$ と設定することに注意する.

本問題では, このようなコスト $c_r \in \mathbb{R}_{>0}$ のかかる航路 $r \in R$ を通る船は $K \in \mathbb{N}$ 隻与えられ, 各々の船をどれか 1 つの航路に割り当てるようにする. 全ての使用航路数は $\sum_{r \in R} z_r$ と書けるので, 各々の船が 1 つの航路を使用するように航路を割り当てるという制約は

$$\sum_{r \in R} z_r = K$$

と書ける. また, 与えられている港 $i \in V$ は全て訪問するようにしたいと考える. しかし, 何度も港 $i \in V$ を訪れるのも無駄である. そこで, 港 $i \in V$ を通るような航路 $r \in R$ は一つだけと約束する.

だが, $r \in R$ の情報だけではこれは実現できない. これを解決するために, $\alpha_{ir} \in \{0, 1\}$ という, 港 $i \in V$ が航路 $r \in R$ 上にあるとき 1, それ以外を 0 とする定数を導入する. このような定数を導入すると, $r \in R$ が港 $i \in V$ を訪ずれているか否かは

$$\sum_{r \in R} \alpha_{ir} z_r$$

と表すことができる. 先程約束したように, 今, 港はただ 1 つの航路のみが通るように約束しているので, この制約は

$$\sum_{r \in R} \alpha_{ir} z_r = 1 \ \ (i \in V)$$

と表すことができる.

以上から, 表 3.3 と 表 3.4 を使って,

表 3.3　使用航路を決定する時に使用する定数

記号	意味
R	航路の集合
V	デポを除いた港の集合
$c_r \in \mathbb{R}_{>0}$	航路 r のコスト
$\alpha_{ir} \in \{0, 1\}$	航路 r が港 $i \in V$ を通る時 1, それ以外 0
$K \in \mathbb{N}$	使用したい船の数

表 3.4　使用航路を決定する時に使用する変数

記号	意味
$z_r \in \{0, 1\}$	航路 $r \in R$ を使用する時 1, それ以外 0

(1) 船の数に対する制約

$$\sum_{r \in R} z_r = K$$

(2) 訪問制約

$$\sum_{r \in R} \alpha_{ir} z_r = 1 \ \ (i \in V)$$

という制約のもと, 使用する航路のコストの総和

$$\sum_{r \in R} c_r z_r$$

を最小とするような問題は

$$\begin{aligned} \min \quad & \sum_{r \in R} c_r z_r \\ \text{s.t.} \quad & \sum_{r \in R} \alpha_{ir} z_r = 1 \ \ (i \in V) \\ & \sum_{r \in R} z_r = K \\ & z_r \in \{0, 1\} \ \ (r \in R) \end{aligned}$$

と定式化できる.

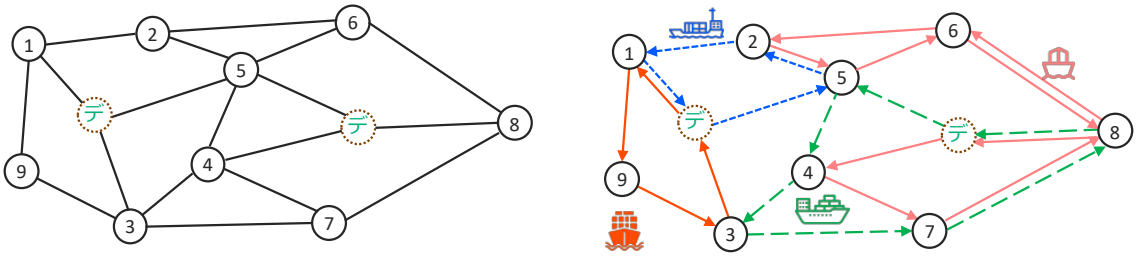

図 4.1　左のようなネットワークに右のように航路割り当て・船の割り当てを行う (“デ”と書いてあるのは出発港を指す)

4　提案モデル

本論文では先程述べた不定期船に対する最適化問題を元に, 定期船に対し, 航路を決定する問題を定式化し, さらに種別の違う船の割当を行う問題を定式化した.

導入でも簡単に触れたが, 不定期船は船の利用者の需要によって, 利用者の time-window 内に利用者の元へ着くように船を移動させなければならない. つまり, 船が動く航路は定まっているわけではなく, その時々の需要に応じて移動する航路は変わることとなる. 対して, 定期船は先に船がどこをどのように通るかが定められている. すなわち, 不定期船のように航路を柔軟に変更することはできず, 一定期間内はその割り当てられた航路を巡回することとなる. 利用者は, 船がどこへどの期間に行くのかを見てどこへどのように物資を運搬するかを決定する. この定期船での性質を利用して定式化を行っていく.

定式化するにあたって, 1 度で航路・船舶割当を決定するようにするのではなく,

(1) 全てのに time-window 内に港を訪ずれるように使用する航路をコストが最小となるよう割当
(2) 割当航路に対して船の割当を決定

の順番で決定する. これは, 一度に航路と船の割当をしようとすると 2 次制約が表れるため, モデル化したい問題が混合整数 2 次制約問題として定式化されるのを避けることが目的である.

本章の続きでは, これらについて更に詳細に記述する.

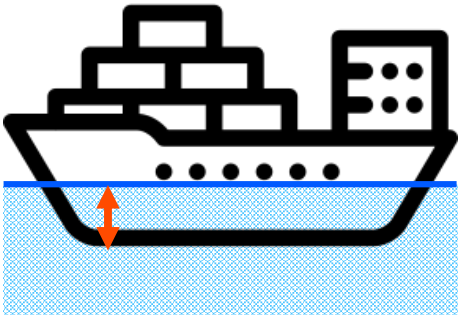

図 4.2　矢印部分が喫水

4.1　喫水

船が水上にあるとき, 船が水に沈む深さのことを**喫水**という. 喫水は安全な航行を確保したり, 貨物を積み込む量についての重要な指標となる.

貨物を積み込めば積み込む程, 当然ながら船はより深く水に沈むこととなる. つまり, 水面から甲板までの距離がより短くなる. 従って, あまりに大量に貨物を積み込んでしまうと沈没する危険性が高まる. 逆に, 貨物を降ろせば降ろす程, より水面から船底までの距離が短くなり, より船が転覆しやすくなってしまう. また, 甲板から水面までの高さが上がってしまうと, 小型船などを甲板上から視認し辛くなってしまうという危険性も孕んでいる.

そこで, 水を入れてこの喫水を調整することがある. 調整のために入れられる水はバラスト水と呼ばれる. 貨物があまり載っていないようであれば, バラスト水を入れ, 貨物が積載されている場合は, 逆にバラスト水を放出するという方法で喫水の調整を行なっている.

4.2　タイムテーブルと航路

本章で扱う問題は全体の期間を一定期間に区切り, その区切られた期間内には, 他の全ての期間と同じパターンの, 港毎に来て欲しい期間 (time-window) が設定されていると仮定する. このような港毎に来て欲しい期間が, 区切られた期間内に設定され, それが周期的に繰り返しているようなものを**タイムテーブル**という. 注意して欲しいのは, 本章で扱う問題では区切る期間の長さは一定であると仮定している. 例えば, 全体の期間を 8 週間として, 区切る長さは 1 週間とする, など一定の期間で区切っているということに注意する.

さらに, 先にも指摘した通り, 期間内に設定される港毎に来て欲しい期間は期間が周期的になっているように仮定しているため, 全ての期間で同じパターンを形成している. 例えば, 全体の期間を 4 週間として区切る長さを 1 週間とし, 1 週間目にある港に

- 火曜日の 10:00 から 14:00
- 金曜日の 12:00 から 15:00

という来て欲しい期間が設定されていた場合, 2 週間目や 3 週間目にも同じ日同じ時刻に来て欲しい期間が設定される.

本章で扱う問題の**航路**は, 出発港 (**デポ**) から出発し, タイムテーブルの中からいくつかの期間を通り, 出発したデポと同じデポへ戻るようなものを指す. なお, 航路はタイムテーブルの最後の期間まで周期的に繰り返されているものとする. また, 航路はタイムテーブルを循環しているものとして用意する. すなわち, タイムテーブル全体の期間数を航路が跨っている期間数で割きれない場合はタイムテーブルを循環しているようにして再度 1 期間目から航路を伸ばすようにする. これは例えばタイムテーブルの 1 期間の長さを 1 週間, タイムテーブル全体として 4 期間用意し, ある航路が 3 期間跨っているような場合, その航路は最後の 4 期目を通ったあと, 1 期間目からその続きを伸ばすというようにする. この詳細は例で述べる.

図 4.3 はタイムテーブルの例である. まず, 右方向へ行くに従って時間が進む. 縦軸は港を表している. 期間は真ん中の縦線で区切られており, 港には両矢印で表されているような time-window が設置される. 1 期と 2 期を見ればわかるように各港の time-window は期が変わっても同じ位置に同じような time-window が設置されていることが確認できる.

また, 図 4.4 は航路を表す例である. この例でのタイムテーブルの期間は 3 期用意されている. 矢印で結ばれているものがそれぞれ航路となる. 航路が 1 周期内に横切っている, 区切っている期間の個数をもとに跨っていると表現される. この例では実線の矢印で示された 1 期間跨っている航路と, 点線の矢印で結ばれた 2 期間跨っている航路の 2 つが描かれている. 今回は 3 期までしかないので, 点線の矢印で結ばれた 2 期間跨るような航路は 3 期目を通ったあと, 1 期目からその続きを伸ばすようにしている. コストは航路の 1 周期を使って, その 1 周期分算出される. 航路自体は 1 期間目から初める必要はなく, 2 期間目や 3 期間目から始まっていても構わない. このような場合も航路の最初の 1 周期目からその 1 周期分算出される.

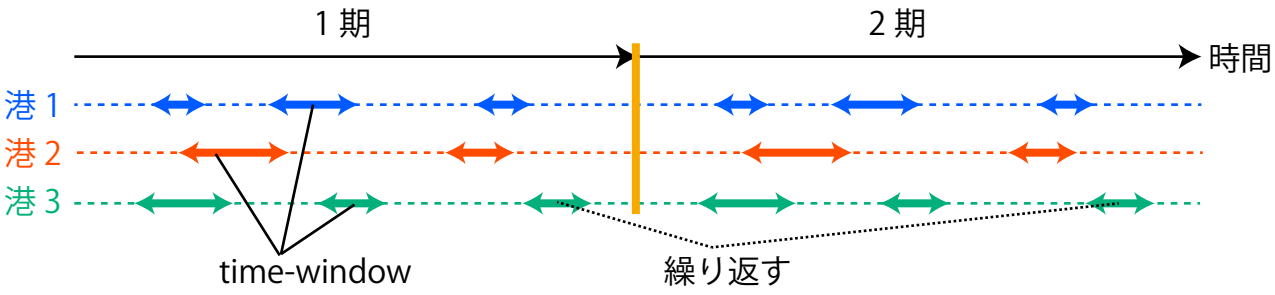

図 4.3　タイムテーブルの例

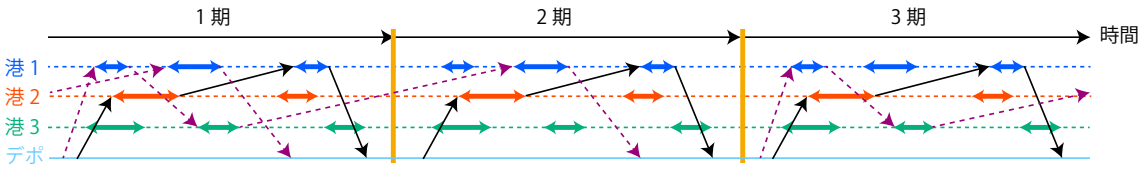

図 4.4　矢印が航路．3 期で終了．点線の矢印で表されている航路のように最後の周期から 1 期目へ循環するようになっている航路もある．

4.3　航路割当

まず, 与えられた航路の集合から割り当てたい航路を決定する．航路を決定する問題は以下の入力をもつ:

- 全体の期間をある一定区間に区切り, その区切った期間内に, 港毎に来て欲しい期間, すなわち time-window がいくつか設定されたタイムテーブル
- デポから出てそのタイムテーブル内のいくつかの time-window を通って元のデポへ戻るような航路を集めた集合 R
- 航路のコスト $c_r \in \mathbb{R}_{>0}$.

なお, 各航路のコスト $c_r \in \mathbb{R}_{>0}$ は先行研究 [6] の航路のコストを計算する問題などによって事前に計算しておく．また, 航路のコストが計算できなかった場合は航路にかかるコストを ∞ と設定すると約束する．

本問題で扱う航路 $r \in R$ は全体の期間が終了するまで周期的に繰り返している．そのため, 各航路は, タイムテーブルにおける区切られた期間の通る数に応じて $l_r \in \mathbb{N}$ 周する．もちろん, この $l_r \in \mathbb{N}$ は, 航路 $r \in R$ 毎に通る区切られた期間の数が異なるために, 一般には航路毎に異なった値となる．さらに, 航路の周期を $l = 1, \cdots, l_r$ として表す．すなわ

ち, 2 周目の航路は $l = 2$ となる. また, 同じようにタイムテーブルは周期的に区切られた期間が繰り返している. 従って, 区切られた期間を参照しやすくするために, 区切られた期間には一番始めの期間を 1 としたインデックス t を設定する. なお, 一番最後の区切られた期間に設定される期間は t_{last} とする. 例えば, 8 週間を全体の期間として区切る長さを 1 週間としたとき, 4 週間目を表すインデックスは 4 となる. また, 区切られた期間の中には time-window が設定されているが, 時間の早いものから順番にインデックスを 1 から 1 つずつ大きくして割り当てる. このようなインデックスを $j \in \mathbb{N}$ とし, 港 $i \in U$ における最も最後に設定されている time-window のインデックスを j^i_{last} とする.

これらの定数を用いて定式化を行っていく. まず, 全体にかかるコストを考える. 航路 $r \in R$ の $l(= 1, \cdots, l_r)$ 週目が割り当てられるとき 1, それ以外 0 となる変数を $x_{rl} \in \{0, 1\}$ とすると, 全体にかかるコストは

$$\sum_{r \in R} \sum_{l=1}^{l_r} c_r x_{rl}$$

と表される.

航路 $r \in R$ が割り当てられる場合, 定期船なので, $l(= 1, \cdots l_r)$ 週全部を利用して欲しい. 従って,

$$\sum_{l=1}^{l_r} x_{rl} = l_r x_{r1} \ \ (r \in R)$$

という制約が付く. また, 設定されている全ての time-window 内に港を訪れるようにしたい. これは航路 $r \in R$ の情報だけではカバーできないので, 期間 t における港 $i \in U$ の $j(= 1, \cdots, j^i_{\text{last}})$ 番目の time-window を航路 $r \in R$ の $l(= 1, \cdots, l_r)$ 週目が通るとき 1, それ以外 0 とする定数 α^{rl}_{ijt} を導入して定式化を行う. この定数を導入することで, 設定されている全ての time-window 内に港を訪れなければならないという制約は

$$\sum_{r \in R} \sum_{l=1}^{l_r} \alpha^{rl}_{ijt} x_{rl} \geq 1 \ \ (i \in U; j = 1, \cdots, j^i_{\text{last}}; t = 1, \cdots, t_{\text{last}})$$

と表すことができる.

以上から, モデルは 表 4.1 と 表 4.2 の記号を利用して, 制約

(1) 航路の周期性

$$\sum_{l=1}^{l_r} x_{rl} = l_r x_{r1} \ \ (r \in R)$$

表 4.1　航路の割当に使用する定数

記号	意味
R	航路の集合
U	荷物を降ろす港の集合
$\alpha_{ijt}^{rl} \in \{0,1\}$	港 $i \in U$ における期間 $t \in \{1,\cdots,t_{\text{last}}\}$ の $j \in \{1,\cdots,j_{\text{last}}^{i}\}$ 番目の time-window を航路 $r \in R$ の $l \in \{1,\cdots,l_r\}$ 周目が通るとき 1, それ以外 0

表 4.2　航路の割当に使用する変数

記号	意味
$x_{rl} \in \{0,1\}$	航路 $r \in R$ の $l = 1,\cdots l_r$ 周目を利用するとき 1, それ以外 0

(2) 全 time-window を通過しなければならないという制約

$$\sum_{r\in R}\sum_{l=1}^{l_r} \alpha_{ijt}^{rl} x_{rl} \geq 1 \ \ (i \in U; j = 1,\cdots,j_{\text{last}}^{i}; t = 1,\cdots,t_{\text{last}})$$

を満たすように全体のコスト

$$\sum_{r\in R}\sum_{l=1}^{l_r} c_r x_{rl}$$

を最小にするような航路を割り当てる問題は

$$(\text{RAP}) = \begin{cases} \min & \displaystyle\sum_{r\in R}\sum_{l=1}^{l_r} c_r x_{rl} \\ \text{s.t.} & \displaystyle\sum_{l=1}^{l_r} x_{rl} = l_r x_{r1} \ \ (r \in R) \\ & \displaystyle\sum_{r\in R}\sum_{l=1}^{l_r} \alpha_{ijt}^{rl} x_{rl} \geq 1 \ \ (i \in U; j = 1,\cdots,j_{\text{last}}^{i}; t = 1,\cdots,t_{\text{last}}) \\ & x_{rl} \in \{0,1\} \ \ (r \in R; l = 1,\cdots,l_r) \end{cases} \tag{4.1}$$

と定式化できる.

4.4 船舶割当

船舶を割り当てる航路を決定したら, その航路へ船舶を割り当てていく. 本問題でも航路の割り当て時に使用したタイムテーブルを利用する. なお, 入力に使用する航路の集合は先の航路決定問題 (RAP) で使用することが決定した航路の集合とする.

割り当てたい航路の集合を $\mathcal{R}$ とし, 割り当てたい船の集合を S とする. 船 $s \in S$ にはそれぞれ容量 $Q_s \in \mathbb{R}_{>0}[\mathrm{t}]$ と喫水 $d_s \in \mathbb{R}_{>0}[\mathrm{m}]$, それに価格 $p_s \in \mathbb{R}_{>0}$ が設定されている. また, 港にはそれぞれ海面から海底までの深さが設定されており, それを元に航路上の最も小さい海面から海底までの深さが算出される. タイムテーブル中の区切られた期間 $t(=1,\cdots,t_{\mathrm{last}})$ 中の港 $i \in U$ の $j(=1,\cdots,j^i_{\mathrm{last}})$ 番目の time-window における貨物需要を $e_{ijt} \in \mathbb{R}_{>0}[\mathrm{t}]$ とする. 航路 $r \in \mathcal{R}$ の中で最小となる海面から海底までの深さを $h_r \in \mathbb{R}_{>0}[\mathrm{m}]$ とする.

船にはコスト p_s がかかるので, 船 $s \in S$ が航路 $r \in \mathcal{R}$ の $l(=1,\cdots l_r)$ 周目を通るとき 1, それ以外 0 となる変数を $x_{rls} \in \{0,1\}$ とすると, 航路の最後の周が $l_r \in \mathbb{N}$ とすると, 船を使用する時にかかる全体のコストは,

$$\sum_{r\in\mathcal{R}}\sum_{s\in S}\sum_{l=1}^{l_r} p_s \frac{x_{rls}}{l_r}$$

となる. 船が r を通るときは $l(=1,\cdots,l_r)$ 周を全て利用して欲しいので,

$$\sum_{l=1}^{l_r} x_{rls} = l_r x_{r1s} \ \ (s \in S; r \in \mathcal{R})$$

という制約をかける. また, 使用が決定している航路は全て利用したいので,

$$\sum_{s\in S} x_{rls} = 1 \ \ (r \in \mathcal{R}; l = 1,\cdots,l_r)$$

という制限をかける. さらに, 船には高々 1 つしか航路を割り当てないように

$$\sum_{r\in\mathcal{R}}\sum_{l=1}^{l_r} \frac{x_{rls}}{l_r} \le 1 \ \ (s \in S)$$

という制約をつける. 船は港に貨物を運んでいるが, 港側の貨物需要を守るようにしたい. しかし, 航路の情報だけでは各港の情報を表すことができないので, 港 $i \in U$ の期間 $t(=1,\cdots,t_{\mathrm{last}})$ における j 番目の time-window を航路 $r \in \mathcal{R}$ の $l(=1,\cdots,l_r)$ 周目が

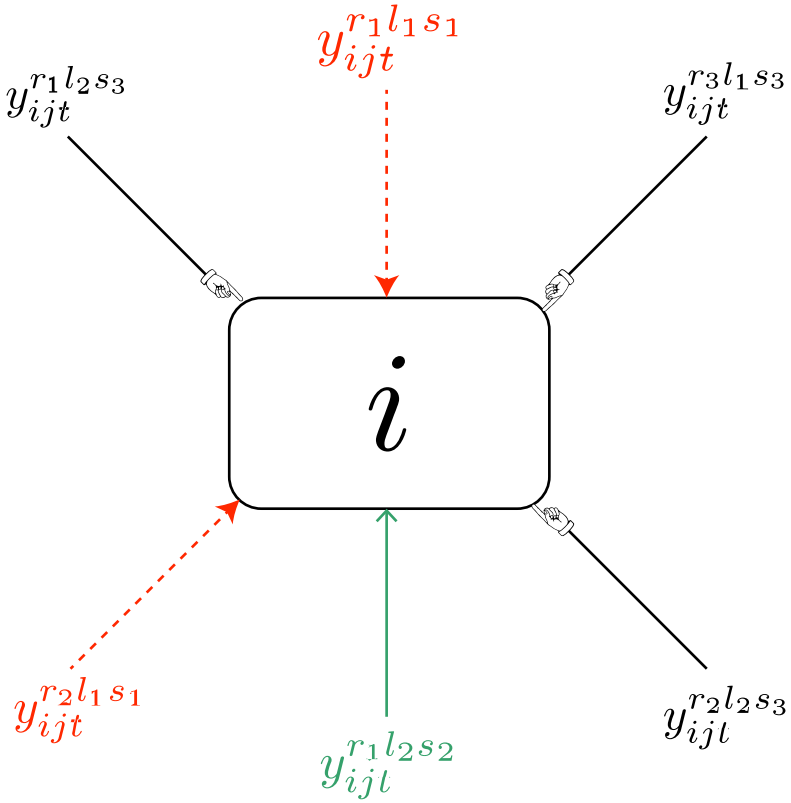

図 4.5　y_{ijt}^{rls} は船 $s \in S$ が航路 $r \in \mathcal{R}$ の $l(=1,\cdots,l_r)$ 周目で運んでいる割合ではなく, 港 $i \in U$ の期間 $t \in \mathbb{N}$ における $j \in \mathbb{N}$ 番目の time-window で降ろされる貨物量の割合を表す

通過するとき 1, それ以外 0 となる定数 $\alpha_{ijt}^{rl} \in \{0,1\}$ を導入する. また, $i \in U$ の期間 $t(=1,\cdots,t_{\text{last}})$ における $j(=1,\cdots,j_{\text{last}}^i)$ 番目の time-window において, 船が $r \in \mathcal{R}$ の $l(=1,\cdots,l_r)$ 周目を通ってこの time-window で降ろす貨物量の割合を, を $y_{ijt}^{rls} \in [0,1]$ という変数を導入する. これは船が航路を通して降ろす割合を表しているのではなく, 港に設定された time-window 上を通る船全ての降ろす貨物量の割合を 1 として, その中でも船 $s \in S$ が降ろす貨物量の割合であることに注意する. すなわち, 各港 $i \in U$ の期間 $t(=1,\cdots,t_{\text{last}})$ における $j(=1,\cdots,j_{\text{last}}^i)$ 番目の time-window において,

$$\sum_{r\in\mathcal{R}}\sum_{l=1}^{l_r}\sum_{s\in S} y_{ijt}^{rls} = 1$$

である. すると, 船 $s \in S$ が港 $i \in U$ の期間 $t(=1,\cdots,t_{\text{last}})$ における港 $i \in U$ の $j(=1,\cdots,j_{\text{last}}^i)$ 番目の time-window 中に航路 $r \in \mathcal{R}$ の $l(=1,\cdots,\text{last})$ 週目を通ってやってきて, そこで $s \in S$ が降ろす貨物量の割合は $\alpha_{ijt}^{rl} y_{ijt}^{rls}$ と表すことができる. これら

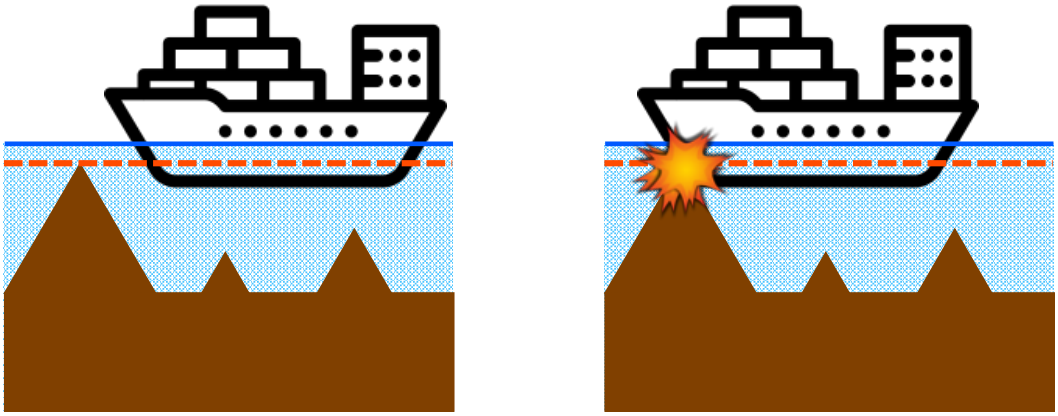

図 4.6　喫水が海面から海底までの深さよりも深いと船は座礁してしまう (点線が海面から海底までの最小の高さ)

から貨物需要を守ろうとする制約は

$$\sum_{r\in\mathcal{R}}\sum_{s\in S}\sum_{l=1}^{l_r}\alpha_{ijt}^{rl}y_{ijt}^{rls}=1\ \ (i\in U;j=1,\cdots,j_{\mathrm{last}}^{i};t=1,\cdots,t_{\mathrm{last}})$$

と書き表すことができる．また, 船には積載量 $Q_s\in\mathbb{R}_{>0}[\mathrm{t}]$ があるので, 航路を移動している間はその積載量を守るようにしたい．船 $s\in S$ が航路 $r\in\mathcal{R}$ の $l(=1,\cdots,t_{\mathrm{last}})$ 週目で降ろす貨物量は $e_{ijt}\alpha_{ijt}^{rl}y_{ijt}^{rls}$ と表すことができるので, このような制約は

$$\sum_{t=1}^{t_{\mathrm{last}}}\sum_{i\in U}\sum_{j=1}^{j_{\mathrm{last}}^{i}}e_{ijt}\alpha_{ijt}^{rl}y_{ijt}^{rls}\le Q_s x_{rls}\ \ (r\in\mathcal{R};s\in S;l=1,\cdots,l_r)$$

と書き表すことができる．最後に船 $s\in S$ には喫水 $d_s\in\mathbb{R}_{>0}[\mathrm{m}]$ が設定されている．そこで, 港や船によっては喫水が深すぎることによって港に入ることができないといった状況が発生することがある．したがって, 喫水を守ることは重要である．そこで, 船の喫水を守るように

$$d_s x_{rls}\le h_r\ \ (r\in\mathcal{R};s\in S;l=1,\cdots,l_r)$$

という制約を入れる．

以上から，表 4.3 と 表 4.4 の記号を用いて, 制約

表 4.3　船・資源の割り当てを行う問題で使用する定数

記号	意味
S	船の集合
$\mathcal{R}$	航路割当問題 (RAP) で割り当てられた航路の集合
U	荷物を降ろす港の集合
$\alpha_{ijt}^{rl} \in \{0,1\}$	港 $i \in U$ における期間 $t \in \{1,\cdots,t_{\text{last}}\}$ の $j \in \{1,\cdots,j_{\text{last}}^i\}$ 番目の time-window を航路 $r \in R$ の $l \in \{1,\cdots,l_r\}$ 周目が通るとき 1, それ以外 0
$Q_s \in \mathbb{R}_{>0}$	船 $s \in S$ の積載量 [t]
$p_s \in \mathbb{R}_{>0}$	船 $s \in S$ の使用にかかるコスト
$d_s \in \mathbb{R}_{>0}$	船 $s \in S$ の喫水 [m]
$h_r \in \mathbb{R}_{>0}$	航路 $r \in \mathcal{R}$ の深さ [m]

表 4.4　船・資源の割り当てを行う問題で使用する変数

記号	意味
$x_{rls} \in \{0,1\}$	船 $s \in S$ が航路 $r \in R$ の $l \in \{1,\cdots,l_r\}$ 周目を通るとき 1, それ以外 0
$y_{ijt}^{rls} \in [0,1]$	港 $i \in U$ の期間 $t \in \{1,\cdots t_{\text{last}}\}$ 中の $j \in \{1,\cdots,j_{\text{last}}^i\}$ 番目の time-window において船 $s \in S$ が航路 $r \in \mathcal{R}$ を通って港 $i \in U$ へやってきたときに船 $s \in S$ が降ろす貨物量の割合 $\left(\sum_{r\in\mathcal{R}}\sum_{l=1}^{l_r}\sum_{s\in S} y_{ijt}^{rls} = 1\right)$

(1) 航路の周期性

$$\sum_{l=1}^{l_r} x_{rls} = l_r x_{r1s} \ \ (s \in S; r \in \mathcal{R})$$

(2) 航路は全て使用

$$\sum_{s\in S} x_{rls} = 1 \ \ (r \in \mathcal{R}; l = 1,\cdots,l_r)$$

(3) 船への航路割り当て数制限

$$\sum_{r\in\mathcal{R}}\sum_{l=1}^{l_r} \frac{x_{rls}}{l_r} \leq 1 \ \ (s \in S)$$

(4) 貨物需要は守る

$$\sum_{r\in\mathcal{R}}\sum_{s\in S}\sum_{l=1}^{l_r}\alpha_{ijt}^{rl}y_{ijt}^{rls}=1\ \ (i\in U;j=1,\cdots,j_{\text{last}}^{i};t=1,\cdots,t_{\text{last}})$$

(5) 積載量制限

$$\sum_{t=1}^{t_{\text{last}}}\sum_{i\in U}\sum_{j=1}^{j_{\text{last}}^{i}}e_{ijt}\alpha_{ijt}^{rl}y_{ijt}^{rls}\le Q_s x_{rls}\ \ (r\in\mathcal{R};s\in S;l=1,\cdots,l_r)$$

(6) 喫水制限

$$d_s x_{rls}\le h_r\ \ (r\in\mathcal{R};s\in S;l=1,\cdots,l_r)$$

を守りつつ, 全体のコスト

$$\sum_{r\in\mathcal{R}}\sum_{s\in S}\sum_{l=1}^{l_r}p_s\frac{x_{rls}}{l_r}$$

を最小にするような問題は

$$(\text{SAP})=\begin{cases}\min & \displaystyle\sum_{r\in\mathcal{R}}\sum_{s\in S}\sum_{l=1}^{l_r}p_s\frac{x_{rls}}{l_r}\\ \text{s.t.} & \displaystyle\sum_{l=1}^{l_r}x_{rls}=l_r x_{r1s}\ \ (s\in S;r\in\mathcal{R})\\ & \displaystyle\sum_{s\in S}x_{rls}=1\ \ (r\in\mathcal{R};l=1,\cdots,l_r)\\ & \displaystyle\sum_{r\in\mathcal{R}}\sum_{l=1}^{l_r}\frac{x_{rls}}{l_r}\le 1\ \ (s\in S)\\ & \displaystyle\sum_{r\in\mathcal{R}}\sum_{s\in S}\sum_{l=1}^{l_r}\alpha_{ijt}^{rl}y_{ijt}^{rls}=1\ \ (i\in U;j=1,\cdots,j_{\text{last}}^{i};t=1,\cdots,t_{\text{last}})\\ & \displaystyle\sum_{t=1}^{t_{\text{last}}}\sum_{i\in U}\sum_{j=1}^{j_{\text{last}}^{i}}e_{ijt}\alpha_{ijt}^{rl}y_{ijt}^{rls}\le Q_s x_{rls}\ \ (r\in\mathcal{R};s\in S;l=1,\cdots,l_r)\\ & d_s x_{rls}\le h_r\ \ (r\in\mathcal{R};s\in S;l=1,\cdots,l_r)\\ & x_{rls}\in\{0,1\}\ \ (r\in\mathcal{R};l=1,\cdots,l_r;s\in S)\\ & 0\le y_{ijt}^{rls}\le 1\\ & \quad (r\in\mathcal{R};l=1,\cdots,l_r;s\in S;i\in U;t=1,\cdots,t_{\text{last}};j=1,\cdots,j_{\text{last}})\end{cases}$$

となる.

表 5.1　実験環境

OS	CPU	メモリ	Gurobi[7] のバージョン
Ubuntu 18.10	Intel(R) Xeon(R) CPU E5-2450 0 @ 2.10GHz	128 GB	8.1.0

5　数値実験

本章では, 提案モデルの (RAP) および (SAP) に対して行った数値実験とその結果について記す.

まず, (RAP) と (SAP) に対して, 入力する航路数を増やした時に, 各モデルにどの程度計算時間として影響が出るのかを測定している. そののちに, (RAP) に対して, 航路が正しく全ての期間内の time-window を通過するように割り当てているのかを確認し, 問題設定に対する有効性を確認する, さらに, (RAP) に対して, 区切る期間の個数を変化させたり, 航路数・作成方法などを変えたときの目的関数値の変動と, 各指定された期間の数を通るような航路の中で割り当てられる航路において, どれくらいの期間を通る航路が割り当てられやすいのかを見ていく.

なお, 本章における実験では, 全て 表 5.1 の通りの環境を使用している. また, 航路のコストを計算する問題 (3.2) において, コストを計算する関数は一貫して Fagerholt ら [4] の論文で提示されている (3.1) を使用している.

5.1　インスタンスの生成方法

インスタンスの生成方法は次のものを利用する:

short　モデルに与える港の数は 10 個 とし, 各港には time-window を一様ランダムに 1 個 から 3 個 用意し, タイムテーブル全体の長さとしては 168 h(1 週間) を 1 期間として 5 期間, つまり, 840 h とする. 航路は通る期間が 1 期だけのもの, 2 期分通るものの 2 種類を得るように, 一様ランダムに通過したい time-window の個数を 2 個 から 6 個 選択して生成し, 各港間の距離は 50 M から 500 M に設定し, 港におけるサービスにかかる時間は一様ランダムに 8 h から 10 h の間となるように生成する. 各港における貨物需要量は 10 t から 50 t で一様乱数で生成し, 船の容量は 200 t から 400 t で一様乱数生成している. 船にかかるコストは 1000 から 4000 で

一様ランダムに生成し, 喫水は 14 m から 17 m で一様乱数生成し, 海面から海底までの深さは喫水と被るように 15 m から 20 m で一様ランダムに生成している.

long　用意する港数は 20 個 とし, 各港には time-window を 4 個 設定する. また, 港間の距離は 50 M から 500 M とし, 期間は 1 期間を 1 週間として 20 期間 用意する. 各港においてサービスにかかる時間は 8 h から 10 h に設定している. 航路はこのうち 1 期間だけを通るものから 1 期間 から指定した期間まで (最大 10 期間) 跨ぐものをそれぞれ 200 個 ずつ用意する.

cluster　一様乱数で距離が 50 M から 150 M となるような港の組を 4 個 用意する. 港はそれぞれの組の中に 5 個ずつ用意し, 別の組の港同士の距離は一様乱数で 300 M から 500 M となるようにする. タイムテーブルの長さは 168 h の長さの期間を 12 個用意し, 航路は 1 期間 から 6 期間 跨ぐように 1000 個 ずつ一様乱数生成するが, 指定された割合だけ, 各クラスタ内の港しか通らないものを用意する. 例えば, 2 割 だけクラスタ内の航路を通るようにする時のクラスタ毎に用意するクラスタ内だけを移動するような航路の個数は $\frac{1000}{4} \times \frac{2}{10} = 50$個 となる. また, サービス時間は 8 h から 10 h とする.

5.2　航路割当の有効性

まず, (RAP) が正しく航路を割り当てていることを確認する. 本実験ではインスタンスの生成方法として short を使用する.

実際に得られた実験結果は 図 5.1 の通りである. 縦軸のうち, 正整数 (1 から 10) は港を表している. 横軸は時間を表し, 単位は h である. グラフ中に表れる縦線は期の境目を粗している. 各港の横に並んでいる線は各港の time-window を表し, この線を通っている線は割り当てられた航路である.

図を見ると, 周期的な形をした線のみが表れていることがわかる. また, どの航路も通っていない time-window は存在していないことがわかる. 以上から, (RAP) は正しく航路割当を行えていることが確認できる.

5.3　提案モデルの計算時間

本節では (RAP) と (SAP) に対して, 入力として与える航路数を変動させたときのモデルの計算時間を測定する. インスタンスは short を利用し, 同じ航路数でそれぞれ 30 個の問題を作って計算時間を測定して, 平均をとった. なお, (RAP) を解いた結果を (SAP)

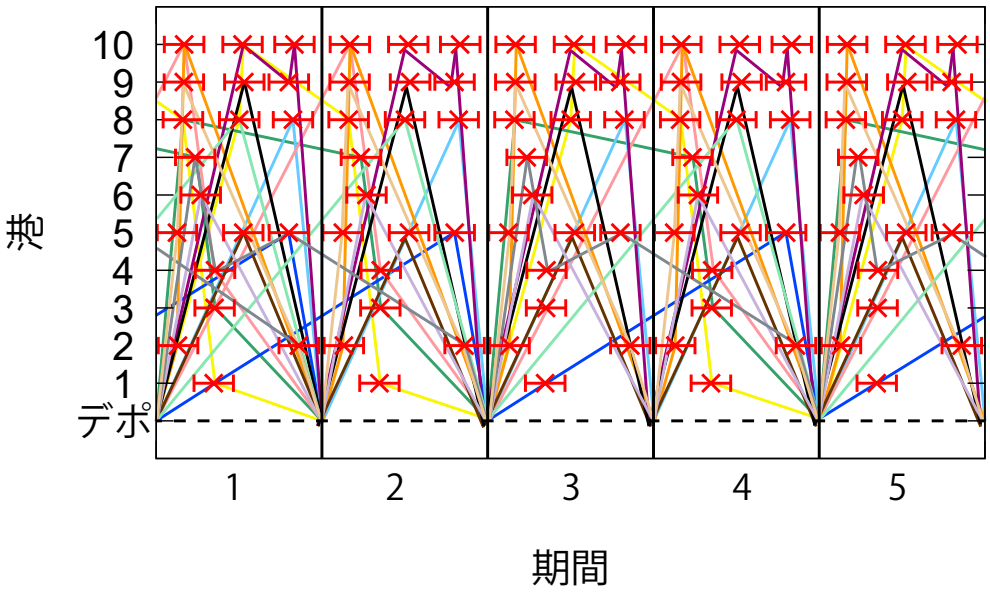

図 5.1　(RAP) が割り当てた航路の図

への入力として利用した.

測定結果として, 図 5.2 を得た. 縦軸は平均計算時間を, 横軸は入力した航路数を表している. また, (1) は (RAP) の計算時間を, (2) は (RAP) の計算時間をそれぞれ表している.

図を見ると, (RAP) は航路数の増加とともに計算時間が上っている. また, 航路数を増やしても (SAP) と (RAP) の計算時間を併せた全体の実行時間は大きく変化していない. これは (RAP) の計算時間のかかりかたの変化がそこまで大きくないことと, (SAP) の計算時間が変化していないからではないかと考えられる. 実際, (SAP) の計算時間は 図 5.3 のようにほとんど変化していない.

5.4　(RAP)

5.4.1　航路の入力数に対する性質

本実験では航路を入力する数を変化させると (RAP) の結果がどのように変化するかを見る. 入力するインスタンスとして short を利用し, 航路は先に 4000 個 用意しておき, それらのなかからランダムに指定個数選んで選んだものを入力として実験するということを 200 個 から 4000 個 まで 200 個 ずつ大きくしながら行った. この実験を 30 回行ってそれぞれの平均を見た.

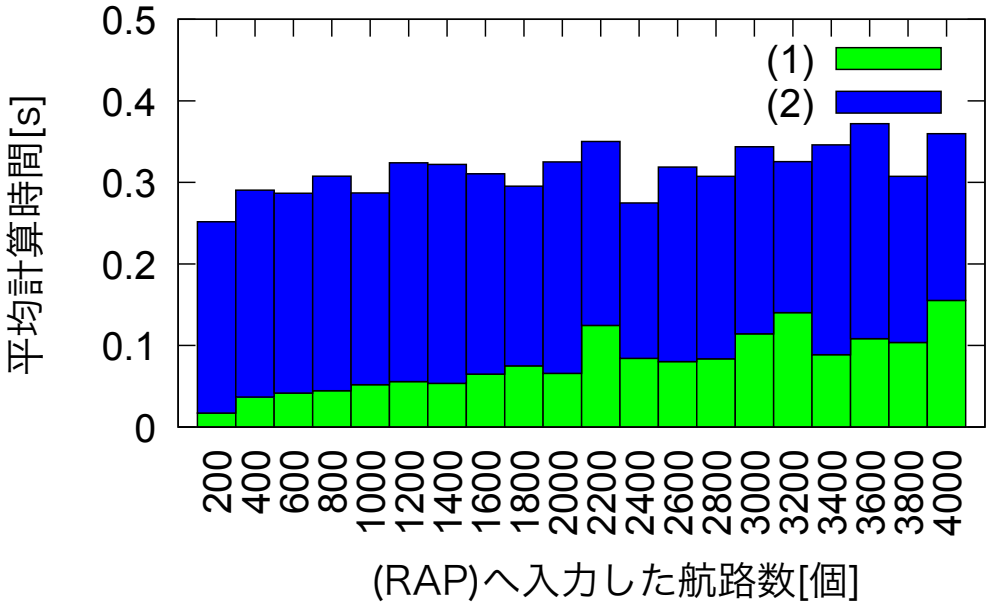

図 5.2　平均計算時間 (単位は秒). (1) は (RAP) の計算時間を, (2) は (SAP) の計算時間を表している.

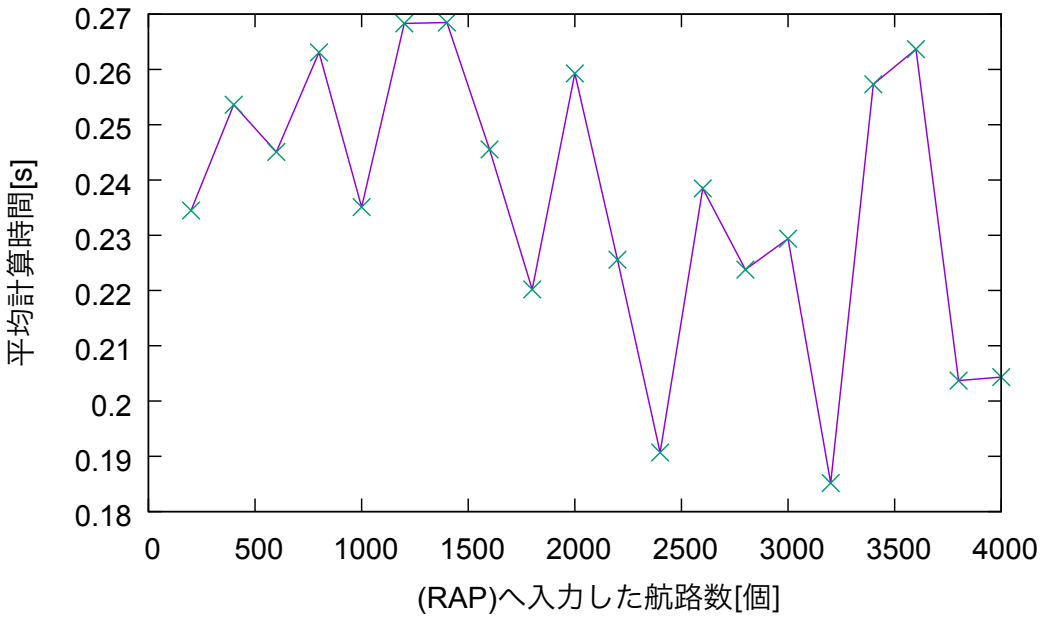

図 5.3　(SAP) の計算時間. 縦軸が平均計算時間 (単位は秒), 横軸が (RAP) へ入力した航路数を表す. × がそれぞれの航路数に対して得られた計算間を表している.

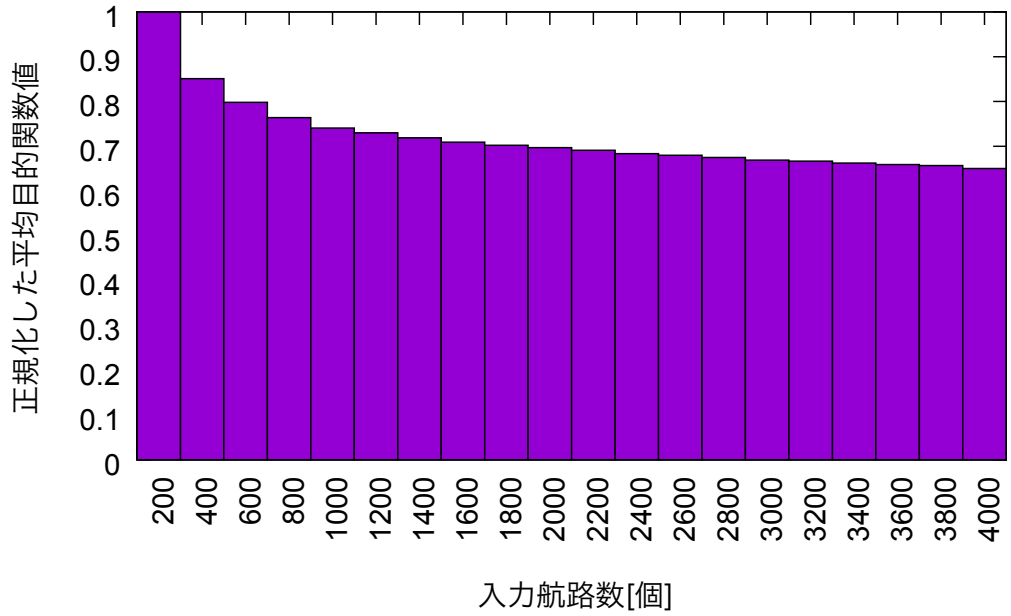

図 5.4　入力航路数を増加させたときの (RAP) の目的関数値

まず, 目的関数値を見る. ここでは, それぞれの目的関数値は大きく異なるので, 各実験における最大値で割って正規化している. 得られた結果は 図 5.4 の通りである. 縦軸は正規化した平均の目的関数値を, 横軸は入力した航路数を表している. 図を見ると入力する航路数が多くなると, 目的関数値が改善されていくのがわかる. 特に, 200 個 から 400 個 入力するあたりで大きく改善され, それ以降は比較的ゆっくりと改善されている様子を見ることができる.

次に, 航路が各入力に対してどの程度の数割り当てられているのかを見る. 得られた結果は 図 5.5 の通りである. 縦軸は割り当てられた平均航路数を表し, 横軸は入力した航路数を示す. ここから航路数が多くなると割り当てられる航路数はより少なく済むということがわかる.

以上から, より候補が多くなると, 目的関数値, 選ばれる航路数がともに減少することがわかった. これは, より多くの候補があることで, より少数で, コストが少ないような航路割り当てを行えることを示していると考えられる. 先程の計算時間の結果と合わせると, 計算時間は大きく増えることはないので, より航路をたくさん用意しても問題はなく, よりコストを抑えるような航路割当を行うことはできるが, 大きく目的関数値を下げることはできないため, 航路を生成する手間などを勘案して, 問題に合った航路数を選択することが提案モデルに重要となる.

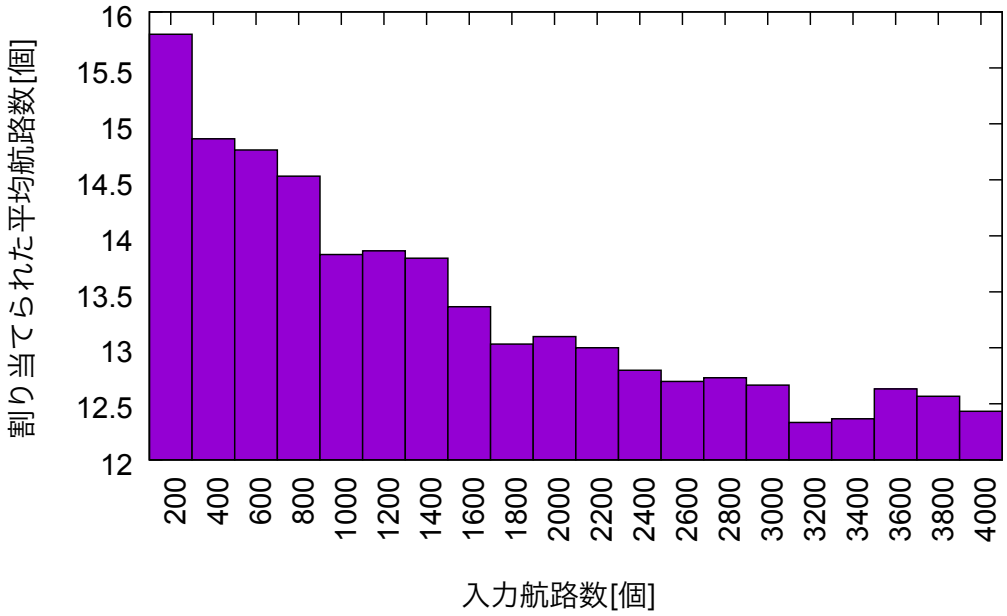

図 5.5　入力航路数を増加させたときの (RAP) の航路数

5.4.2　航路が跨っている期間数に対する性質

本実験では．航路が通る期間の数 (例えば航路が 3 つの期間を通るなら 1 期目・2 期目・3 期目を通り, 航路が 2 つの期間を通るなら 1 期目・2 期目を通る) を変更した時の, (RAP) の目的関数値やどのような航路が割り当てられるのかを見る．入力するインスタンスは long を利用している．実験は 20 回行って, それぞれの値は平均をとっている．

まず, 目的関数値を見る．なお, 入力として, まず航路を 1 期間 から 10 期間 まで跨いでるもの全て生成してから, 指定した期間分跨いでいる航路, たとえば, 2 期間 までの航路を入力とするように指定されたら, 1 期間 だけ跨いでいる航路と 2 期間 跨いでいる航路を入力とするようにしている．また, 目的関数値は問題毎に大きく異るので, 最大値で割って正規化している．得られた結果は 図 5.6 の通り．縦軸は正規化した平均目的関数値を, 横軸は入力に使用した, 航路が跨って良い最大の期間の数を表している．つまり, 最大の期間数として 3 が指定されていれば, 1 期間 だけ跨っている航路, 2 期間 跨っている航路, 3 期間 跨っている航路をまとめて入力している．結果を見ると, 2 期間跨ぐような航路を入力した時に大きく目的関数値が下がり, 3 期間 跨ぐような航路を入力するようになると, 小さく目的関数値が下がり, それ以降は変化していないことがわかる．

次に, 航路の分布を見る．これは生成した航路を一度に全て入力として実験しているこ

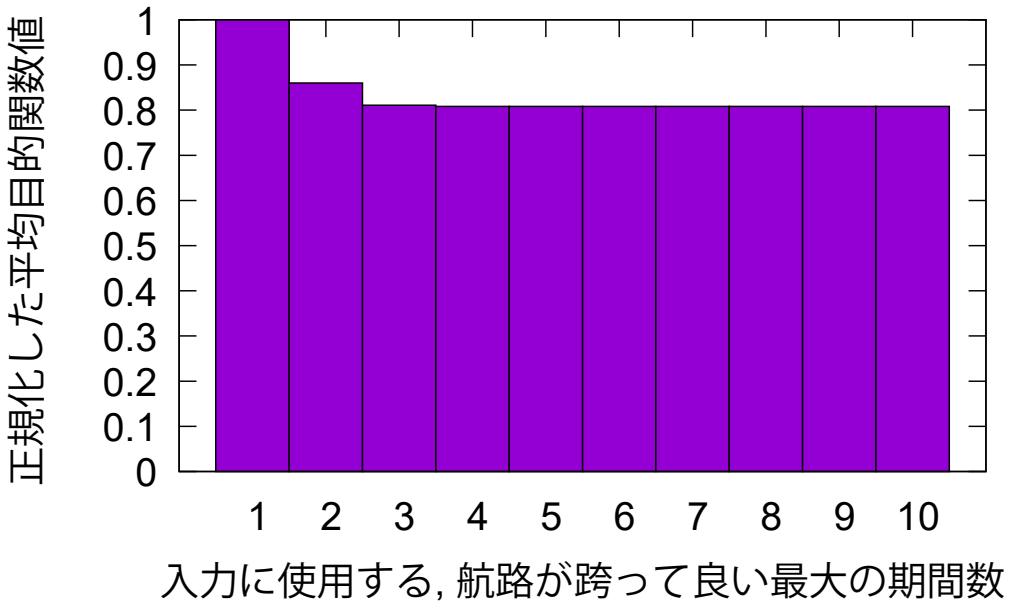

図 5.6　指定された期間を跨いでいる航路全てを入力したときの (RAP) の目的関数値

とに注意する．得られた結果は 図 5.7 の通り．縦軸は割り当てられた平均航路数を, 横軸は航路が跨っている期間数を表している．したがって, どのくらいの期間数を跨ぐような航路が, どのくらい割り当てられているのかを表している．

1 期間 だけ跨ぐような航路だけでなく, 2 期間 まで跨ぐような航路も選択されているが, 3 期間 以上跨ぐような航路はほぼ選択されていないということがわかる．

以上から, 1 期間だけの情報から航路を構成するだけでなく, 2 期間目の情報も活用すれば, よりコストの抑えられる航路割当を行えることがわかった．3 期間 以上またぐような航路がほとんど割り当てられていないのは, 他の航路を割り当てようとしたときに, 同じ time-window を通ろうとしてしまうからではないかと考えられる．つまり, (RAP) としてはなるべく 1 つの time-window は 1 つの航路だけが通った方が望ましいが, 航路が長くなると他のより短い航路との兼ね合いで, time-window を共有しようとしてしまうために, 長い航路は割り当て辛くなってしまっている．

5.4.3　部分問題を解いた結果と併せたときの性質

本実験では指定された期間数未満を跨っている航路を入力として (RAP) を解いた後, その結果を利用して指定した期間数を跨っている航路と併せて入力するものと指定された期間以下の航路全て入力としたときの比較を行う．これによって, 部分問題の結果を利用

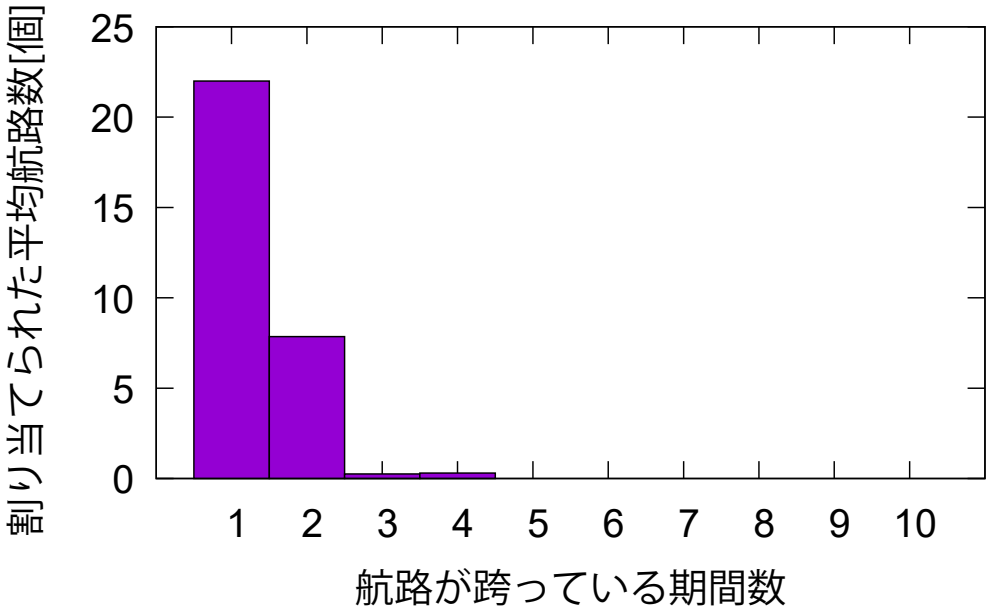

図 5.7　long で生成した航路を全て入力した時の (RAP) の航路の分布

したものとそうでないものの比較を行うことを狙う．インスタンスの生成方法としては, long を利用し, 実験は 30 回行い, それぞれの平均値を見た．

まず, 目的関数値を示す．得られた結果は 図 5.8 なお, 目的関数値は問題毎に大きさが異なるので, 最大値で割ったものを正規化している．縦軸が正規化した平均目的関数値で, 横軸が航路が跨っている最大の期間数を表している．また, (1) が指定した期間数で解いた結果を利用したもの, (2) が指定した期間数以下を跨いでいる航路を全て入力した結果である．

2 期間 跨ぐようなものが入力されるとどちらも目的関数値が小さくなるが, 指定した最大期間数以下の航路を全て入力する方がより良い結果が得られていることが確認できる．

次に, 航路の割り当て分布を見る．得られた結果は 図 5.9 の通り．縦軸は割り当てられた平均航路数, 横軸は航路が跨っている期間の数をそれぞれ表している．(1) は指定された期間数よりも小さい期間跨いでいる航路を入力して問題を解いた結果を利用したときの結果を表し, (2) は指定された期間数以下を跨いでいる航路全てを入力した結果である．

結果から, 指定された期間数よりも小さい問題を解いたものは全てを入力したものよりも 1 期間だけ跨いでいる航路の割当数が多く, 2 期間以上跨ぐような航路は全て入力するものよりも割り当てられにくいことがわかる．

従って, 指定された期間数よりも小さい航路を入力として問題を解いたものは全てを入

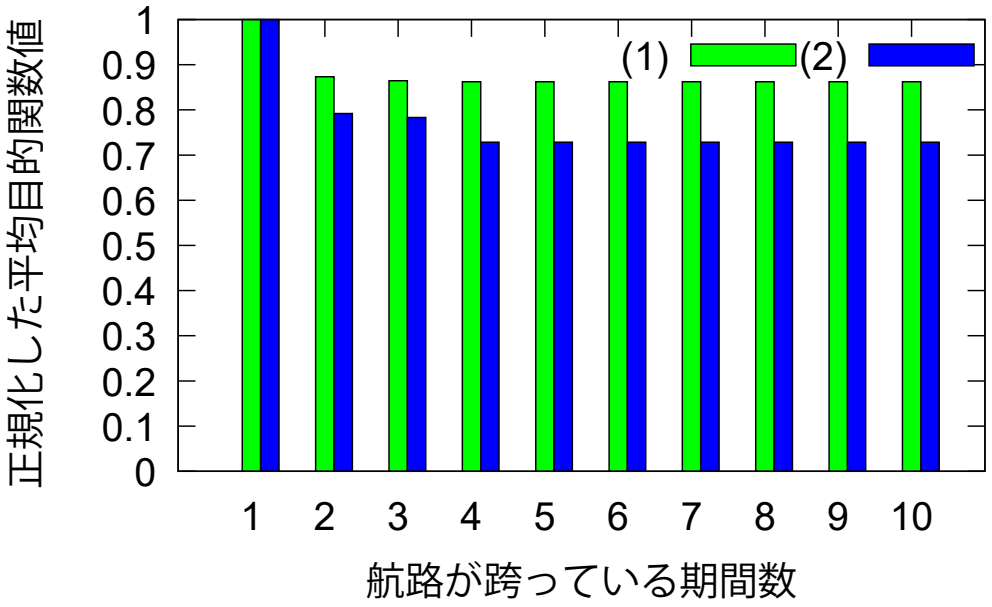

図 5.8　指定された期間数よりも小さい期間跨いでいる航路を入力として問題を解いた結果を利用したときの (RAP) の目的関数値の比較結果. (1) は問題を解いた結果を利用したもので, (2) は指定期間数以下跨いでいる航路を全てをそのまま入力した場合

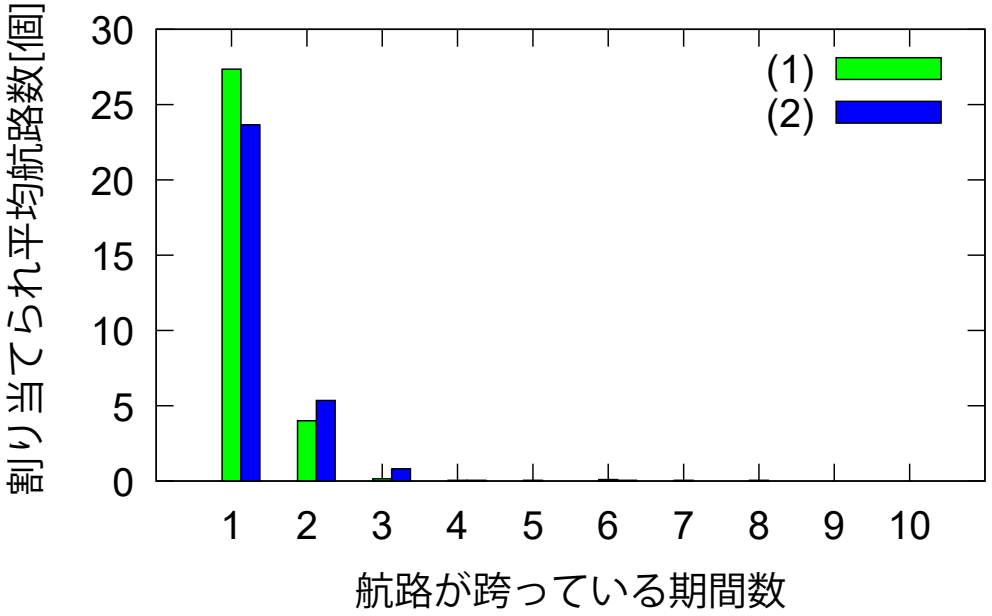

図 5.9　指定された期間数よりも小さい航路を入力とする問題を解いた結果を利用したときの (RAP) の航路数の分布の比較結果. (1) は問題を解いた結果を利用したもので, (2) は指定期間数以下跨いでいる航路全てをそのまま入力したもの

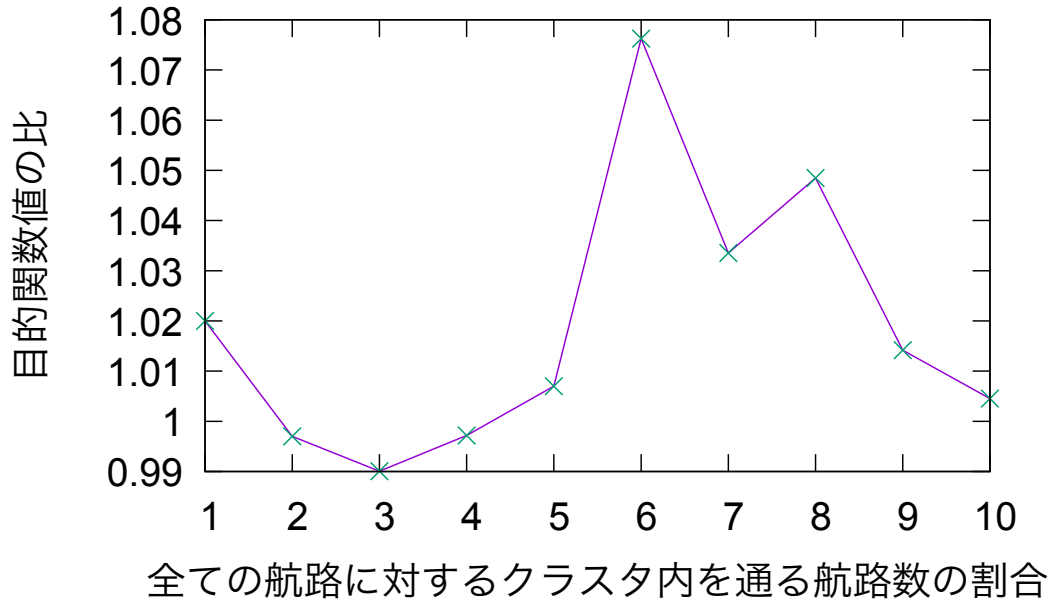

図 5.10　クラスタを通るように航路を生成したものとそうでないものとの目的関数値の比

力したものよりも悪くなることがわかる．これは, 選択肢が少なくなることで, 新しく入力に加えられるものを, time-window を複数の航路が共有しないように割り当てることが, 全ての航路を入力して解くものよりも難しくなり, 既に得られた結果をそのまま利用した方が time-window を複数航路が通らないようになるからではないかと考えられる．

5.4.4　近いものでクラスタを構成したときの性質

本実験では近いものでクラスタを構築し, 一定の割合だけ, そのクラスタ内を通るような航路を生成したときの目的関数値がどのようになっているかを見る．インスタンスの生成方法としては cluster を利用し, クラスタ内を通るような航路の割合を 10 % から 100 % まで 10 % ずつ上げて変化させる．目的関数値はクラスタ内を通るような航路を生成する割合を 0 % に設定した目的関数値で割ってその変動を見ている．

得られた目的関数値は 図 5.10 のようになった．縦軸は目的関数値の比を, 横軸はクラスタ内を通る航路数の割合を表している．

目的関数値はほとんど変わらないことがわかるため, なるべく小さなルートを選んでルートを作るようなことをせずとも, ランダムに作る, つまり, 長いルートが紛れこんでしまうようにルートを作って問題ないことを指していると考えられる．

6　まとめと今後の課題

6.1　まとめ

本論文では定期船に対し, 周期的な time-window が設定を元にした周期的な航路の割当問題 (RAP) とその航路に対する船舶割当問題 (SAP) を提案・定式化した. 定式化するにあたっては, まず, 港へ着いて欲しい期間を表す time-window を記したタイムテーブルというものを導入した. そのタイムテーブルを用いて, 出発港からそのタイムテーブル上の time-window をいくつか結んで同じ出発港へ帰るようなものを航路とし, その航路が複数回周回するようにして定期船への航路割当を行う最適化問題を提案した. その割り当てた航路を入力として, 喫水や貨物への積載量制限, 港の需要量を守る制限, 割り当てられた航路は全て使用する制限などを考慮した船舶割当を行う最適化問題を提案した.

これらの提案した最適化問題に対し, 数値実験を通して, 様々な性質を見た. まず, (RAP) と (SAP) が現実的な時間で解けることを示し, (RAP) への入力航路数を増やしても大きくかかる時間の増加量が増えないことを示した. また, (RAP) が与えられたタイムテーブルに対して, 全ての time-window を通るような有効な定期航路の割当を行っていることを確認した. (RAP) においてはさらに,

- 入力航路数を増やすと, 割り当てられる航路の総数は, 徐々に少なくなり, それに伴って目的関数値も徐々に減少していくこと
- タイムテーブルを区切っている期間のうち 1 期間だけ通っているような航路だけでなく, より多くの区切られている期間を通っている方が目的関数値が減少すること
- 指定された期間数未満を通っている航路を入力として部分問題を解いた結果を指定された期間数通っている航路と併せて入力しても目的関数値はそこまで改善されないこと
- 近い港どうしでクラスタを構築して, そのクラスタ内を通るような航路を混ぜて航路割当問題を解いても, 目的関数値はほとんど変わらないこと

を確かめた.

以上から, 定期航路を割り当てる際には 1 期間だけの情報を利用するのではなく, 複数期間の情報を利用し, さらに入力する航路数を多くした方が, より良い航路割当を行えることがわかった.

6.2 今後の課題

今回提案したモデルにおいて, 港の貨物需要に対して, ロバスト化を行うことが考えられる. 提案モデルでは各港の time-window における貨物需要はデータとして与えられているが, 一般にどこの港へどれくらいの貨物を運んで欲しいかはわからない. 例えば, フェリーなどで船を運営している側は確実にこの人数が利用する, ということを知ることはできない. 従って, 提案モデルの積載量制約をロバストなものとする必要があると考えられる.

また, 実データを利用して, 実際にモデルを解き, このモデルが現実問題を解くときにはどのようなところが弱いのかを確認したり, 大規模なものに対しても解けるのかなどを確認していくことも今後の課題としたい.

参考文献

[1] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, Mar. 8, 2004, p. 138 (cit. on p. 18).

[2] Jens Clausen, Allan Larsen, Jesper Larsen, and Natalia J. Rezanova. "Disruption management in the airline industry - Concepts, models nad method". In: *Computes & Operations Research* 37 (2010), pp. 809–821.

[3] George Bernard Dantzig. "Programming in Linear Structure". In: *Comptroller* (1948) (cit. on p. 19).

[4] K Fagerholt, G Laporte, and I Norstat. "Reducing fuel emissions by optimizing speed on shipping routes". In: *Journal of the Operational Research Society* 61 (2010), pp. 523–529 (cit. on pp. 23, 40).

[5] Julie Sand Fuglum and Marie Ameln. *The Linear Shipping Network Design Problem - Strengthened formulations consideering complex route structures, transhipment and transit time*. 2015. URL: https://brage.bibsys.no/xmlui/handle/11250/2352993.

[6] Ricardo Fukasawa, Qie He, Fernando Santos, and Yongjia Song. *A Joint Routing and Speed Optimization Problem*. Optimization Online. Mar. 5, 2017. URL: http://www.optimization-online.org/DB_FILE/2016/02/5344.pdf (cit. on pp. 23, 32).

[7] *Gurobi Optimizer 8.1*. Gurobi Optimization, LLC. URL: https://www.gurobi.com/index (cit. on p. 40).

[8] Qie He, Xiaochen Zhang, and Kameng Nip. "Speed optimization over a path with heterogeneous arc costs". In: *Transportation Research Part B: Methodological* 104 (2017), pp. 198–214.

[9] Ibrahim El-Henawy and Negham Ahmed Abdelmegeed. "Meta-Heuristics Algorithms: A Survey". In: *International Journal of Computer Applications* 179.22 (Feb. 2018) (cit. on p. 21).

[10] *International trade in goods by mode of transport*. eurostat statistics explained. URL: https://ec.europa.eu/eurostat/statistics-explained/index.php/International_trade_in_goods_by_mode_of_transport (cit. on p. 3).

[11] Narendra Karmarkar. "A new polynomial-time algorithm for linear programming". In: *Combinatorica* 4 (4 1984), pp. 373–395 (cit. on p. 19).

[12] Leonid G. Khachiyan. "Polynomial algorithms in linear programming". In: *USSR Computational Mathematics and Mathematical Physics* 20 (1 1980), pp. 53–72 (cit. on p. 19).

[13] *RORO* 船. 日本通運. URL: https://www.nittsu.co.jp/support/words/pqrs/roll-on-roll-off-ship.html (cit. on p. 3).

[14] Paolo Toth and Daniele Vigo, eds. *Vehicle Routing Problems, Methods, and Applications*. Society for Industrial and Applied Mathematic, Dec. 5, 2014, pp. 381–408.

[15] Chengliang Zhang, George Nemhauser, and Joel Sokol. "Flexible solutions to maritime inventory routing problems with delivery time windows". In: *Computers & Operations Research* 89 (2018), pp. 153–162.

[16] 久保幹雄; J. P. ペドロソ; 村松政和; A. レイス. **あたらしい数理最適化 - *Python* 言語と *Gurobi* で解く -**. 近代科学社, Nov. 30, 2012, p. 177 (cit. on p. 22).

[17] 迫 昭彦. **定期船事業と市場の仮象性**. 日本海事センター 企画研究部. URL: www.jpmac.or.jp/research/pdf/496.pdf (cit. on p. 3).

[18] 田村明久; 村松正和. **最適化法**. 共立出版株式会社, Apr. 1, 2013.

[19] 久保幹雄; 田村明久; 松井知己. **応用数理計画ハンドブック**. 朝倉書店, Mar. 20, 2005, p. 240 (cit. on p. 21).

[20] 室田 一雄; 杉原 正顯. **線形代数 *I***. Ed. by 東京大学工学教程編纂委員会. 丸善出版, Sept. 30, 2015, pp. 182–183 (cit. on p. 5).

[21] 小島政和; 水野真治; 土谷隆; 矢部博. **内点法**. 朝倉書店, Aug. 1, 2001, pp. 203–204 (cit. on p. 20).

[22] 福島雅夫. **非線形最適化の基礎**. 朝倉書店, May 25, 2012.

[23] 寒野 善博; 土谷 隆. **最適化と変分法**. Ed. by 東京大学工学教程編纂委員会. 丸善出版, Oct. 20, 2014.