数理解析研究所講究録 1349 巻 2004 年 94-112

移動時間コスト関数を考慮した 時間枠つき配送計画問題に対する局所探索法

京都大学大学院情報学研究科数理工学専攻
橋本英樹 (Hideki Hashimoto)
今堀慎治 (Shinji Imahori)
柳浦睦憲 (Mutsunori Yagiura)
茨木俊秀 (Toshihide Ibaraki)
Dept. of Applied Mathematics and Physics
Graduate School of Informatics
Kyoto University

1 序論

配送計画問題 (vehicle routing problem, VRP) は, 様々な制約条件の下で, 複数の車両を用いて全ての客をちょうど一回ずつ訪問するような経路集合の中で, コストが最小のものを求める問題である. これは, 代表的な組合せ最適化問題の一つであると同時に, 非常に実用性のある問題で, 郵便・新聞配達, 廃棄物収集, 石油運搬やスクールバスのスケジューリングなどの応用を持つ. その実用性の高さが認識され研究が始まったのは 1950 年代であるが, 特にここ 15 年ほどの研究は活発である [1, 2, 7]. この問題は NP 困難であることが知られており, 大規模な問題例に対して厳密な最適解を求めることは現実的に極めて困難であると考えられている. そのような問題に対する現実的妥協策として近似解法があり, 配送計画問題に対しても種々の近似解法が提示されている.

本研究で扱う時間枠つき配送計画問題は, 制約条件として各車両の容量制約と時間枠制約を取り扱う. 容量制約とは, 一台の車両が訪問する客の要求量の総和が, 一台の車両で処理出来る量を越えないというものである. 時間枠制約とは, 客が指定した時間枠内にサービスを開始しなければならないというものである. いずれの制約も, 容量制約のみを課した問題は分割問題を, 時間枠制約のみを課した問題はスケジューリング問題を特殊な場合として含むことから, 一方の制約のみを課した問題に対する実行可能解の存在の判定がすでに NP 完全である [3]. そのため, 探索を実行可能解に限定してしまう手法 [5] では, 制約が厳しい場合には探索が効率よく行えないと考えられる. この点を克服する方法の 1 つとして, これらの制約を考慮制約として扱い, 制約の違反量に応じたコストを付加した目的関数を最小化するという方針がとられている [4].

本研究では, 通常定数として扱われる移動時間を, あらたに定義しなおして考慮制約として考える. 従来の時間枠つき配送計画問題 [4] では, 現在の客と次の客のサービス開始時刻の間に, 現在の客をサービスする時間, 次の客へ移動する時間, および待ち時間があった. そしてこれらのうち, サービス時間と移動時間は定数として与えられ, 待ち時間が非負という条件が絶対制約として与えられていた. しかし, 多くの場合, それらはコストやリスクを伴えば縮めることが可能であると思われる. そこで本研究では, 現在の客をサービスする時間, 次の客へ移動する時間, および待ち時間の 3 つを 1 つにまとめて, 現在の客と次の客のサービス時刻の差をあらためて移動時間と定義し, 移動時間に対して移動時間コスト関数を導入する. 本研究の狙いは, たとえば移動時間をある値より短くすれば, その量に応じてコストがかかる関数を用意して, 移動時間で多少無理をしても, 他の重要なコストを大幅に削減できるような柔軟性の高い配送計画を行うことである. また, 本研究で用いる定式化は, 従来の時間枠つき配送計画問題を表現可能であり, 従来の問題よりもさらに汎用性が高くなっている.

ところで, このような考慮制約の違反度を表現するコストの処理は, 容量コストについては容易であるが, 時間枠コストと移動時間コストについては, 1つの車両が処理する客のルートを定めたのち, これらのコストを最小とするように各客のサービス時刻を決定しなければならない. しかし, 移動時間コスト関数と時間枠コスト関数が一般であれば, 各客のサービス時刻を決定する問題はNP困難 (3.1 節の定理 3.1) であることが分かった.

そこで, 本研究では, 移動時間コスト関数を区分線形凸関数に限定し, この場合には動的計画法により多項式時間で最適サービス時刻が求まることを示す. 現実の問題においても, 移動時間コスト関数が凸である場合は多いと思われる. また, この制限を加えても, 移動時間がある定数以上という従来の制約を扱うことは可能である. なお, 提案する動的計画法は, 時間枠制約に対しては, コスト関数が区分線形関数であれば不連続や非凸であっても扱えるので, 例えば, 時間枠が2つあり, そのどちらでサービスが行われても良いというような状況にも対応出来る. さらに, この動的計画法を組み込んだ局所探索法を提案し, その効果を計算実験により確認する.

代表的なベンチマーク問題に対する計算実験の結果, 移動で多少無理をすることを許せば, これまでの最良解よりも車両台数を減らしても, 時間枠制約と容量制約をみたす配送計画が得られることがいくつかの問題例に対して確認できた.

2 問題定義

本節では, 本研究で扱う移動時間コスト関数を考慮した時間枠つき配送計画問題の定義を行う. 節点集合 $V = \{0, 1, \ldots, n\}$ と枝集合 $E = \{(i,j) | \ i,j \in V, i \neq j\}$ で与えられる完全有向グラフ $G = (V, E)$ と車両集合 $M = \{1, 2, \ldots, m\}$ を考える. ここで節点0はデポと呼ばれる特殊な節点であり, また, 他の節点はサービスを受ける客を表す. 各客 $i \in V$, 各車両 $k \in M$, 各枝 $(i,j) \in E$ には以下のものが与えられる.

- サービス要求量 a_i (≥ 0).
- サービス時刻 t に対する時間枠コスト関数 $p_i(t)$ (≥ 0).
- 車両 k の容量 u_k (≥ 0).
- 枝 (i,j) の距離 d_{ij} (≥ 0).
- 枝 (i,j) での移動時間が t であるときの移動時間コスト関数 $q_{ij}(t)$ (≥ 0).

時間枠コスト関数 p_i は区分線形関数とし, 不連続点 t においては $p_i(t) \leq \lim_{\varepsilon \to 0} \min\{p_i(t+\varepsilon), p_i(t-\varepsilon)\}$ を満たすものとする. また, $t < 0$ では $p_i(t) = +\infty$ を仮定して, 各客のサービス時刻は0以降であるとする. 移動時間コスト関数 q_{ij} についても同様に, 区分線形関数で, 不連続点 t においては $q_{ij}(t) \leq \lim_{\varepsilon \to 0} \min\{q_{ij}(t+\varepsilon), q_{ij}(t-\varepsilon)\}$ を満たし, $t < 0$ では $q_{ij}(t) = +\infty$ として各客間の移動時間は0以上であるとする. さらに, $t \geq 0$ では $p_i(t) \geq 0$ かつ $q_{ij}(t) \geq 0$ であると仮定する. 以上の仮定は, サービス時刻が $+\infty$ となることを排除し最適解が存在することを保証するためのものである. 区分線形関数の各区分の情報は明示的に入力として与えられるものとする. また q_{ij} は3.1節を除いて凸関数であるとする.

ここで, 各車両 k についてそのルートを σ^k, 車両 k の h 番目の客を $\sigma^k(h)$ で表し, $\boldsymbol{\sigma} = (\sigma^1, \sigma^2, \ldots, \sigma^m)$ とする. さらに, 車両 k が訪問する客数を n_k とする. なお, 各車両はデポを出発しデポに帰還するため, $\sigma^k(0) = \sigma^k(n_k + 1) = 0$ と定義する. さらに, s_i を客 i でのサービス時刻, s^{a}_k を車両 k がデポに帰還する時刻とし, $\boldsymbol{s} = (s_1, s_2, \ldots, s_n, s^{\mathrm{a}}_1, s^{\mathrm{a}}_2, \ldots, s^{\mathrm{a}}_m)$ とする. すべての車両は時刻0にデポを出発するものとし, 便宜上 $s_0 = 0$ と定義する. 以上を用いて, 配送スケジュールは $(\boldsymbol{\sigma}, \boldsymbol{s})$ で与えらる.

ルートを $\boldsymbol{\sigma}$ としたとき, $y_{ik}(\boldsymbol{\sigma}) \in \{0,1\}, i \in V, k \in M$ を, 車両 k が客 i をサービスするなら $y_{ik}(\boldsymbol{\sigma}) = 1$, そうでないなら $y_{ik}(\boldsymbol{\sigma}) = 0$ と定義する. $d_{\mathrm{sum}}(\boldsymbol{\sigma}), p_{\mathrm{sum}}(\boldsymbol{s}), a_{\mathrm{sum}}(\boldsymbol{\sigma}), q_{\mathrm{sum}}(\boldsymbol{\sigma}, \boldsymbol{s})$ をそれぞれ車両の総移動距離, サービス時刻に対するコストの総和, 車両の容量超過量の総和, 移動時間コストの総和とすると,

$$
\begin{aligned}
d_{\mathrm{sum}}(\boldsymbol{\sigma}) &= \sum_{k \in M} \sum_{h=0}^{n_k} d_{\sigma^k(h),\sigma^k(h+1)} \\
p_{\mathrm{sum}}(\boldsymbol{s}) &= \sum_{i \in V \setminus \{0\}} p_i(s_i) + \sum_{k \in M} p_0(s_k^{\mathrm{a}}) \\
a_{\mathrm{sum}}(\boldsymbol{\sigma}) &= \sum_{k \in M} \max\left\{ \sum_{i \in V} a_i y_{ik}(\boldsymbol{\sigma}) - u_k, 0 \right\} \\
q_{\mathrm{sum}}(\boldsymbol{\sigma}, \boldsymbol{s}) &= \sum_{k \in M} \sum_{h=0}^{n_k - 1} q_{\sigma^k(h),\sigma^k(h+1)} \left(s_{\sigma^k(h+1)} - s_{\sigma^k(h)} \right) \\
&\quad + \sum_{k \in M} q_{\sigma^k(n_k),0} \left(s_k^{\mathrm{a}} - s_{\sigma^k(n_k)} \right)
\end{aligned}
\tag{1}
$$

と表せる. したがって移動時間コスト関数つき配送計画問題は以下のように定式化できる.

$$
\begin{aligned}
&\text{minimize} && cost(\boldsymbol{\sigma}, \boldsymbol{s}) = d_{\mathrm{sum}}(\boldsymbol{\sigma}) + p_{\mathrm{sum}}(\boldsymbol{s}) + a_{\mathrm{sum}}(\boldsymbol{\sigma}) + q_{\mathrm{sum}}(\boldsymbol{\sigma}, \boldsymbol{s}) && (2) \\
&\text{subject to} && \sum_{k \in M} y_{ik}(\boldsymbol{\sigma}) = 1, \qquad \forall i \in V \setminus \{0\}. && (3)
\end{aligned}
$$

3 最適サービス時刻決定問題

本節では, ルート $\boldsymbol{\sigma}$ が与えられたときに各客のサービス時刻を最適化する問題を考える. この問題は車両ごとに独立である. 各車両 $k \in M$ に対し,

$$
\begin{aligned}
p_{\mathrm{sum}}^k(\boldsymbol{s}) &= \sum_{h=1}^{n_k} p_{\sigma^k(h)}(s_{\sigma^k(h)}) + p_0(s_k^{\mathrm{a}}) \\
q_{\mathrm{sum}}^k(\boldsymbol{s}) &= \sum_{h=0}^{n_k - 1} q_{\sigma^k(h),\sigma^k(h+1)} \left(s_{\sigma^k(h+1)} - s_{\sigma^k(h)} \right) + q_{\sigma^k(n_k),0} \left(s_k^{\mathrm{a}} - s_{\sigma^k(n_k)} \right)
\end{aligned}
$$

を定義する. 車両 k に対する最適サービス時刻決定問題は,

$$
\begin{aligned}
&\text{minimize} && p_{\mathrm{sum}}^k(\boldsymbol{s}) + q_{\mathrm{sum}}^k(\boldsymbol{s}) && (4) \\
&\text{subject to} && \boldsymbol{s} \in \boldsymbol{R}^{n+m}
\end{aligned}
$$

と定義される. 本節では, まずこの問題が一般にはNP困難であることを示したのち, 移動時間コスト関数 q_{ij} が凸の場合は多項式時間で解けることを示す.

3.1 NP困難性

本節では, 最適サービス時刻決定問題が一般にはNP困難であることを示す. 本節に限って移動時間コスト関数 q_{ij} は凸に限らない一般の区分線形関数であるとする (ただし, 2節で述べたその他の仮定はみたすものとする). このとき次の定理が成り立つ.

定理 3.1 移動時間コスト関数 $q_{ij}(t), (i,j) \in E$ が一般である場合 , 時間枠コスト関数 $p_i(t), i \in V$ が凸であっても , 客の最適サービス時刻 $\boldsymbol{s}$ を決定する問題はNP困難である.

証明. 0-1 ナップサック問題がこの問題に帰着可能であることを示す. 0-1 ナップサック問題は,

$$\begin{aligned} \text{maximize} \quad & \sum_{i=1}^{n} c_i x_i \\ \text{subject to} \quad & \sum_{i=1}^{n} w_i x_i \leq b, \\ & x_i \in \{0,1\}, \qquad i = 1,2,\ldots,n \end{aligned} \tag{5}$$

と定義され, 代表的な NP 困難問題の 1 つである. ここで 0-1 ナップサック問題の目的関数 (5) は

$$\text{minimize} \quad \sum_{i=1}^{n} c_i(1-x_i) = \sum_{i=1}^{n} c_i - \sum_{i=1}^{n} c_i x_i \tag{6}$$

と等価であることに注意する.

簡単のため, 車両は客 $1,2,\ldots,n-1$ をサービスするとし, 客 i を i 番目にサービスするとする. また節点 n はデポとする. さらに, デポへの帰還時刻を s_n と表すことにすると, 最適サービス時刻決定問題は

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{n} p_i(s_i) + \sum_{i=1}^{n} q_{i-1,i}(s_i - s_{i-1}) \\ \text{subject to} \quad & \boldsymbol{s} \in \boldsymbol{R}^n \end{aligned} \tag{7}$$

と書ける. 与えられた 0-1 ナップサック問題の問題例に対する最適サービス時刻決定問題の移動時間コスト関数 $q_{i-1,i}(t)$, $i = 1,2,\ldots,n$ と時間枠コスト関数 $p_i(t)$, $i = 1,2,\ldots,n$ を次のように定める.

$$p_i(t) = \begin{cases} 0, & t \in [0,b] \\ +\infty, & t \in (b,+\infty) \end{cases} \tag{8}$$

$$q_{i-1,i}(t) = \begin{cases} c_i, & t \in [0,w_i) \\ 0, & t \in [w_i,+\infty) \end{cases} \tag{9}$$

0-1 ナップサック問題の実行可能解 $\tilde{\boldsymbol{x}}$ に対し, $\tilde{\boldsymbol{s}}$ を

$$\begin{aligned} \tilde{s}_0 &= 0 \\ \tilde{s}_i &= \begin{cases} \tilde{s}_{i-1}, & \tilde{x}_i = 0 \\ \tilde{s}_{i-1} + w_i, & \tilde{x}_i = 1 \end{cases} \end{aligned}$$

と定める. $\tilde{\boldsymbol{x}}$ は実行可能解だから

$$\tilde{s}_i = \textstyle\sum_{j \leq i} w_j \tilde{x}_j \leq b, \quad i = 1,2,\ldots,n \tag{10}$$

が成り立つ. よって, (8) より $\sum_{i=1}^{n} p_i(\tilde{s}_i) = 0$, また (9) より $\sum_{i=1}^{n} q_{i-1,i}(\tilde{s}_i - \tilde{s}_{i-1}) = \sum_{i=1}^{n} c_i(1-\tilde{x}_i)$. すなわち $\tilde{\boldsymbol{s}}$ の目的関数値は $\tilde{\boldsymbol{x}}$ の目的関数値に一致する.

逆にサービス時刻決定問題に対して有限の目的関数値をもつ解 $\hat{\boldsymbol{s}}$ に対し, $\hat{\boldsymbol{x}}$ を以下のように定義する.

$$\hat{x}_i = \begin{cases} 1, & \hat{s}_i - \hat{s}_{i-1} \geq w_i \\ 0, & \hat{s}_i - \hat{s}_{i-1} < w_i \end{cases} \tag{11}$$

このとき,

$$\sum_{i=1}^{n} w_i \hat{x}_i \leq \sum_{i=1}^{n} (\hat{s}_i - \hat{s}_{i-1})\hat{x}_i \leq \sum_{i=1}^{n} (\hat{s}_i - \hat{s}_{i-1}) = \hat{s}_n \leq b. \tag{12}$$

なお, 最後の不等式は $\hat{s}$ の目的関数値が有限であることと (8) 式より従う. また, $\hat{s}$ の目的関数値は有限であるから, $\sum_{i=1}^{n} p_i(s_i) = 0$ であることと, (9) より

$$\sum_{i=1}^{n} c_i(1 - \hat{x}_i) = \sum_{i=1}^{n} p_i(\hat{s}_i) + \sum_{i=1}^{n} q_{i-1,i}(\hat{s}_i - \hat{s}_{i-1}). \tag{13}$$

以上より (6) と (7) の最適値は一致し, 0-1 ナップサック問題は最適サービス時刻決定問題に帰着された. ∎

3.2 動的計画法

本節では, 車両 k のルート σ^k が与えられたとき, 各客間の移動時間コストと各客の時間枠コストの総和を最小にする最適サービス時刻を求める動的計画法を提案する. 2 節ですでに述べたように, 本節以降では移動時間コスト関数 $q_{ij}(t), (i,j) \in E$ は凸関数であるとする.

3.2.1 動的計画法の基本戦略

車両 k が時刻 t に客 $\sigma^k(h)$ のサービスをするとき, 客 $\sigma^k(0), \sigma^k(1), \ldots, \sigma^k(h)$ をすべてサービスするのにかかる移動時間コストと時間枠コストの総和の最小値を $f_h^k(t)$ と定義し, 前向きコスト最小関数と呼ぶ. また, 簡単のため

$$\begin{aligned} p_h^k(t) &= p_{\sigma^k(h)}(t) \\ q_h^k(t) &= q_{\sigma^k(h),\sigma^k(h+1)}(t) \end{aligned}$$

と定義する.

$f_h^k(t)$ は漸化式

$$\begin{aligned} f_0^k(t) &= \begin{cases} +\infty, & t \neq 0 \\ 0, & t = 0 \end{cases} \\ f_h^k(t) &= p_h^k(t) + \min_{t'}\{f_{h-1}^k(t') + q_{h-1}^k(t - t')\}, \quad 1 \leq h \leq n_k + 1, -\infty < t < +\infty \end{aligned} \tag{14}$$

によって計算できる. このときルート σ^k の時間枠コストと移動時間コストの総和の最小値は $\min_t f_{n_k+1}^k(t)$ を計算することにより得られる.

3.2.2 f_h^k の計算

漸化式 (14) を計算するためのデータ構造について考える. $p_h^k(t)$ と $q_h^k(t)$ が区分線形関数であることから, $f_h^k(t)$ も区分線形関数となる. よって, これらの関数を, 関数の各区分を要素とする連結リストで記憶する. すなわち, リストの要素には区分線形関数の各区間とその区間での関数が記憶される. 今後の解析のために, $\delta(g)$ を区分的線形関数 g の区分数とする.

さて, 式 (14) で問題となるのは $\min_{t'}\{f_{h-1}^k(t') + q_{h-1}^k(t - t')\}$ の計算である. q_{h-1}^k は凸関数であるが, f_{h-1}^k は一般に凸関数ではない. そこで, $f_{h-1}^k(t)$ が凸関数であるような t の区間で極大なものを凸区分と呼び, f_{h-1}^k の定義域を凸区分に分割する. $\bar{\delta}(g)$ を関数 g の凸区分の数と定義する. f_{h-1}^k の凸区分の数を $K = \bar{\delta}(f_{h-1}^k)$, 各凸区分を $S_1, S_2, \ldots, S_K$ とする. ただし $t_i \in S_i, t_j \in S_j, i < j \Rightarrow t_i < t_j$ とする. この凸区分を拡張して次のように凸関数をつくる.

$$F_i(t) = \begin{cases} +\infty, & t \notin S_i \\ f_{h-1}^k(t), & t \in S_i \end{cases}$$

ここで,

$$E_i(t) = \min_{t'}\{F_i(t') + q^k_{h-1}(t-t')\} \tag{15}$$

とおけば,

$$\begin{aligned}\min_{t'}\{f^k_{h-1}(t') + q^k_{h-1}(t-t')\} &= \min_{t'}\min_{1\le i\le K}\{F_i(t') + q^k_{h-1}(t-t')\}\\ &= \min_{1\le i\le K}\min_{t'}\{F_i(t') + q^k_{h-1}(t-t')\}\\ &= \min_{1\le i\le K} E_i(t)\end{aligned}$$

とできる. すなわち $\min_{t'}\{f^k_{h-1}(t') + q^k_{h-1}(t-t')\}$ は $E_1, E_2, \ldots, E_K$ の下側エンベロープである.

3.2.3 E_i の計算

$E_i(t) = \min_{t'}\{F_i(t') + q^k_{h-1}(t-t')\}$ の計算において, F_i と q^k_{h-1} が共に区分的線形凸関数であることから次の定理が成り立つ.

定理 3.2 E_i は F_i と q^k_{h-1} から $O(\delta(E_i))$ 時間で計算できる. また, $\delta(E_i) \le \delta(F_i) + \delta(q^k_{h-1})$ が成り立つ.

証明. まず図 1 のような 2 次元平面を考える. 横軸には客 $\sigma^k(h-1)$ のサービス時刻 s^k_{h-1}, 縦軸に

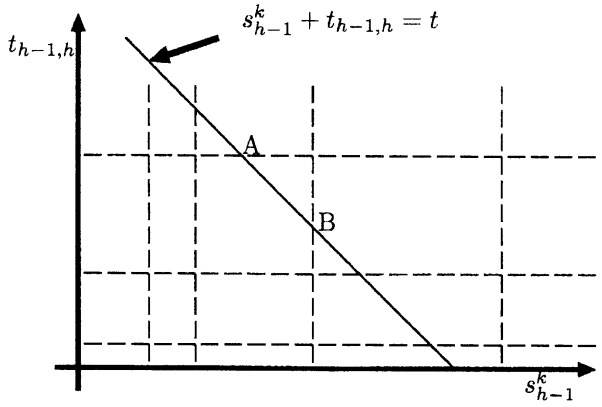


図 1: サービス時刻 s^k_{h-1} と移動時間 $t_{h-1,h}$

は $\sigma^k(h-1)$ から $\sigma^k(h)$ への移動時間 $t_{h-1,h} = s^k_h - s^k_{h-1}$ をとる. 2 次元平面の各点 $(s^k_{h-1}, t_{h-1,h})$ には値 $F_i(s^k_{h-1}) + q^k_{h-1}(t_{h-1,h})$ が対応するものとする. 図中の縦の破線, 横の破線はそれぞれ区分線形凸関数 F_i と q^k_{h-1} の区分の変り目を表す. この図において $E_i(t)$ は, 直線 $s_{h-1} + t_{h-1,h} = t$ が通る点の中で値 $F_i(s_{h-1}) + q^k_{h-1}(t_{h-1,h})$ が最小のものである. まず, 線分 AB 上にある点で最小値を実現する点について考える. 線分 AB を含む極小の破線四角形は横は F_i, 縦は q^k_{h-1} の 1 つの区分からできているから, 線分 AB 上にある点の値の変化は一様である. つまり線分 AB 上にある点の値は, A から B へいくにつれて大きくなるか, 小さくなるか, または線分 AB 上の点はすべて同じ値かのどれかである. 従って, 線分 AB 上にある点で最小値を実現する点は A と B のどちらかであるとしてよい. 次に, 直線 $s_{h-1} + t_{h-1,h} = t$ 上の点について考える. 直線 $s_{h-1} + t_{h-1,h} = t$ は線分 AB のような破線四角形に含まれる線分に分割できる. 各線分において, その線分上で値が最小になる点は破線格子上であるとしてよい. 従って, 任意の t において, $E_i(t)$ は破線格子上で実現されるとしてよい.

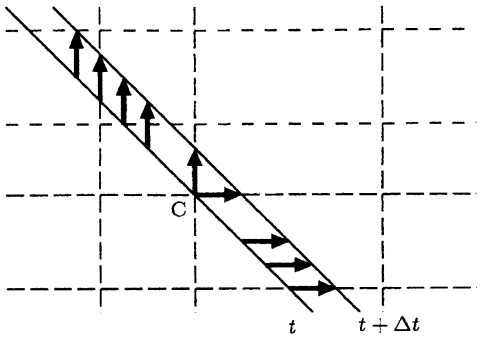


図 2: $E_i(t+\Delta t)$

いま図2の点Cで$E_i(t)$が実現されているものとする. このとき$t+\Delta t$で$E_i(t+\Delta t)$がどの点で実現されるかを考える. 図中には上への矢印と右への矢印がある. 上への矢印の中で傾きが最も小さいものは点Cからでている矢印である. というのは, この図で縦方向はq^k_{h-1}を表し, q^k_{h-1}は凸関数だからである. 同様に, 右への矢印の中で傾きが最も小さいものは点Cからでている矢印である. すなわち点Cからでている上への矢印と右への矢印のうちで傾きの小さい方の矢印で, $E_i(t+\Delta t)$が実現される. 従って$E_i(t)$を求めるには, 図1の$(0,0)$から格子上をたどっていき, 格子点で上の区分と右の区分の傾きの小さい方に進みつつ, 対応する区分の線形関数をリストに追加していけばよいので$O(\delta(E_i))$時間で計算可能である. また, 以上の議論より, $\delta(E_i) \leq \delta(F_i) + \delta(q^k_{h-1})$がただちに分かる. ∎

3.2.4 $E_1, E_2, \ldots, E_K$の下側エンベロープ

本節では$E_1, E_2, \ldots, E_K$の下側エンベロープを$O\left(\sum_{i=1}^{K} \delta(E_i)\right)$時間で求める方法について述べる.

$E_1, E_2, \ldots, E_K$の下側エンベロープを表すのに必要な情報は,

- 下側エンベロープをtが小さい方から大きい方へみたとき, E_iがどのような順に現れるか,
- E_iからE_jへ入れ替わると分かったときにどこで入れ替わるか,

の2つである. これらを求めるおおまかな流れは次の通りである.

下側エンベロープの求め方

0. $k = 1$とする.

k. $E_1, E_2, \ldots, E_k$の下側エンベロープ$\min_{1 \leq i \leq k} E_i(t)$を求める.
$k = K$なら終了. そうでなければ$k := k+1$としてkへ.

ステップkでは, ステップ$k-1$で$E_1, E_2, \ldots, E_{k-1}$の下側エンベロープを求めているから, そこに$E_k$がどのように影響を及ぼすかだけを調べればよい.

ここで下側エンベロープの順列について, 以下の補題3.3が成り立つ.

補題 3.3 任意の時刻tにおいて, $i < j$ならばE_iの傾きはE_jの傾き以上である.

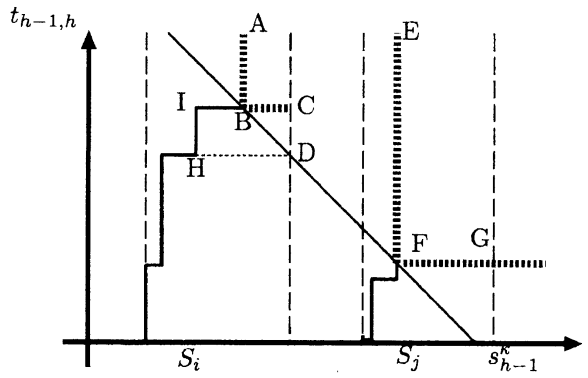


図 3: E_i と E_j

証明. 図 3 は, 図 1 と同様のグラフを, E_i と E_j の 2 つに対して同時に図示したものである. 図中, E_i と E_j がそれぞれ点 B と F まで分かっているものとする. このとき点 B 以降の E_i は線分 AB と BC の中で傾きの小さい区分を選んで進む. 同様に, 点 F 以降の E_j は線分 EF, FG の中で傾きの小さい区分を選んで進む.

ここで E_i を点 B まで計算した結果, BC に含まれる区分は選ばれておらず, HI に含まれる区分が選ばれていることに着目する. E_i の計算方法から, BC に含まれる区分の傾きは HI に含まれる区分の傾き以上であることが分かる. 従って, BC に含まれる区分の傾きは CD に含まれるどの区分の傾きよりも大きいか等しい.

EF は AB と CD に含まれる区分をすべて含んでおり, また, E_j は EF と FG の中で傾きの小さい区分を選んで進むのだから, 点 B 以降で現れる $E_i(t)$ の傾きは, 点 F 以降で現れる $E_j(t)$ の傾き以上である. ∎

これを用いて以下の定理 3.4 が成り立つ.

定理 3.4 $E_1, E_2, \ldots, E_k$ のそれぞれは下側エンベロープに高々1 回現れ, その出現順序は添字の昇順になっている.

証明. 以下, $i < j$ とし, t を次第に大きくしていくときの $E_i(t)$ と $E_j(t)$ の大小関係を考える. 補題 3.3 より, $E_i(t)$ と $E_j(t)$ の大小関係はたかだか 1 回しか入れ替わらない. また入れ替わる場合は $E_i(t) \leq E_j(t)$ から $E_i(t) > E_j(t)$ へと入れ替わる. よって下側エンベロープへの出現順序は添字の昇順になっている. ∎

以下では, $E_1, E_2, \ldots, E_K$ の下側エンベロープの求め方を考える. たとえば図 4 のように, E_1, E_2, E_3 の下側エンベロープが分かっているときに E_1, E_2, E_3, E_4 の下側エンベロープを求めたいときには, 以下のように計算すればよい.

1. まず E_4 を右からみていって, 下側エンベロープと大小関係が入れ替わるかどうか調べる. それには, まず E_2 と E_3 の交点と E_4 の大小関係をみる. この場合は E_4 が下になっているので, さらに左に進んで E_1 と E_2 の交点との大小関係をみる. ここで E_4 が上になっていて, 大小関係が入れ替わったことが分かる. すなわち, この時点で, E_4 は E_2 と交わることが判定できる.

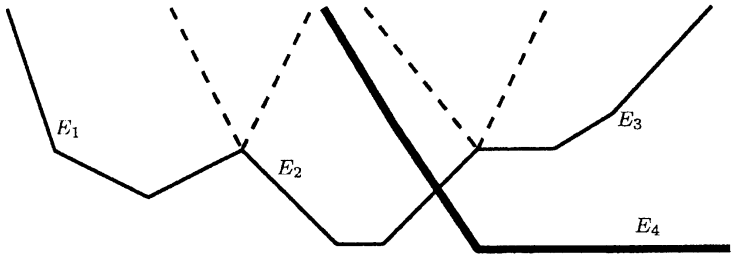


図 4: 下側エンベロープ

2. つぎに E_2 と E_3 の交点から左に下側エンベロープ (すなわち E_2) をたどっていき, E_4 との交点を求める.

$E_1, E_2, \ldots, E_{k-1}$ までの下側エンベロープに E_k を追加するときの計算も同様である. すなわち, $E_1, E_2, \ldots, E_{k-1}$ の下側エンベロープに現れるこれらの交点と対応する E_k の値との大小関係を右から順に調べていき, その中で E_k が上になることが分かった最初の交点をまず求め, 次にその 1 つ手前の交点から下側エンベロープを右から左へたどりつつ, E_k との交点を求めるのである. 以上を効率よく計算するため, $E_1, E_2, \ldots, E_{k-1}$ の下側エンベロープにおける E_i と E_j の交点を右から順に記憶した連結リストと, 各交点からその交点を実現する 2 つの区分が参照できるようなデータ構造を用意しておく. E_k との大小比較において上にあることが分かった交点や区分はそれ以降の探索で調べる必要がないことに注意すると, 上述の操作を繰り返して $E_1, E_2, \ldots, E_K$ の下側エンベロープを求める計算時間は, 既存の交点と新たに追加する E_k との大小比較に

$$O\left(\sum_{i=1}^{K}\delta(E_i) + \text{大小比較した交点の数}\right), \tag{16}$$

新たな交点を求めるのに

$$O\left(\text{交点を求めるためにたどった下側エンベロープの区分数} + \sum_{i=1}^{K}\delta(E_i)\right) \tag{17}$$

であり, 以上を合わせて $O(\sum_{i=1}^{K}\delta(E_i))$ 時間である.

3.2.5 最適サービス時刻を求める計算時間

(14) 式で f_{h-1}^k から f_h^k を計算するには,

1. $E_1, E_2, \ldots, E_{\overline{\delta}(f_{h-1}^k)}$ を計算する
2. $E_1, E_2, \ldots, E_{\overline{\delta}(f_{h-1}^k)}$ の下側エンベロープを計算する
3. p_h^k を足す

を行わなければならない.

1. あるひとつの i に対して E_i を計算するには $O(\delta(E_i))$ だけかかるので, E_i をすべて計算するのは $O(\sum_{i=1}^{\overline{\delta}(f_{h-1}^k)}\delta(E_i))$ 時間で可能である.
2. 下側エンベロープを求めるのも 1 と同じ時間で可能である.

3. p_h^k を足すのは $O\left(\sum_{i=1}^{\overline{\delta}(f_{h-1}^k)} \delta(E_i) + \delta(p_h^k)\right)$ 時間で計算できる.

以上の 1 から 3 をすべて合わせると $O\left(\sum_{i=1}^{\overline{\delta}(f_{h-1}^k)} \delta(E_i) + \delta(p_h^k)\right)$ である. ここで

$$\begin{aligned}
\Delta_p^k &= \sum_{h=1}^{n_k+1} \delta(p_h^k) \\
\overline{\Delta}_p^k &= \sum_{h=1}^{n_k+1} \overline{\delta}(p_h^k) \\
\Delta_q^k &= \sum_{h=0}^{n_k} \delta(q_h^k)
\end{aligned}$$

とおく. 求めた f_h^k の凸区分の数に着目すると

$$\begin{aligned}
\overline{\delta}(f_h^k) &\leq \overline{\delta}(f_{h-1}^k) + \overline{\delta}(p_h^k) - 1 \\
&= O(\overline{\Delta}_p^k)
\end{aligned}$$

が成り立ち, また f_h^k の区分数は, 下側エンベロープの区分数と $\delta(p_h^k)$ の和以下になるので

$$\begin{aligned}
\delta(f_h^k) &\leq \sum_{i=1}^{\overline{\delta}(f_{h-1}^k)} \delta(E_i) + \delta(p_h^k) \\
&\leq \sum_{i=1}^{\overline{\delta}(f_{h-1}^k)} \left(\delta(F_i) + \delta(q_{h-1}^k)\right) + \delta(p_h^k) \\
&= \delta(f_{h-1}^k) + \overline{\delta}(f_{h-1}^k)\delta(q_{h-1}^k) + \delta(p_h^k) \\
&\leq \delta(f_0^k) + \sum_{l=1}^{h} \left(\overline{\delta}(f_{l-1}^k)\delta(q_{l-1}^k) + \delta(p_l^k)\right) \\
&= \sum_{l=1}^{h} O(\overline{\Delta}_p^k)\delta(q_{l-1}^k) + O(\Delta_p^k) \\
&= O(\overline{\Delta}_p^k \Delta_q^k + \Delta_p^k)
\end{aligned}$$

が成り立つ. 上の計算過程より, $\sum_{i=1}^{\overline{\delta}(f_{h-1}^k)} \delta(E_i) + \delta(p_h^k) = O(\overline{\Delta}_p^k \Delta_q^k + \Delta_p^k)$ が分かるので, f_{h-1}^k から f_h^k を求める計算時間は $O(\overline{\Delta}_p^k \Delta_q^k + \Delta_p^k)$ となる.

よって, $f_1^k, f_2^k, \ldots, f_{n_k+1}^k$ を求める計算時間は $O\left(n_k(\overline{\Delta}_p^k \Delta_q^k + \Delta_p^k)\right)$ となる. 従って最適サービス時刻を計算するのには $O\left(n_k(\overline{\Delta}_p^k \Delta_q^k + \Delta_p^k)\right)$ 時間で可能である. 区分線形関数 p_h^k と q_h^k の各区分の情報は明示的に入力として与えられているので Δ_q^k と Δ_p^k は入力サイズのオーダーであり, また, $\overline{\Delta}_p^k \leq \Delta_p^k$ である. よって, これは多項式時間である.

4 局所探索法

まず, 局所探索法の探索空間について考察する. ルート $\boldsymbol{\sigma} = (\sigma^1, \sigma^2, \ldots, \sigma^m)$ を決定すると, 目的関数における $d_{\mathrm{sum}}(\boldsymbol{\sigma})$ と $a_{\mathrm{sum}}(\boldsymbol{\sigma})$ の値は簡単に計算できる. また, $p_{\mathrm{sum}}(\boldsymbol{s})$ と $q_{\mathrm{sum}}(\boldsymbol{\sigma}, \boldsymbol{s})$ についても, 前節で述べたように $p_{\mathrm{sum}}(\boldsymbol{s}) + q_{\mathrm{sum}}(\boldsymbol{\sigma}, \boldsymbol{s})$ が最小となるような各客 i のサービス時刻 s_i を動的計画法により計算できる. つまり, ルート $\boldsymbol{\sigma}$ が決まれば, 対応する最適な配送スケジュール $(\boldsymbol{\sigma}, \boldsymbol{s})$ が

定まる. 以上より, 探索空間をルート $\boldsymbol{\sigma}$ の集合とする. なお, これ以降, 解を表現する際には, ルート $\boldsymbol{\sigma}$ のみを用いる.

解 $\boldsymbol{\sigma}$ の近傍 $NB(\boldsymbol{\sigma})$ とは, $\boldsymbol{\sigma}$ に対して適当な操作を加えて得られる解の集合のことである. 局所探索法は, 現在の解 $\boldsymbol{\sigma}$ の近傍 $NB(\boldsymbol{\sigma})$ 内に $\boldsymbol{\sigma}$ より良い解があればそれに置き換える, という操作を反復するものである. これを手続きとして以下にまとめておく.

LOCAL SEARCH

0. 適当な近似解法で初期解を求め, $\boldsymbol{\sigma}$ とする. $k=1$ とする.

k. $\boldsymbol{\sigma}$ の近傍 $NB(\boldsymbol{\sigma})$ 内で $\boldsymbol{\sigma}$ より良い目的関数値をもつ実行可能解 $\boldsymbol{\sigma'}$ を探す. そのような $\boldsymbol{\sigma'}$ が見つかれば, 解を $\boldsymbol{\sigma}$:=$\boldsymbol{\sigma'}$ と更新したのち $k := k+1$ として k へ. そうでなければ現在の解 $\boldsymbol{\sigma}$ を返す.

局所探索においては, 近傍をどのように定義するかによって最終的に得られる解の精度が大きく異なる. したがって個々の問題に応じた効率の良い近傍を定義する必要がある. 以下では, 本研究で用いる 3 つの近傍について述べた後, それらの近傍の探索順序について述べる.

4.1 近傍

本研究では, クロス交換近傍, 2-opt* 近傍, およびルート内挿入近傍の 3 つの近傍を組合せて用いる. 以下, これらを説明する.

4.1.1 クロス交換近傍と 2-opt* 近傍

異なる 2 つのルート間にまたがる近傍操作としてクロス交換操作と 2-opt* 操作がある.

まずクロス交換操作であるが, これは, 異なる 2 つのルートのそれぞれから長さ L^{cross} 以下のパスを選び, それらを互いに交換する操作である. そして, 現在の解 $\boldsymbol{\sigma}$ に対してこの操作を加えることによって得られる解集合の全体をクロス交換近傍と呼び, $N^{\mathrm{cross}}(\boldsymbol{\sigma})$ で表すことにする. 特に, 現在の解 $\boldsymbol{\sigma}=(\sigma^1,\ldots,\sigma^m)$ について, 2 つの異なるルート σ^k と $\sigma^{k'}$ に対してクロス交換操作を行って得られる解集合を $N^{\mathrm{cross}}(\boldsymbol{\sigma},k,k')$ とする. このとき,

$$N^{\mathrm{cross}}(\boldsymbol{\sigma}) = \bigcup_{k<k'} N^{\mathrm{cross}}(\boldsymbol{\sigma},k,k') \tag{18}$$

となる.

クロス交換操作の例を図 5 に示す. この図において黒い四角はデポを表し, 白丸はルート内の客を表す. まず, 一つのルートから 2 本の枝 $(\sigma^k(h_1^k-1),\sigma^k(h_1^k))$ と $(\sigma^k(h_2^k-1),\sigma^k(h_2^k))$ が抜かれ, さらにもう一方のルートから 2 本の枝 $(\sigma^{k'}(h_1^{k'}-1),\sigma^{k'}(h_1^{k'}))$ と $(\sigma^{k'}(h_2^{k'}-1),\sigma^{k'}(h_2^{k'}))$ が抜かれる. そして, 新しい 4 本の枝 $(\sigma^k(h_1^k-1),\sigma^{k'}(h_1^{k'}))$, $(\sigma^{k'}(h_2^{k'}-1),\sigma^k(h_2^k))$, $(\sigma^{k'}(h_1^{k'}-1),\sigma^k(h_1^k))$ および $(\sigma^k(h_2^k-1),\sigma^{k'}(h_2^{k'}))$ を加えることにより, パス $\sigma^k(h_1^k)\to\sigma^k(h_2^k-1)$ と $\sigma^{k'}(h_1^{k'})\to\sigma^{k'}(h_2^{k'}-1)$ が交換される.

クロス交換近傍の特別な場合として, 交換されるパスの長さに制限をおかない (すなわち $L^{\mathrm{cross}}=n$) 場合には, 以下の 2-opt* 近傍も含む. 2-opt* 近傍は, 異なる 2 本のルートそれぞれから 1 本ずつ枝を取り除くことで, 各ルートを前半と後半の 2 つのパスに分け, 後半のパスをルート間で互いに交換することにより得られる解集合である (図 6 参照). 2 つの異なるルート $\sigma^k,\sigma^{k'}$ に対して 2-opt* 操作を行って得られる解集合を $N^{2\mathrm{opt}^*}(\boldsymbol{\sigma},k,k')$, 2-opt* 近傍の全体を $N^{2\mathrm{opt}^*}(\boldsymbol{\sigma})$ と表す. すなわち,

$$N^{2\mathrm{opt}^*}(\boldsymbol{\sigma}) = \bigcup_{k<k'} N^{2\mathrm{opt}^*}(\boldsymbol{\sigma},k,k') \tag{19}$$

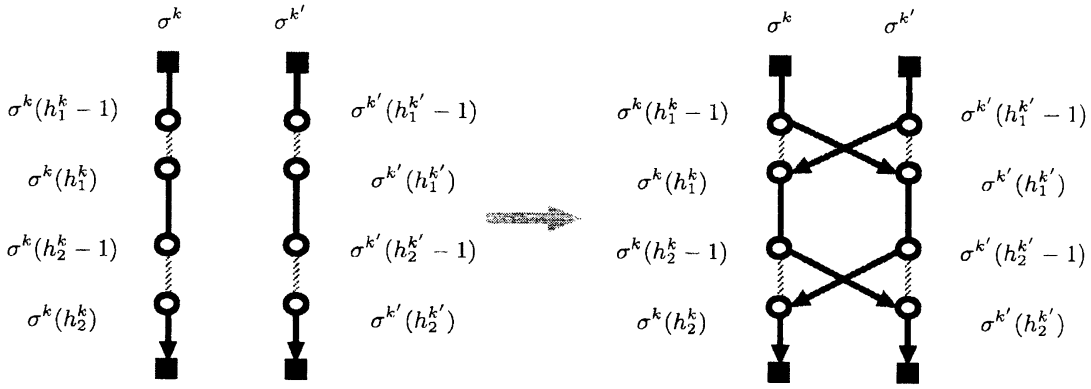


図 5: クロス交換操作の例

となる.

これはクロス交換近傍において, 交換する 2 つのパスそれぞれの最後の客 (図 5 の $\sigma^k(h_2^k-1), \sigma^{k'}(h_2^{k'}-1)$) が共に各ルートの最後の客であるときに相当する. しかし, 通常, 交換するパスの長さの上限 L^{cross} は小さな定数として設定されるが, この場合には, 2-opt* 近傍の中でクロス交換近傍に含まれない解が存在する. よって, 本研究では, 2-opt* 近傍をクロス交換近傍と別扱いし, 探索の候補に含める.

クロス交換近傍, 2-opt* 近傍共に, 交換の対象となるパス内の客の訪問順序を変えない解のみを探索するが, 時間枠制約付きの問題に対しては効果的であることが報告されている [5, 8].

以下 2 つの近傍のサイズを考察する. まず, クロス交換近傍であるが, 交換の対象となるパスの長さが最大 L^{cross} と制限されているので, 1 つのルートから取り除く 2 本の枝の決め方を全ルートで合計すると $O(nL^{\text{cross}})$ 通りである (n は考えている問題の総客数). よって, 近傍サイズは,

$$O(nL^{\text{cross}}) \times O(nL^{\text{cross}}) = O(n^2(L^{\text{cross}})^2) \tag{20}$$

となる. 次に 2-opt* 近傍であるが, 1 つのルートから取り除く 1 本の枝の選び方を全てのルートで合計すると $O(n)$ 通りあるので, 近傍サイズは,

$$O(n) \times O(n) = O(n^2) \tag{21}$$

となる.

4.1.2 ルート内挿入近傍

クロス交換近傍や 2-opt* 近傍は共に異なるルート間でのパスの交換を扱うもので, 同一ルート内での客の順序の入れ替えは候補に入っていない. 本研究では, 長さ $L_{\text{path}}^{\text{intra}}$ 以下のパスを同一ルートの他の位置へ挿入することによって得られる解集合をルート内挿入近傍と呼び, 探索に利用する. ただし挿入する位置は取り除かれるパスから $L_{\text{ins}}^{\text{intra}}$ 以内であるとする.

ルート σ^k に対してルート内挿入操作を行って得られる解集合を $N^{\text{intra}}(\boldsymbol{\sigma}, k)$, ルート内挿入近傍の全体を $N^{\text{intra}}(\boldsymbol{\sigma})$ と表す. すなわち,

$$N^{\text{intra}}(\boldsymbol{\sigma}) = \bigcup_{k \in M} N^{\text{intra}}(\boldsymbol{\sigma}, k) \tag{22}$$

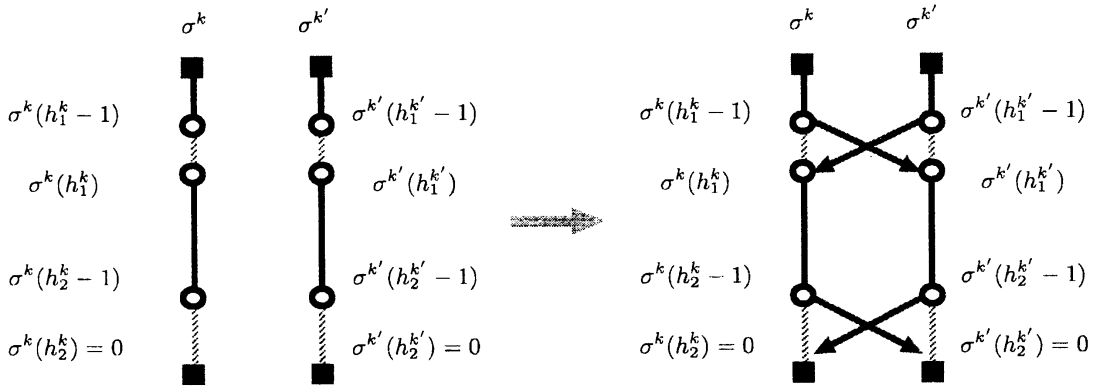


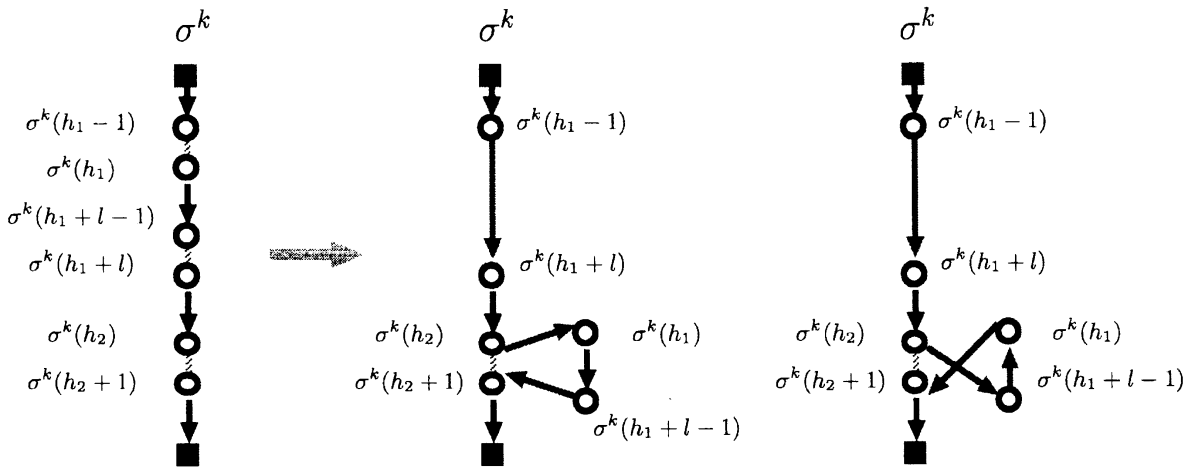
図 6: クロス変換の特別な場合: 2-opt*

となる. ルート内挿入近傍の一例を図 7 に示す. ただし, この図において l は, 取り除かれる客数を表す.

ルート内挿入近傍のサイズは, 取り除くパスの長さの決め方に $O(L_{\text{path}}^{\text{intra}})$ 通り, 取り除く位置の決め方に $O(n)$ 通り, 挿入する位置の決め方に $O(L_{\text{ins}}^{\text{intra}})$ 通りあるので, 全体で,

$$O(L_{\text{path}}^{\text{intra}}) \times O(n) \times O(L_{\text{ins}}^{\text{intra}}) = O(nL_{\text{path}}^{\text{intra}} L_{\text{ins}}^{\text{intra}}) \tag{23}$$

となる.



(a) パスの向きを変えずに挿入, (b) パスを反転して挿入

図 7: ルート内挿入: (a) パスの向きを変えずに挿入, (b) パスを反転して挿入

4.1.3 探索順序

本研究の局所探索法における上記で定めた近傍の探索順序は,

1. ルート内挿入近傍探索

2. 2-opt* 近傍探索

3. クロス近傍探索

である. さらに, 各探索は, その近傍内での解の改善が起こらなくなるまで行う. すなわち, ある探索によって解の改善が起こると, 繰り返し同じ近傍を探索するのである. また, クロス交換近傍探索を終えるとルート内挿入近傍探索に戻る. このループは, 上記の 3 つの近傍全てにおいて解の改善が起こらなくなるまで続ける.

5 局所探索法の高速化

この節では, 探索を高速化するための方法について考える. 説明の都合上, 目的関数 (2) を下記のように表記する.

$$cost(\boldsymbol{\sigma}) = d_{\mathrm{sum}}(\boldsymbol{\sigma}) + p_{\mathrm{sum}}(\boldsymbol{\sigma}) + q_{\mathrm{sum}}(\boldsymbol{\sigma}) + a_{\mathrm{sum}}(\boldsymbol{\sigma}) \tag{24}$$

現在の解を $\boldsymbol{\sigma}^{\mathrm{curr}}$, 近傍 $NB(\boldsymbol{\sigma}^{\mathrm{curr}})$ 内のある解を $\boldsymbol{\sigma}^{\mathrm{new}}$ とする. このとき

$$\begin{aligned}
\Delta d_{\mathrm{sum}} &= d_{\mathrm{sum}}(\boldsymbol{\sigma}^{\mathrm{new}}) - d_{\mathrm{sum}}(\boldsymbol{\sigma}^{\mathrm{curr}}) \\
\Delta p_{\mathrm{sum}} &= p_{\mathrm{sum}}(\boldsymbol{\sigma}^{\mathrm{new}}) - p_{\mathrm{sum}}(\boldsymbol{\sigma}^{\mathrm{curr}}) \\
\Delta q_{\mathrm{sum}} &= q_{\mathrm{sum}}(\boldsymbol{\sigma}^{\mathrm{new}}) - q_{\mathrm{sum}}(\boldsymbol{\sigma}^{\mathrm{curr}}) \\
\Delta a_{\mathrm{sum}} &= a_{\mathrm{sum}}(\boldsymbol{\sigma}^{\mathrm{new}}) - a_{\mathrm{sum}}(\boldsymbol{\sigma}^{\mathrm{curr}})
\end{aligned}$$

とすると, 解の改善が起こるのは以下で定義する $\Delta cost$ が負となるときである.

$$\begin{aligned}
\Delta cost &= cost(\boldsymbol{\sigma}^{\mathrm{new}}) - cost(\boldsymbol{\sigma}^{\mathrm{curr}}) \\
&= \Delta d_{\mathrm{sum}} + \Delta p_{\mathrm{sum}} + \Delta q_{\mathrm{sum}} + \Delta a_{\mathrm{sum}}
\end{aligned}$$

以下の節では, これら Δd_{sum}, $\Delta p_{\mathrm{sum}} + \Delta q_{\mathrm{sum}}$, および Δa_{sum} の計算に要する時間について考察する.

5.1 $\Delta p_{\mathrm{sum}} + \Delta q_{\mathrm{sum}}$ について

$\Delta p_{\mathrm{sum}} + \Delta q_{\mathrm{sum}}$ は, 近傍操作の対象となる 2 つ (ルート内挿入操作の場合は 1 つのみ) の解に対するコストの変化量のみを計算すれば求まる. しかし, この計算を, 3.2 節で述べた前向きコスト最小関数の漸化式 (14) を毎回計算し直していたのでは効率が悪い. そこで, 以下に述べる関数 $b_h^k(t)$ をあらかじめ用意しておくことで, この部分の計算の高速化を図る.

関数 $b_h^k(t)$ を, ルート σ^k における h 番目の客のサービス時刻が t で, 客 $\sigma^k(h), \sigma^k(h+1), \ldots, \sigma^k(n_k)$ をサービスするときの時間枠コストと移動時間コストの合計の最小値と定義し, これを以後, 後ろ向きコスト最小関数と呼ぶ. すると, $b_h^k(t)$ は, $f_h^k(t)$ の計算と対称的に, 以下の漸化式より計算できる.

$$\begin{aligned}
b_{n_k+1}^k(t) &= p_0^k(t) \\
b_h^k(t) &= p_h^k(t) + \min_{t'} \left(b_{h+1}^k(t') + q_h^k(t'-t) \right), \quad 1 \le h \le n_k
\end{aligned} \tag{25}$$

ある h に対し, 前向きコスト最小関数 $f_h^k(t)$ と後ろ向きコスト最小関数 b_{h+1}^k が既知であれば, これらを利用することにより, コストの最小値 $\min_t f_{n_k+1}^k(t)$ は, 1 つの h に対し

$$\min_t \left(f_h^k(t) + \min_{t'} \left(b_{h+1}^k(t') + q_h^k(t'-t) \right) \right) \tag{26}$$

と計算できるため, 漸化式 (14) を反復再計算して $\min_t f_{n_k+1}^k$ を求めるよりも効率的である. すなわち, (26) 式の計算は, (14) 式の計算において f_{h-1}^k から f_h^k を求める計算と同様にでき, $O(\overline{\Delta}_p^k \Delta_q^k + \Delta_p^k)$

時間で可能である. 2-opt* 近傍にこのアイデアが利用できることはすぐに分かる. それ以外の場合は, 交換されるパス内の客については前向きおよび後ろ向きコスト最小関数を計算しなおす必要があるが, [4] に述べられているように, 局所探索法の近傍内の探索順序を工夫して, 交換されるパスに対してこの計算を行う時点で, 長さが 1 つ短いパスに対する前向きおよび後向きコスト最小関数が既知であるようにしておくことで, 各 $\Delta p_{\mathrm{sum}} + \Delta q_{\mathrm{sum}}$ の計算にこのアイディアを利用できる. なお, この場合, 各客に対する前向きと後ろ向きのコスト最小関数をあらかじめ計算して記憶しておき, さらに, 解の更新時には, 変更の加わったルートに含まれる全ての客に対してこれらの関数を計算し直す必要があるが, 通常, 解の評価回数に比べて解の更新回数は十分小さいため, オーダー的には無視できる.

5.2 Δa_{sum} と Δd_{sum} について

まず, Δa_{sum} について考える. 各ルートに対し, ルート内の客の総要求量をあらかじめ計算し, 記憶しておく. 近傍操作の前後でのこの値の変化を考える. ルート内挿入近傍では, この値は変化しない. クロス交換近傍と 2-opt* 近傍では, 交換操作の対象となる 2 つのルートの値が変化する. 以下では, この変化量の効率的計算法を示す.

ルート σ^k 内の i 番目の客のサービス要求量を a_i^k と表す. このとき, ルート σ^k 内の各客 i に対して, 累積要求量を,

$$\begin{aligned} &\gamma_0^k = 0 \\ &\gamma_i^k = \textstyle\sum_{j=1}^{i} a_j^k = \gamma_{i-1}^k + a_i^k, \qquad\qquad i = 1, \ldots, n_k \end{aligned} \tag{27}$$

と定義し, 記憶しておく. この計算は, ルート σ^k に対して全体で $O(n_k)$ 時間で計算できる. すると, ルート内の i 番目から j 番目の客による部分パスの総要求量は, $\gamma_j^k - \gamma_{i-1}^k$ となり, 定数時間で計算できる. 局所探索を行うときの初期解の設定時と, 解の更新が行われたときに, 新しく生成されたルートに対して, ルート内の客の総要求量と γ_i^k を再計算する必要があるが, これは生成されたルートの客数の時間で可能であり, また, 局所探索法における解の評価回数に比べてこのような再計算の回数は十分小さいことから, この計算はオーダー的に無視できる.

Δd_{sum} については, パスをつなぎ変えるために入れ替える枝数が $O(1)$ 本であることと, (27) 式の Δa_{sum} と同様の計算方法でパスを反転したときのパス内の枝の距離の変化量を計算できることから (距離行列は一般には非対称), 探索順序によらず $O(1)$ 時間で計算できる.

6 反復局所探索法

反復局所探索法 (iterated local search, ILS) は, 過去の探索で得られたよい解にランダムな変形を加えたものを初期解として, 局所探索法を反復する方法である. これを手続きとして以下にまとめておく.

ILS

1. 初期解 $\boldsymbol{\sigma}$ を生成し, $\boldsymbol{\sigma}_{\mathrm{seed}} := \boldsymbol{\sigma}$ とする.
2. 局所探索法により $\boldsymbol{\sigma}$ を改善する.
3. $cost(\boldsymbol{\sigma}) \leq cost(\boldsymbol{\sigma}_{\mathrm{seed}})$ ならば, $\boldsymbol{\sigma}_{\mathrm{seed}} := \boldsymbol{\sigma}$ とする.
4. ある停止条件が満たされれば探索中に得られた最良解を出力して終了. そうでなければ $\boldsymbol{\sigma}_{\mathrm{seed}}$ に少しランダムな変形を加えた新たな初期解 $\boldsymbol{\sigma}$ を生成し, 2 に戻る.

本研究においては, 4. における $\boldsymbol{\sigma}_{\text{seed}}$ の変形は, ランダムなクロス交換操作を基本ルーチンとして行っている. ランダムなクロス交換操作とは, 異なる 2 本のルートをランダムに選択し, 交換されるパスの位置と長さもそれぞれランダムに選択し, 交換を行うというものである. そして, この操作を $\boldsymbol{\sigma}_{\text{seed}}$ に対して λ 以下のランダムな回数だけ繰り返して, 新しい初期解 $\boldsymbol{\sigma}$ を得る. λ は最初は 1 に設定しておいて, 前回 λ を更新したときから, $\boldsymbol{\sigma}_{\text{seed}}$ が 3 回連続更新されていないなら $\lambda := \lambda + 1$ とする. ただし λ が 3 の場合はそれ以上大きくしない. もし更新した場合は 1 に戻す.

7 計算実験

計算実験は, PC (Pentium4 2.8GHz, 1GB memory) 上で C 言語を用いて行った. 用いた問題例は, Solomon のベンチマーク問題 [6] である. この問題は, $[0, 100]^2$ の正方形内に 100 人の客が分布しており, 客 i と客 j の距離 d_{ij}, 移動時間 t_{ij} はユークリッド距離に等しい. 各客 i には, 時間枠 $[e_i, l_i]$, 要求量 a_i, サービス時間 τ_i が与えられている. 各車両の容量は全て等しい. このベンチマーク問題では, 容量制約と時間枠制約を絶対制約として扱う. また, 解は (車両台数, 移動距離) の辞書式順序で評価される. すなわち, 移動距離が大きくても車両台数が小さい方が良い解となる. 問題のタイプは 6 種類あり, R1, C1, RC1, R2, C2, RC2 に分けられる. タイプ R は客が一様に分布しており, タイプ C は 10 人を 1 グループとする 10 グループがそれぞれ固まって分布している. タイプ RC は両方のタイプの性質を混合したものとなっている. また, タイプ 1 は, デポの時間枠が短く, そのため各客の時間枠を満たしてサービスを行う場合, 比較的多くの車両が必要となる. それに対してタイプ 2 は, デポの時間枠が長く, 少数の車両で全ての客の時間枠を満たしながらサービスを行うことが可能である. 以下では, これまでの最良解

http://www.sintef.no/static/am/opti/projects/top/vrp/bknown.html (2003 年 11 月 21 日)

との比較を行った.

本研究のアルゴリズムで, このベンチマーク問題を解くために, 時間枠コスト関数 $p_i(t)$ と移動時間コスト関数 $q_{ij}(t)$ を

$$p_i(t) = \begin{cases} \alpha_1(e_i - t), & t < e_i \\ 0, & e_i \le t \le l_i \\ \alpha_1(t - l_i), & l_i < t \end{cases}$$

$$q_{ij}(t) = \begin{cases} +\infty, & t < 0.9(\tau_i + t_{ij}) \\ \alpha_2(\tau_i + t_{ij} - t), & 0.9(\tau_i + t_{ij}) \le t < \tau_i + t_{ij} \\ 0, & \tau_i + t_{ij} \le t \end{cases}$$

と定める. ここで, α_1, α_2 はパラメータである. 反復局所探索法の停止条件は 2000 秒とし, 各パラメータの値は $L^{\text{cross}} = 3$, $L^{\text{intra}}_{\text{path}} = 3$, $L^{\text{intra}}_{\text{ins}} = 20$ とした.

計算結果を表 1, 2, 3 に示す. 表 1 と 2 は車両台数をこれまでの最良解と同じにした場合, 表 3 はこれまでの最良解よりも 1 台少ない台数に設定した場合の結果である. 表 1 と 2 では $\alpha_1 = \alpha_2 = 10$, 表 3 では $\alpha_1 = \alpha_2 = 100$ とした. 各表において, P は時間枠からのずれの総和 (すなわち従来の時間枠制約の違反量), Q は移動時間を短縮した量の総和 (すなわち従来の移動時間を違反した量) を表し, 括弧内の数字はそれぞれ, 時間枠を満たさなかった客数, 移動時間を短縮した回数を表す. 容量制約についてはすべて満たしたので省略する. 実行可能解は, 探索中に見つけた実行可能解の総移動距離の最小値を表す. ここで実行可能解というのは, このベンチマーク問題での実行可能解

表 1: タイプ 1 の問題例に対する結果

問題タイプ	d_{sum}	P	Q	目的関数値	実行可能解	最良解	車両台数
R101	*1616.54	0.57(2)	0.12(1)	*1625.52	–	1645.79	19
R102	*1422.78	0.73(2)	1.54(3)	*1445.47	1509.00	1486.12	17
R103	*1174.57	3.23(2)	1.42(1)	*1221.16	–	1292.68	13
R104	1039.78	0	12.16(9)	1161.42	–	1007.24	9
R105	*1372.18	0	0.10(1)	*1373.22	*1377.11	1377.11	14
R106	1257.96	0	0	1257.96	1257.96	1251.98	12
R107	1113.90	0	0	1113.90	1113.90	1104.66	10
R108	966.07	0	0.49(1)	970.98	973.29	960.88	9
R109	1201.65	0	0	1201.65	1201.65	1194.73	11
R110	1169.04	0	0.05(1)	1169.54	1170.18	1118.59	10
R111	1103.12	0	0.57(2)	1108.81	1108.84	1096.72	10
R112	1008.35	0	1.42(2)	1022.53	1031.38	982.14	9
C101	*828.94	0	0	*828.94	*828.94	828.94	10
C102	*828.94	0	0	*828.94	*828.94	828.94	10
C103	*828.06	0	0	*828.06	*828.06	828.06	10
C104	*824.78	0	0	*824.78	*824.78	824.78	10
C105	*828.94	0	0	*828.94	*828.94	828.94	10
C106	*828.94	0	0	*828.94	*828.94	828.94	10
C107	*828.94	0	0	*828.94	*828.94	828.94	10
C108	*828.94	0	0	*828.94	*828.94	828.94	10
C109	*828.94	0	0	*828.94	*828.94	828.94	10
RC101	*1629.99	0.25(1)	5.70(4)	*1689.42	–	1696.94	14
RC102	*1526.18	0.39(1)	0.73(1)	*1537.30	–	1554.75	12
RC103	1302.83	0	0	1302.83	1302.83	1261.67	11
RC104	1135.52	0	0	1135.52	1135.52	1135.48	10
RC105	*1508.79	0	4.15(3)	*1550.30	–	1629.44	13
RC106	*1382.03	0	1.87(2)	*1400.74	–	1424.73	11
RC107	*1212.48	0	0.76(1)	*1220.08	1232.26	1230.48	11
RC108	1147.43	0	0	1147.43	1147.43	1139.82	10

のことである. すなわち時間枠コスト, 移動時間コスト, および容量超過量がすべて 0 の解である. * はこれまでの最良解の総移動距離よりも小さいか等しい値であることを示す. また, 表 3 の $P_{\max}$ と $Q_{\max}$ はそれぞれ, 最大時間枠違反量と最大移動時間短縮量である.

表 1 と 2 より, タイプ C のほとんどの問題例に対して, これまでの最良解と等しい精度の解を得ていることが分かる. タイプ R と RC の計算結果の中には P, Q の値が 0 でないものが多数あるが, それらはほとんどの場合, 1 人の客あたり, あるいは 1 回の移動あたり高々2 であり, デポの時間枠の l_0 が, タイプ R1 は 230, C1 は 1236, RC1 は 240, R2 は 1000, C2 は 3390, RC2 は 960 であることを考慮すれば, 多くの応用において現実には許されると思われる値である. タイプ R1 と RC1 では, 実行可能解が見つからないケースがいくつかあるものの, 移動時間コストが少しかかるが, 総移動距離がこれまでの最良解より小さい解が得られており, これまでの最良解と同等の実行可能解が得られた問題例もある. しかし, タイプ R2 と RC2 の問題例に対しては, これまでの最良解に比べて総移動距離がやや大きい解にとどまる場合が多い. この原因として, タイプ 2 では 1 台の車両で多くの客をサービスするため, 動的計画法の計算時間が大きくなって局所探索法に時間がかかり, そのため探索が十分に行われていないものと思われる. それでもほとんどの問題例に対し実行可能解が得られている.

表 3 に, 利用する車両台数の多いいくつかの問題例に対し, 既知の最小の車両台数よりも 1 台少ない車両台数で探索した結果を示す. この条件で実行可能解が見つかると, 既知の最良解を更新する

表 2: タイプ 2 の問題例に対する結果

問題タイプ	d_{sum}	P	Q	目的関数値	実行可能解	最良解	車両台数
R201	1257.43	0	0	1257.43	1257.43	1252.37	4
R202	1212.23	0	0	1212.23	1212.23	1191.70	3
R203	944.34	0	0	944.34	944.34	939.54	3
R204	876.03	0	0	876.03	876.03	825.52	2
R205	1056.41	0	0	1056.41	1056.41	994.42	3
R206	936.26	0	0	936.26	936.26	906.14	3
R207	897.49	0	0	897.49	897.49	893.33	2
R208	754.66	0	0	754.66	754.66	726.75	2
R209	940.35	0	0	940.35	940.35	909.16	3
R210	969.26	0	0	969.26	969.26	939.34	3
R211	957.60	0	0.67(1)	964.24	970.00	892.71	2
C201	*591.56	0	0	*591.56	*591.56	591.56	3
C202	*591.56	0	0	*591.56	*591.56	591.56	3
C203	600.21	0	0	600.21	600.21	591.17	3
C204	596.55	0	0	596.55	596.55	590.60	3
C205	*588.88	0	0	*588.88	*588.88	588.88	3
C206	*588.49	0	0	*588.49	*588.49	588.49	3
C207	*588.29	0	0	*588.29	*588.29	588.29	3
C208	*588.32	0	0	*588.32	*588.32	588.32	3
RC201	1414.59	0.06(1)	0.77(1)	1422.88	1424.65	1406.91	4
RC202	1369.28	0	2.76(3)	1396.86	–	1367.09	3
RC203	1084.95	0	0	1084.95	1084.95	1049.62	3
RC204	840.59	0	0	840.59	840.59	798.41	3
RC205	1299.99	0	0	1299.99	1299.99	1297.19	4
RC206	1207.61	0.10(1)	0.13(1)	1209.94	1216.40	1146.32	3
RC207	1077.23	0	0	1077.23	1077.23	1061.14	3
RC208	867.58	0	0	867.58	867.58	828.14	3

ことになるが, そのような解は見つかっていない. しかし, 本来の時間枠制約と移動時間制約を若干破っているが, 移動距離は最良解よりも小さい解がいくつかの問題例に対して見つかっている. 特に, R101, R102, R103, RC105 の 4 問に対しては, 違反量 P と Q の値はスケジュール全体から比べれば非常に小さいにも関わらず, これまでの最良解からの改善量は十分に大きく, 本論文が提案した柔軟性の高い定式化の効果の大きさを示しているといえよう. また, Solomon の問題例で解が (車両台数, 移動距離) の辞書式順序で評価されるように, コスト削減において車両台数を減らすことは移動距離を縮めること以上に意味がある. 従って R105, R106, RC103, RC107 のような解を見つけることができたのも重要である. なお, タイプ 2 の問題例では, 既知の最小の車両台数が 2 3 台程度であり, 台数をさらに 1 台減らした解を考えることは非現実的である.

表 3: 車両台数を 1 台減らした計算結果

問題タイプ	d_{sum}	P	Q	目的関数値	Pmax	Qmax
R101	*1636.28	0.30(1)	0	1665.85	0.30	0
R102	*1473.77	0	1.65(2)	1638.70	0	1.25
R103	*1266.35	2.82(1)	1.42(1)	1690.61	2.82	1.42
R105	1469.11	0.70(1)	4.13(3)	1951.51	0.70	2.89
R106	1333.37	0.55(2)	5.13(3)	1906.92	0.54	2.12
RC103	1362.80	2.21(1)	2.73(3)	1857.27	2.21	1.86
RC105	*1625.64	0	7.80(5)	0	0	2.17
RC107	1351.98	0	7.02(8)	2054.04	0	1.58

8 まとめ

本研究では, 時間枠つき配送計画問題の制約条件として, 従来の時間枠制約と容量制約に加えて, あらたに移動時間も考慮制約として考えた. そして移動時間コスト関数を考慮した時間枠つき配送計画問題の解法の1つとして, 動的計画法を用いて各客の最適サービス時刻を決定することを局所探索法に組み込んだ解法を提案した. 代表的なベンチマーク問題に対する計算実験の結果より, 移動で多少無理をすれば車両の台数を減らせる場合があることが確認できた. このことより, 従来より柔軟性の高い配送計画を行うことが可能となった.

今後の課題として, 局所探索法での近傍の探索順序や, より効果的な近傍の開発, 近傍リストの活用, パラメータの自動調整などが挙げられる. また, タブー探索法などの他のメタヒューリスティクスと組合せることによる性能向上なども考えている.

参考文献

[1] M. Desrochers, J. K. Lenstra, M. W. P. Savelsbergh and F. Soumis, "Vehicle Routing with Time Windows: Optimization and Approximation," in *Vehicle Routing: Methods and Studies*, B. L. Golden and A. A. Assad (eds.), North-Holland, Amsterdam, 65–84 (1988).

[2] J. Desrosiers, Y. Dumas, M. M. Solomon, and F. Soumis, "Time Constrained Routing and Scheduling," in *Handbooks in Operations Research and Management Science*, Vol.8. *Network Routing*, M. O. Ball, T. L. Magnanti, C. L. Monma and G. L. Nemhauser (eds.), North-Holland, Amsterdam, 35–139 (1995).

[3] M. R. Garey and D. S. Johnson, *Computers and Intractability*, Freeman (1979).

[4] T. Ibaraki, S. Imahori, M. Kubo, T. Masuda, T. Uno and M. Yagiura, "Effective Local Search Algorithms for Routing and Scheduling Problems with General Time Window Constraints," *Transportation Science*, to appear.

[5] J. Y. Potvin, T. Kervahut, B. L. Garcia and J. M. Rousseau, "The Vehicle Routing Problem with Time Windows. Part 1: Tabu Search," *INFORMS Journal on Computing*, Vol.8, No.2, 158–164 (1996).

[6] M. M. Solomon, "The Vehicle Routing and Scheduling Problems with Time Window Constraints," *Operations Research*, Vol.35, No.2, 254–265 (1987).

[7] M. M. Solomon and J. Desrosiers, "Time Window Constrained Routing and Scheduling Problems," *Transportation Science*, Vol.22, 1–13 (1988).

[8] E. Taillard, P. Badeau, M. Gendreau, F. Guertin and J. Y. Potvin, "A Tabu Search Heuristic for the Vehicle Routing Problem with Soft Time Windows," *Transportation Science*, Vol.31, No.2, 170–186 (1997).