特別研究報告書

オンライン割当て問題に対する劣勾配法

指導教員　　山下信雄　教授

京都大学工学部情報学科
数理工学コース
平成 23 年 4 月入学

池上　哲矢

平成 27 年 1 月 30 日提出

摘要

割当て問題は仕事や商品の最適な配分を決定する問題である. 近年, インターネットの普及により, リアルタイムの意思決定を行うオンライン割当て問題が注目を集めている. このオンライン割当て問題の解法として, 一回学習アルゴリズムが知られている. 一回学習アルゴリズムでは, 初期に到着したデータに基づいた割当て問題の双対問題を一回解き, その解をデータ全体の問題の潜在価格の近似値とする. そして, それ以降に到着するデータに対して, その潜在価格の近似値のみで, 意思決定を行う. 一回学習アルゴリズムでは, 決めた潜在価格の近似値を固定するため, 初期のデータが不十分であれば, 最適とはほど遠い決定を行う可能性がある.

そこで, 本報告書ではこの潜在価格の近似値をデータが到着するごとに更新することを考える. つまり, 初期に到着したデータの情報だけでなく, それ以降に到着したデータの情報も用いて潜在価格の近似値を更新する. 具体的には, 一回学習アルゴリズムによって潜在価格の近似値を求めた後, オンライン劣勾配法でこの潜在価格の近似値を更新するアルゴリズムを提案する. これにより, 初期のデータが不十分であった場合でも潜在価格の近似値が修正され, より最適な意思決定を実現することが期待できる. さらに, 数値実験によって, 提案手法が一回学習アルゴリズムよりも最適な意思決定ができることを実証する.

目次

1 序論

割当て問題とは, 二つの集合が与えられたとき, 片方の集合の要素をもう一方の集合のどの要素に割当てるかを決定する問題である. さらに, 二つの集合の要素間に収益が与えられ, その収益を最大化するような割当てを決定する問題が知られている. その応用として, 仕事の割当てやオークションでの商品の割当てなど様々なモデルが提案されている.

近年, 逐次的にデータが到着する状態で, すべてのデータが揃う前に, オンライン (リアルタイム) で意思決定を行うオンライン割当て問題が注目を集めている [2, 3, 5]. 例として, EC サイトに顧客がアクセスしたときに推薦する商品の決定, 検索履歴から表示する広告の決定などが挙げられる. オンラインでの意思決定が求められる場合では, それ以降に到着する情報なしに意思決定を行わなければならない. このような, 割当てる対象が逐次的に訪れるような問題をオンライン割当て問題という.

本報告書では, 二つの集合 $I = \{1, \cdots, n\}$, $J = \{1, \cdots, m\}$ を考え, I の各要素 i を J の要素 j に割当て, 収益を最大化する次の割当て問題を考える. (具体的な応用例は 2.1 節で紹介する.)

$$
\begin{aligned}
\max \quad & \sum_{i=1}^{n}\sum_{j=1}^{m} r_{ij}x_{ij} \\
\text{s.t.} \quad & \sum_{j=1}^{m} x_{ij} \le c \qquad (i = 1, \cdots, n) \\
& \sum_{i=1}^{n} a_{ij}x_{ij} \le b_j \quad (j = 1, \cdots, m) \\
& x_{ij} \in \{0, 1\}
\end{aligned}
\tag{1}
$$

ただし, x_{ij} は決定変数であり, i に j を割当てるときは 1, そうでないときは 0 をとる. さらに定数 r_{ij} は i に j を割当てたときの収益である. つまり, 目的関数は総収益をあらわす. また, c は各 i に割当てられる j の上限である. a_{ij} は i に j を割当てたときのコストであり, b_j は j に関するコストの上限をあらわす. 一番目の制約式は, 各 i に対する割当てる j の数の上限をあらわす. 二番目の制約式は, 各 j に対するコストの上限をあらわす. 三番目の制約式より, 決定変数が 0 または 1 であるため, この問題は $0-1$ 整数計画問題である.

オンライン割当て問題では, 次の状態のもとで割当て問題 (1) に対する最適な意思決定を求める. まず, 割当てられる集合 J と, それに対応するコストの上限 b_j は事前に定められているものとする. $t = 1, 2, \cdots$ としたとき, データ $(r_{tj}, a_{tj})(j = 1, \cdots, m)$ が逐次的に与えられるとし, それまでに得られた $i = 1, \cdots, t-1$ から $x_{tj}(j = 1, \cdots, m)$ を決定する. すなわ

ち, $t+1$ 番目以降の情報を用いることなく t 番目の x_{tj} を決定しなければならない. ただし, I の要素数 n は事前にわかっているものとする.

割当て問題 (1) の解法として, バッチ最適化が知られている. バッチ最適化は, 得られたすべての情報を含んだ目的関数から最適解を求める. 一般的なバッチ最適化のアルゴリズムとして, 分枝限定法などが挙げられる [6]. オンライン割当て問題においては, それまでに到着した情報を用いて, バッチ最適化によって意思決定することが考えられる. しかし, データが増えるに従って, 問題の規模が大きくなり, リアルタイムでの意思決定は難しくなる.

そこで, Devanur ら [5] は, 初期に到着したデータのみを用いて構成された双対問題を利用して意思決定する手法を提案した. 初期に到着したデータの割合を $\epsilon \in [0,1]$ であらわす. この手法ではまず, $t \leq \epsilon n$ までのデータのみの問題を考え, その問題を線形緩和した線形計画問題の双対問題の解を求める. その解が二番目の制約の潜在価格となることを利用して, $t > \epsilon n$ の x_{ij} の意思決定を行う. 以下では, 上記のような双対問題の解を求めることを学習と呼び, この手法を一回学習アルゴリズムと呼ぶ. また, ϵ は全データから学習に用いるデータの割合をあらわし, 以降はこれを学習率と呼ぶ. 簡単のため ϵn は整数であるとする. 適当な仮定のもと一回学習アルゴリズムは, オフライン最適解に近づくことが保証されている [5]. 一回学習アルゴリズムで得られる潜在価格は, 一度決定すると更新を行わないため, 以降に到着するデータの情報を含んでいない. したがって, 訓練データが小さいとき, すなわち ϵ が小さいとき, 良い学習ができず, 最適な意思決定とはほど遠い決定を行ってしまう場合がある. そこで Agrawal ら [1] は, この学習を $t = \epsilon n, 2\epsilon n, 4\epsilon n, \cdots$ の複数回行うことにより, さらにオフライン最適解に良い精度で近づくことを示した. しかし, この手法ではデータが増えるほどその問題の次元は大きくなり, リアルタイムな意思決定が難しくなる.

本報告書では, 訓練データ以降の情報も利用して潜在価格を更新し, さらにリアルタイムでの意思決定を行う手法を提案する. 提案手法は, 一回学習アルゴリズムの後, Agrawal らの手法 [1] を用いるのではなく, 劣勾配法を用いることにより, 訓練データ以降の情報を取り入れ, 潜在価格を更新し, より収益を大きくすることを考える. 劣勾配法を用いることで, 学習に用いたデータ以降の情報も潜在価格に取り入れることができ, リアルタイムでの良い意思決定を実現する. また, アルゴリズムをより実践的なものとするため, アルゴリズム中での計算量の削減を実現する工夫も提案する.

本報告書の構成を述べる. 2 節では, オンライン割当て問題の一例であるオンライン推薦商品最適化問題を紹介し, その問題に対する既存手法を説明する. 3 節では, 劣勾配の導出について述べ, それを用いてオンライン割当て問題に対する最適な意思決定を求める手法を提案する. 4 節では, 提案手法の効率的な実装について述べる. 5 節では, 数値実験の結果を報告し, 提案手法の有用性を示す. 最後に 6 節で, 結論を述べる.

本報告書における下添字は, ベクトルの成分をあらわし, 上添字はアルゴリズム中での反復

をあらわす. また $\boldsymbol{e}$ はすべての要素が 1 であるようなベクトルをあらわす. さらに, $[\cdot]_+$ は, max 関数つまり $[a]_+ = \max\{0, a\}$ をあらわし, a がベクトルのときは, 各成分にこれを適用したものとする. 凸関数 $f : \mathbb{R}^n \to \mathbb{R}$ に対して, $f(\boldsymbol{y}) - f(\boldsymbol{x}) \geq \langle \xi,\ \boldsymbol{y} - \boldsymbol{x} \rangle (\boldsymbol{y} \in \mathbb{R}^n)$ を満たすベクトル $\xi \in \mathbb{R}^n$ を凸関数 f の点 $\boldsymbol{x}$ における劣勾配と呼ぶ. また, 劣勾配 ξ 全体の集合を $\partial f(\boldsymbol{x})$ とあらわす.

2 準備

本節では, まずオンライン割当て問題の一例であるオンライン推薦商品最適化問題を紹介する. 次に, その問題に対する Lagrange 双対問題を導出し, オンライン割当て問題の解法の一つである一回学習アルゴリズムを紹介する. 最後に, 提案手法の基盤となるオンライン劣勾配法とその性質について説明する.

2.1 オンライン推薦商品最適化問題

顧客のニーズに合わせた商品を推薦するシステムは現在広く用いられている [8]. この推薦する商品を決定し, 利益を最大化する問題を推薦商品最適化問題と呼ぶ. 推薦商品最適化問題は, 広く研究され様々な決定方法が提案されており, オンライン割当て問題 (1) の型へ定式化可能である. したがって, 以下では, 説明を容易にするため, オンライン割当て問題の一例であるオンライン推薦商品最適化を考える.

改めて, このオンライン推薦商品最適化問題を以下のように定式化する.

$$
\begin{aligned}
\max \quad & \sum_{i=1}^{n} \sum_{j=1}^{m} r_{ij} x_{ij} \\
\text{s.t.} \quad & \sum_{j=1}^{m} x_{ij} \leq c \qquad (i = 1, \cdots, n) \\
& \sum_{i=1}^{n} a_{ij} x_{ij} \leq b_j \quad (j = 1, \cdots, m) \\
& x_{ij} \in \{0, 1\}
\end{aligned}
\tag{2}
$$

ただし, x_{ij} は決定変数であり, 顧客 i に商品 j を推薦するときは 1, そうでないときは 0 をとる. さらに定数 r_{ij} は顧客 i に商品 j を推薦したときの利益である. つまり, 目的関数は総利益をあらわす. また, c は各顧客 i に推薦される商品数の上限である. a_{ij} は顧客 i に商品 j を推薦したときのコストであり, b_j は商品 j に関するコストの上限をあらわす. 一番目の制約式は, 各顧客 i に対する推薦する商品数の上限をあらわす. 二番目の制約式は, 各商品 j に対するコストの上限をあらわす. 三番目の制約式より, 決定変数が 0 または 1 であるため, この

問題は $0-1$ 整数計画問題である.

以降の節では, 簡単のため, t 番目の顧客の情報を得ることを到着したと表現する. 逐次的に顧客が到着するため, 到着した顧客に対してそれ以降に到着する顧客の情報なしに到着した顧客に商品を推薦しなければならない.

問題 (2) において, 二番目の制約式がなければ, この問題は顧客ごとの問題に分解することができ, 簡単に解ける. そこで, 問題を顧客ごとに分解するため, 二番目の制約式に関してのLagrange 緩和問題を考える.

2.2 Lagrange 双対問題の導出

本小節では, 割当て問題 (2) の双対問題を考える.

まず, 割当て問題 (2) と等価な次の問題を考える.

$$\begin{aligned} \min \quad & -\sum_{i=1}^{n}\sum_{j=1}^{m} r_{ij}x_{ij} \\ \text{s.t.} \quad & \sum_{j=1}^{m} x_{ij} \le c \qquad (i=1,\cdots,n) \\ & \sum_{i=1}^{n} a_{ij}x_{ij} \le b_j \quad (j=1,\cdots,m) \\ & x_{ij} \in \{0,1\} \end{aligned} \tag{3}$$

この問題 (3) において, Lagrange 乗数を $\boldsymbol{\lambda} \ge \mathbf{0}$ として, 二番目の制約式を緩和したLagrange 緩和問題 L_0 を考える. この問題は, 以下のように書ける.

$$\begin{aligned} \min \quad & L_0(\boldsymbol{x},\boldsymbol{\lambda}) = -\sum_{i=1}^{n}\sum_{j=1}^{m} r_{ij}x_{ij} + \sum_{j=1}^{m}\lambda_j\left(\sum_{i=1}^{n} a_{ij}x_{ij} - b_j\right) \\ \text{s.t.} \quad & \boldsymbol{x} \in \boldsymbol{X} \end{aligned} \tag{4}$$

ただし, $\boldsymbol{X}$ は 問題 (3) の一番目の制約式と三番目の制約式を満たす $\boldsymbol{x}$ の集合, すなわち

$$\boldsymbol{X} = \left\{ \boldsymbol{x} \in \mathbb{R}^{n\times m} \;\middle|\; \begin{array}{ll} \sum_{j=1}^{m} x_{ij} \le c & (i=1,\cdots,n), \\ x_{ij} \in \{0,1\} & (i=1,\cdots,n,\ j=1,\cdots,m) \end{array} \right\}$$

である.

ここで, $\boldsymbol{X}$ は有界閉集合であるから, 問題 (4) は最小値を持つ. このときの値を $\omega_0(\boldsymbol{\lambda})$ とすると,

$$\omega_0(\boldsymbol{\lambda}) = \min_{\boldsymbol{x}\in\boldsymbol{X}} L_0(\boldsymbol{x},\boldsymbol{\lambda})$$

と書ける.

この緩和が最も引き締まるようなラグランジュ乗数の値を求める問題を Lagrange 双対問題と呼ぶ. よって, 問題 (3) の Lagrange 双対問題は以下のようになる.

$$\begin{array}{rl} \max & \omega_0(\boldsymbol{\lambda}) \\ \text{s.t.} & \boldsymbol{\lambda} \geq 0 \end{array}$$

ここで, Lagrange 双対問題の目的関数 ω_0 は, 以下のように変形できる.

$$\begin{aligned} \omega_0(\boldsymbol{\lambda}) &= \min_{\boldsymbol{x}\in\boldsymbol{X}} L_0(\boldsymbol{x},\boldsymbol{\lambda}) \\ &= \min_{\boldsymbol{x}\in\boldsymbol{X}} \left\{ \sum_{i=1}^{n}\sum_{j=1}^{m} -r_{ij}x_{ij} + \sum_{j=1}^{m}\lambda_j \left(\sum_{i=1}^{n} a_{ij}x_{ij} - b_j \right) \right\} \\ &= \min_{\boldsymbol{x}\in\boldsymbol{X}} \left\{ \sum_{i=1}^{n}\sum_{j=1}^{m} (-r_{ij} + a_{ij}\lambda_j)x_{ij} - \sum_{j=1}^{m}\lambda_j b_j \right\} \end{aligned}$$

このとき, 上式を $i = 1, \cdots, n$ のそれぞれに関して分解する. このために, $\boldsymbol{x}_t$, $\boldsymbol{X}_t$ を以下のように定める.

$$\begin{aligned} \boldsymbol{x}_t &= (x_{t1},\ x_{t2},\ \cdots, x_{tm}), \\ \boldsymbol{X}_t &= \left\{ \boldsymbol{x}_t \in \mathbb{R}^{1\times m} \;\middle|\; \begin{array}{ll} \sum_{j=1}^{m} x_{tj} \leq c, & \\ x_{tj} \in \{0,1\} & (j = 1, \cdots, m) \end{array} \right\} \end{aligned}$$

以上より,

$$\begin{aligned} \omega_0(\boldsymbol{\lambda}) = & \min_{\boldsymbol{x}_1\in\boldsymbol{X}_1} \left\{ \sum_{j=1}^{m} (-r_{1j} + a_{1j}\lambda_j)x_{1j} \right\} \\ & + \min_{\boldsymbol{x}_2\in\boldsymbol{X}_2} \left\{ \sum_{j=1}^{m} (-r_{2j} + a_{2j}\lambda_j)x_{2j} \right\} \\ & + \quad \vdots \\ & + \min_{\boldsymbol{x}_n\in\boldsymbol{X}_n} \left\{ \sum_{j=1}^{m} (-r_{nj} + a_{nj}\lambda_j)x_{nj} \right\} \\ & - \sum_{j=1}^{m}\lambda_j b_j \end{aligned}$$

となり, Lagrange 双対問題はユーザごとの最小化問題を解くことによって計算できる.

ここで, $\phi^i(\boldsymbol{\lambda}) = \min_{\boldsymbol{x}_i \in \boldsymbol{X}_i} \left\{ \sum_{j=1}^{m} (-r_{ij} + a_{ij}\lambda_j) x_{ij} \right\}$ とおくと, 目的関数は

$$\omega_0(\boldsymbol{\lambda}) = \sum_{i=1}^{n} \phi^i(\boldsymbol{\lambda}) - \sum_{j=1}^{m} \lambda_j b_j$$

とあらわすことができる.

以上より, 割当て問題 (3) に対する Lagrange 双対問題は以下のようにあらわされる.

$$\begin{aligned} \max \quad & \sum_{i=1}^{n} \phi^i(\boldsymbol{\lambda}) - \sum_{j=1}^{m} \lambda_j b_j \\ \text{s.t.} \quad & \boldsymbol{\lambda} \geq \mathbf{0} \end{aligned}$$

2.3 一回学習アルゴリズム

一回学習アルゴリズムは, オンラインで到着する n 人のうち, 最初の ϵn 人の情報に基づいた双対問題の解が潜在価格となることを利用して問題 (2) を解く.

まず, 最初の ϵn 人が到着したときの割当て問題 (2) の部分問題は以下のようになる.

$$\begin{aligned} \max \quad & \sum_{i=1}^{\epsilon n} \sum_{j=1}^{m} r_{ij} x_{ij} & & \\ \text{s.t.} \quad & \sum_{j=1}^{m} x_{ij} \leq c & & (i = 1, \cdots, \epsilon n) \\ & \sum_{i=1}^{\epsilon n} a_{ij} x_{ij} \leq \epsilon b_j & & (j = 1, \cdots, m) \\ & x_{ij} \in \{0, 1\} & & \end{aligned} \tag{5}$$

二番目の制約式は ϵn 人分の上限に抑えるため各 j に対して上限を ϵb_j としている.

ここで, この問題 (5) を線形緩和し, 線形計画問題として扱う. すなわち, 三番目の制約式を $x_{ij} \in [0, 1]$ とする. このとき, この緩和した線形計画問題に対する双対問題は

$$\begin{aligned} \min \quad & c \sum_{i=1}^{\epsilon n} \alpha_i + \epsilon \sum_{j=1}^{m} b_j p_j & & \\ \text{s.t.} \quad & \alpha_i + a_{ij} p_j \geq u_{ij} & & (i = 1, \cdots, \epsilon n,\ j = 1, \cdots, m) \\ & \alpha_i \geq 0 & & (i = 1, \cdots, \epsilon n) \\ & p_j \geq 0 & & (j = 1, \cdots, m) \end{aligned} \tag{6}$$

となる. ただし, この問題における決定変数は $\alpha_i (i = 1, \cdots, \epsilon n)$ と $p_j (j = 1, \cdots, m)$ である.

ここで, この双対問題 (6) の最適解は $\alpha_i = \max_j(u_{ij} - a_{ij}p_j)$ となることから, p_j のみの関数としてあらわすことができる. すなわち,

$$\begin{array}{ll} \min & c\displaystyle\sum_{i=1}^{\epsilon n}\max_j(u_{ij} - a_{ij}p_j) + \sum_{j=1}^{m} b_j p_j \\ \text{s.t.} & p_j \geq 0 \ \ (j = 1, \cdots, m) \end{array}$$

となり, この p_j を潜在価格として学習する. これを用いて $\epsilon n + 1$ 以降に到着した人に対し, 意思決定を行う.

以下に, 問題 (2) に対する一回学習アルゴリズムを示す.

一回学習アルゴリズム

Step 1 : $t = 1, \cdots, \epsilon n$ に対して, x_{tj} を問題の制約条件を満たすよう適当に定める.
Step 2 : 双対問題 (6) の解 $\bar{\lambda}$ を求め, $\boldsymbol{\lambda}^{\epsilon n} = \bar{\lambda}$ とする.
Step 3 : $t = \epsilon n + 1, \cdots, n$ に対して以下の操作を行う.
$-r_{tj} + a_{tj}\lambda_j^{\epsilon n}$ の値の下位 c 個の j について $x_{tj} = 1$ とする. そうでない j について $x_{tj} = 0$ とする. ただし, x_{tj} が実行可能でないときは, 実行可能解に変換する.

Step 3 での x_{tj} の決定について説明する. 前小節より, $t = \epsilon n + 1, \cdots, n$ 番目の顧客に対する Lagrange 双対問題の目的関数は $\phi^t(\boldsymbol{\lambda})$ である. この $\phi^t(\boldsymbol{\lambda})$ に含まれる $\boldsymbol{x}_t$ に関する最小化問題の解が Step 3 で決定される $\boldsymbol{x}_t$ となる. このように選ぶことで, 利益だけでなくコストも考慮したような意思決定が実現される.

次に Step 3 での, x_{tj} の実行可能解への変換方法を紹介する. 自然な実行可能解への変更方法として, 係数の小さい順に選ぶ方法が挙げられる. すなわち, アルゴリズム中で $-r_{tj}+a_{tj}\lambda_j$ の値の下位 c 個の j について $x_{tj} = 1$ としたとき, ある商品 j に関して, 制約式を満たしていなければ下位 $c+1, c+2, \cdots$ 番目の j と入れ替えて実行可能になるまでそれを繰り返す. これにより, コストの上限を超えてしまった商品に対してその次に $-r_{tj} + a_{tj}\lambda_j^{\epsilon n}$ の値が小さくなる商品を推薦することができる. 5 節の数値実験においても, 実行可能解への変換方法としてこの手法を用いる.

一回学習アルゴリズムは問題の全体を解くのではなく, ϵn 人に制限した双対問題を解くので計算量はもとの問題を解くより少なくて済む. さらに, 一回双対問題を解き, 潜在価格を学習すれば以降は解かなくてよいので大規模な問題も解くことができる.

しかし, 学習割合が少なく, 十分なデータが得られなかったり, 訓練データが極端な情報を持つデータとなった場合, 良い学習ができず, 以降の推薦が適切でなくなる場合がある.

2.4　オンライン劣勾配法

本小節では, 一般的なオンライン劣勾配法のアルゴリズムとそれに関する性質を示す.

まず, 目的関数が T 個の関数の和であらわされるような以下の問題を考える.

$$\begin{aligned} \min \quad & \sum_{t=1}^{T} \psi^t(\boldsymbol{\lambda}) \\ \text{s.t.} \quad & \boldsymbol{\lambda} \geq \mathbf{0} \end{aligned}$$

ここで $\psi^t : \mathbb{R}^n \to \mathbb{R}$ は凸関数とする. ただし, 微分可能性は特に仮定しない.

この問題に対する一般的なオンライン劣勾配法のアルゴリズムを以下に示す.

オンライン劣勾配法

Step 1 : 初期値 $\boldsymbol{\lambda}^0$ を適当に定める.

Step 2 : $t = 1,\ 2,\ \cdots$ に対して以下の操作を行う.

(a)　ステップ幅 $\gamma_t > 0$ を定める. $\psi^t(\boldsymbol{\lambda})$ の劣勾配 $\eta^t \in \partial\psi^t$ を求める.

(b)　$\boldsymbol{\lambda}^{t+1}$ を以下のように更新する.

$$\boldsymbol{\lambda}^{t+1} = [\boldsymbol{\lambda}^t - \gamma_t \eta^t]_+$$

劣勾配法は, 微分が不可能な関数に対しても適用できる勾配法であり, これにより顧客が到着する度に, その情報から潜在価格を更新することができる.

劣勾配法のステップ幅はさまざまなものが知られており, ステップ幅 $\gamma^t = const$(定数) とするものや $\gamma^t = \frac{a}{\sqrt{t}+b}$(ただし a, b は定数) ととる方法が知られている.

また, このとき, 劣勾配法の最適値に関して, 目的関数が凸であるとき, オンライン劣勾配法で得られた点列によって得られる目的関数の合計とバッチ最適化における最適値との差が $\mathrm{O}(\sqrt{T})$ 以下となることが知られている [9]. これは, 一回の反復あたりの上記の差が小さくなることを意味している.

3　オンライン割当て問題に対する劣勾配法

本節では, 前節で紹介した Lagrange 双対問題から提案手法で用いる劣勾配を導出し, 提案手法について述べる.

3.1 Lagrange 双対問題と劣勾配

2.2 節で導出した Lagrange 双対問題の目的関数は凹関数であるから, これは凹関数の最大化問題である. そこで, 等価な問題として, 次の凸計画問題を考える.

$$\begin{aligned} \min \quad & -\sum_{i=1}^{n} \phi^i(\boldsymbol{\lambda}) + \sum_{j=1}^{m} \lambda_j b_j \\ \text{s.t.} \quad & \boldsymbol{\lambda} \geq \mathbf{0} \end{aligned}$$

この凸計画の最小化問題に対してオンライン劣勾配法を適用する.

この問題に対して直接的にオンライン劣勾配法を適用するため, $t = 1, 2, \cdots, n$ までは $-\phi^t$ の劣勾配を用い, $t = n+1$ のときに最後の項 $\sum_{j=1}^{m} \lambda_j b_j$ に対して, $\boldsymbol{\lambda}$ を更新することになる. ここで, オフラインでバッチ最適化を用いて最適解を求める際は $\sum_{j=1}^{m} \lambda_j b_j$ の項を一度に評価できるが, 上記のようにこの項を最後にまとめて評価してしまうと, 最後になってはじめて b_j の情報が与えられるため, 途中の正しい値が得られない. 言い換えれば, ステップの途中でコストの上限である b_j が考慮されない. そこで, 新たに $\frac{\sum_{j=1}^{m} \lambda_j b_j}{n}$ の項を含んだ関数 ψ^t を次のように定める.

$$\psi^t(\boldsymbol{\lambda}) = -\phi^t(\boldsymbol{\lambda}) + \frac{\sum_{j=1}^{m} \lambda_j b_j}{n}$$

このとき上述の問題は

$$\begin{aligned} \min \quad & \sum_{i=1}^{n} \psi^i(\boldsymbol{\lambda}) \\ \text{s.t.} \quad & \boldsymbol{\lambda} \geq \mathbf{0} \end{aligned}$$

と等価である. この問題にオンライン劣勾配法を適用すれば, $\psi^t(\boldsymbol{\lambda})$ はコスト b_j の情報を得ているため, より適切な潜在価格が得られるはずである.

以下では, $\psi^t(\boldsymbol{\lambda})$ の劣勾配 $\boldsymbol{\eta}^t$ を求める. 関数 ψ^t は, 定め方から $\boldsymbol{\lambda}$ に関して凸関数である. ここで, $\phi^t(\boldsymbol{\lambda})$ に含まれる $\boldsymbol{x}_t$ に関する最小化問題の解を $\bar{\boldsymbol{x}}_t$ とすると, $-\phi^t(\boldsymbol{\lambda})$ の劣勾配の第 j 成分は $-a_{tj}\bar{x}_{tj}$ となる [7]. また, 第二項は $\boldsymbol{\lambda}$ に関して微分可能である. よって, $\psi^t(\boldsymbol{\lambda})$ の劣勾配 $\boldsymbol{\eta}^t$ は

$$\boldsymbol{\eta}^t = \begin{bmatrix} -a_{t1}\bar{x}_{t1} + \frac{1}{n}b_1 \\ -a_{t2}\bar{x}_{t2} + \frac{1}{n}b_2 \\ \vdots \\ -a_{tm}\bar{x}_{tm} + \frac{1}{n}b_m \end{bmatrix}$$

である.

この劣勾配の計算は $\mathrm{O}(m)$ となる. 商品数が非常に大きい問題では, この計算が計算時間のボトルネックとなる. ただし, 特別な場合においては, 劣勾配のすべての要素を毎回評価する必要がなく, 計算量を抑えることができる. 詳細は 5.2 節で述べる.

3.2 提案手法

本小節では, 前節までの結果を用いて, 一回学習アルゴリズムとオンライン劣勾配法を組み合わせたアルゴリズムを提案する.

提案手法

Step 1 : $t = 1, \cdots, \epsilon n$ に対して, x_{tj} を問題の制約条件を満たすよう適当に定める.
Step 2 : 双対問題 (6) の解 $\bar{\lambda}$ を求め, $\boldsymbol{\lambda}^{\epsilon n} = \bar{\lambda}$ とする.
Step 3 : $t = \epsilon n + 1, \cdots, n$ に対して以下の操作を行う.

(a) $-r_{tj} + a_{tj}\lambda_j^t$ の値の下位 c 個の j について $x_{tj} = 1$ とする. そうでない j について $x_{tj} = 0$ とする. ただし, x_{tj} が実行可能でないときは, 適当な手法を用いて実行可能解に変換する.
(b) ステップ幅 γ_t を定める. $\psi^t(\boldsymbol{\lambda})$ の劣勾配 $\eta^t \in \partial\psi^t$ を求める.
(c) $\boldsymbol{\lambda}^{t+1}$ を以下のように更新する.

$$\boldsymbol{\lambda}^{t+1} = [\boldsymbol{\lambda}^t - \gamma_t \eta^t]_+ = \begin{bmatrix} \max\{0, \lambda_1^t - \gamma_t \eta_1^t\} \\ \max\{0, \lambda_2^t - \gamma_t \eta_2^t\} \\ \vdots \\ \max\{0, \lambda_m^t - \gamma_t \eta_m^t\} \end{bmatrix}$$

このアルゴリズムでは, 一回学習を行った後で到着したデータに対してオンライン劣勾配法を用いることで, $\boldsymbol{\lambda}$ を逐次的に更新している. これにより十分な学習ができなかった場合でも, 適切な $\boldsymbol{\lambda}$ が得られると考えられる.

Step 1 では, はじめの ϵn 人のデータから潜在価格を決定するため, それが求められるまでは適当な手法で x_{tj} を決定する必要があり, 様々な決め方が考えられる. 最も簡単な手法はランダムに推薦商品を決定し, それがもとの問題の制約条件を満たしていれば, それを推薦し, 満たしていなければ実行可能解へと推薦する手法である. しかしながら, 推薦商品をランダムに決定してしまうのは最適な決定とほど遠い可能性が高い. そこで, 学習後に用いられるオンライン劣勾配法を適当な初期値からはじめて学習前にも適用することを考える. これにより,

$\boldsymbol{\lambda}$ の初期値によって最適でない推薦が行われるかもしれないが, ある程度利益とコストを考慮した推薦が可能となる.

4 実装

本節では, アルゴリズムの実装において, 計算量の削減を行い, 高速化を実現する手法を示す. 計算の負担が最も大きい部分は主に, 係数の下位 c 個を選ぶ操作と $\boldsymbol{\lambda}$ の更新である. 大規模な問題, すなわち n, m が数百万といったような規模の問題を考えるとき, この計算量を削減することは重要である.

4.1 中央値アルゴリズムを用いた割当て法

提案アルゴリズム Step 3(a) において, $-r_{tj} + a_{tj}\lambda_j$ の下位 c 個を選ぶ際, 商品の数だけ項を選ばなければならない. 下位 c 個を選ぶとき, ヒープソートなどでソートを行うとその計算量は $\mathrm{O}(m\log m)$ となる. しかし, ここでは順番に関係なく下位 c 個を選択できればよい. そこで, 中央値アルゴリズムを用いれば, 計算量は $\mathrm{O}(m)$ となる. 以下では, 中央値アルゴリズムを用いた手法を提案する.

まず, 選択問題を考える. 選択問題とは, 入力として自然数 n, 実数 $z_1, \cdots, z_n$, および整数 $k \in \{1, \cdots, n\}$ が与えられたとき, $z_1, \cdots, z_n$ の中で k 番目に小さい数を求める問題である.

この選択問題は, 重み付き中央値アルゴリズムの特殊な場合としてみなすことができる [4]. 以下に中央値の定義を示す.

定義 4.1 n を自然数, $z_1, \cdots, z_n$ を実数, $w_1, \cdots, w_n$ を非負実数とし, 実数 W を $0 < W < \sum_{i=1}^{n} w_i$ とする. このとき,

$$\sum_{i:z_i<z^*} w_i < W \leq \sum_{i:z_i\leq z^*} w_i$$

を満たす唯一の数 z^* を $(w_1, \cdots, w_n; W)$ で重み付けしたときの $(z_1, \cdots, z_n)$ の重み付き中央値という.

さらに, $w_i = 1 (i = 1, \cdots, n)$ とし, $W = \lceil \frac{n}{2} \rceil$ のときは中央値あるいは重みなし中央値という.

このとき, 入力として自然数 n, 実数 $z_1, \cdots, z_n$, 非負の実数 $w_1, \cdots, w_n$, および $0 < W < \sum_{i=1}^{n} w_i$ なる実数 W が与えられたとき, $(w_1, \cdots, w_n; W)$ で重み付けした $(z_1, \cdots, z_n)$ の重み付き中央値を返すようなアルゴリズムを中央値アルゴリズムという. 以下に, このアルゴリズムを示す.

中央値アルゴリズム

Step 1 : リスト $z_1, \cdots, z_n$ を 5 個ずつのブロック (最後のブロックのみは 5 個未満でも良い) に分割する. 各ブロックで中央値を求め, これら $\lceil \frac{n}{5} \rceil$ 個の中央値のリストを M とする.
Step 2 : M の中央値を求める. それを z_m とする.
Step 3 : 各要素を z_m と比較する. 一般性を失うことなく $i = 1, \cdots, k$ に対して $z_i < z_m$ で, $i = k+1, \cdots, l$ に対して, $z_i = z_m$ で, 残りの $i = l+1, \cdots, n$ に対して $z_i > z_m$ であると仮定できる.
Step 4 :

(a) $\sum_{i:z_i<z^*} w_i < W \leq \sum_{i:z_i \leq z^*} w_i$ なら, $z^* = z_m$ を返して終了する.
(b) $\sum_{i=1}^{l} w_i < W$ なら, $(w_{l+1}, \cdots, w_n; W - \sum_{i=1}^{l} w_i)$ で重み付けした $z_{l+1}, \cdots, z_n$ の重み付け中央値を再帰的に求めてそれを返して終了する.
(c) $\sum_{i=1}^{k} w_i \geq W$ なら, $(w_1, \cdots, w_k; W)$ で重み付けした $z_1, \cdots, z_k$ の重み付け中央値を再帰的に求めてそれを返して終了する.

この重み付き中央値アルゴリズムの計算量は $\mathrm{O}(n)$ である.

さらに, $w_i = 1 (i = 1, \cdots, n)$, $W = k$ とすれば, 重み付き中央値アルゴリズムは選択問題の解を与えることから, 選択問題は $\mathrm{O}(n)$ 時間で解ける.

下位 k 個を得るには, 選択問題で得られた値以下の $k-1$ 個の要素を取り出せばよく, これは最悪の場合でも $\mathrm{O}(n)$ であるから結局, 全体の計算量は $\mathrm{O}(n)$ となる.

つまり, この選択問題において $z_j = -r_{tj} + a_{tj}\lambda_j$, $k = c$ として中央値アルゴリズムを用いれば, $z_j = -r_{tj} + a_{tj}\lambda_j$ の下位 c 個を $\mathrm{O}(m)$ の計算量で求められる. これにより, ソートを用いる手法に比べて, 計算量を抑えることができる.

4.2 スパース性を利用した $\boldsymbol{\lambda}$ の更新

提案アルゴリズム Step 3(c) において, 単純にアルゴリズムを実装すると, m 次元ベクトル $\boldsymbol{\lambda}$ が n 回更新されるため, その計算量は $\mathrm{O}(nm)$ となる. 商品数 m が非常に大きくなるとき, 劣勾配の計算や $\boldsymbol{\lambda}$ の更新にかかる計算時間が実用時間でなくなることがある. そこで今 t 番目に到着した顧客の各商品に対する利益 $r_{tj} (j = 1, \cdots, m)$ はほとんどの成分が 0 であり, 非零成分が m に比べて少ない場合を考え, より効率的な手法を与える. 利益 r_{tj} が 0 の成分は推薦する商品の候補として無視できるから, 成分が非零の商品 j に関してのみ, $\boldsymbol{\lambda}$ を更新すれ

ばよい. 更新しなかった成分に関しては, それ以降に来た客の推薦する商品の候補になったときに更新すればよい.

以下で上述の方法を実現するアルゴリズムを示す.

$\boldsymbol{\lambda}$ の更新

Step 3-1 : $t = 1, \cdots, n$ に対して以下の操作を行う. $\Gamma = 0$, $z_j = 0 (j = 1, \cdots, m)$ とする.
Step 3-2 : $\Gamma = \Gamma + \gamma^t$
Step 3-3 : $r_{tj} \neq 0$ の j に対して以下の操作を行う.

(a) $\lambda_j^t = \max\{0, \lambda_j^t - (\Gamma - z_j)\frac{b_j}{n}\}$ を更新する. $-r_{tj} + a_{tj}\lambda_j$ の値が小さくなる下位 c 個の j について $x_{tj} = 1$ とする. そうでない j について $x_{tj} = 0$ とする. ただし, x_{tj} が実行可能でないときは, 適当な手法を用いて実行可能解に変換する. $z_j = \Gamma$ とする.

(a) $x_{tj} = 1$ となった j に対して, $\lambda_j^t = \max\{0, \lambda_j^t - \gamma_t \eta_j^t\}$ とする.

アルゴリズム中で Γ はそれまでに蓄積されたステップ幅の合計, z_j は j 番目の商品に関して推薦すれば更新, 推薦しなければ更新しないことで推薦してきた分のステップ幅の蓄積を表す.

簡単に計算量を見積もるため, 各顧客ごとの商品に対する非零成分の最大個数を $m_{n0} (<< m)$ とする.

一回の $\boldsymbol{\lambda}$ の更新で更新されるのは m_{n0} 個であり, それが n 回行われるから $\boldsymbol{\lambda}$ の更新にかかる計算量は $\mathrm{O}(nm_{nonzero})$ である.

5 数値実験

本節では, 前節で紹介したアルゴリズムの有用性を示す数値実験結果を示す. 実験は CPU が Intel(R)Core(TM)i7-4650U 1.70GHz, プログラムの実装には MATLAB(R2013a) を用いた.

本実験では, 顧客数 n は 5000 人, 商品数 m は 100 個, 推薦商品数 c は 5 個とする. また, コスト a_{ij} は各 j ごとに平均を区間 $[0, 1]$ から一様に定めたもの, 標準偏差が 0.3 の正規分布に従う乱数とし, コスト上限 b_j は区間 $[10, 200]$ から一様に定める. リターン r_{ij} は各 j ごとに平均を a_{ij} と同じもの, 標準偏差が 0.5 の正規分布に従う乱数とする.

提案手法の Step 1 で, 一回学習までは制約条件を満たすよう決定変数を適当に定めるとし

ているが, 数値実験ではオンライン劣勾配法を適用している.

数値実験では, まず, ステップ幅による学習率 $\epsilon = 0$ としたときの提案手法の目的関数値の振舞いを調べる. 次に一回学習アルゴリズムと提案手法の学習率による比較を行う. 最後に, 各 ϵ に対する提案手法の初期点 $\boldsymbol{\lambda}^0$ による影響を調べる.

手法を比較するために, t 番目までに到着した客の目的関数の累積値 $R_{sum}(t)$ を以下のように定義する.

$$R_{sum}(t) = \sum_{i=1}^{t} \sum_{j=1}^{m} r_{ij} x_{ij}$$

5.1 ステップ幅の違いによる振舞い

本小節では, ステップ幅の違いによる学習率 $\epsilon = 0$ としたときの提案手法の振舞いを調べる. そのために, ステップ幅 η はある定数として固定する. また, 初期値 $\boldsymbol{\lambda}^0 = \mathbf{0}$ とする.

図 1 は, ステップ幅を四つの値, $\eta = 0.001, 0.01, 0.1, 1$ と固定して, 問題に対し提案手法を適用したときの R_{sum} である. 図 1 は, 横軸がアルゴリズム中でのステップ数, すなわち到着した客数をあらわし, 縦軸が目的関数の累積値 R_{sum} をあらわす. また, 表 1 は各ステップ幅に対するアルゴリズムの反復終了時, すなわち $t = 5000$ のときの目的関数の累積値をあらわす.

表 1　各ステップ幅に対する最終的な目的関数値

ステップ幅	$\eta = 0.001$	$\eta = 0.01$	$\eta = 0.1$	$\eta = 1$
目的関数の値	2.9257e+04	3.1450e+04	3.4617e+04	3.4222e+04

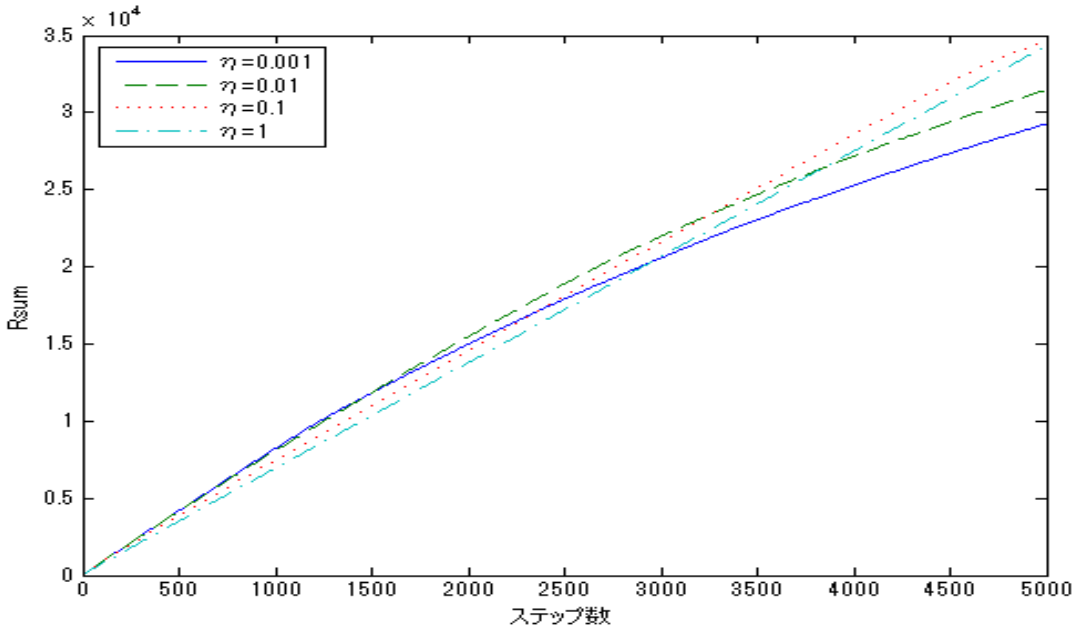

図 1　ステップ幅の比較

図 1 と表 1 からわかるように, ステップ幅 $\eta = 0.1$ のとき, 最終的な目的関数値の和 R_{sum} は最大となっている. これはステップ幅が大きいと $\boldsymbol{\lambda}$ の振動が大きくなり, ステップ幅が小さいと $\boldsymbol{\lambda}$ が最適な潜在価格に近づけず, ともに良い推薦が行えていないと考えられる.

本数値実験の問題設定に対して, ステップ幅 $\eta = 0.1$ が適切なものと考えられる. よって, 以降の節では, ステップ幅 $\eta = 0.1$ として実験を行う.

5.2　一回学習アルゴリズムと提案手法の学習率による比較

本小節では, 学習率 ϵ を変化させたときの一回学習アルゴリズムと提案手法の比較を行う. $\boldsymbol{\lambda}$ の初期値は前節と同様に $\boldsymbol{\lambda}^0 = \mathbf{0}$ とし, ステップ幅は前節で得られた $\eta = 0.1$ とする. 以下の図 2 から図 4 はそれぞれ学習率 $\epsilon = 0.0002, 0.002, 0.02$ としたときの R_{sum} である, 横軸はアルゴリズム中でのステップ数, すなわち到着した客数をあらわし, 縦軸は目的関数の累積値 R_{sum} をあらわす. また表 2 は, 各学習率 ϵ に対するそれぞれの手法の反復終了時, すなわち $t = 5000$ のときの目的関数の累積値をあらわす. ただし () 内の数字は各 ϵ の一回学習アルゴリズムを 1 としたときの比率である.

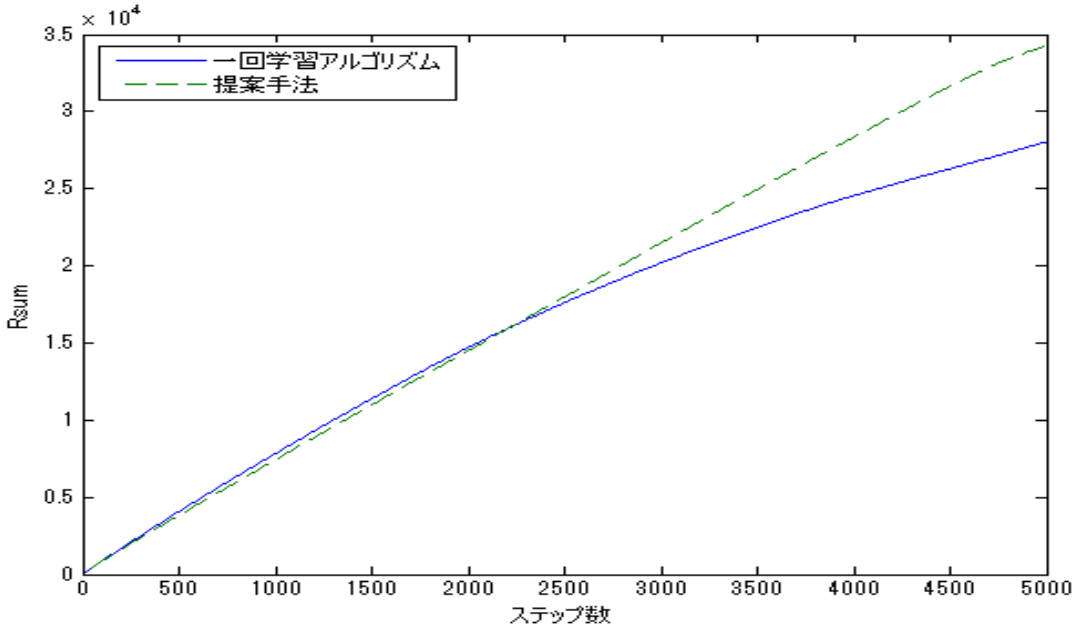

図 2　学習率 $\epsilon = 0.0002$ のときの一回学習アルゴリズムと提案手法

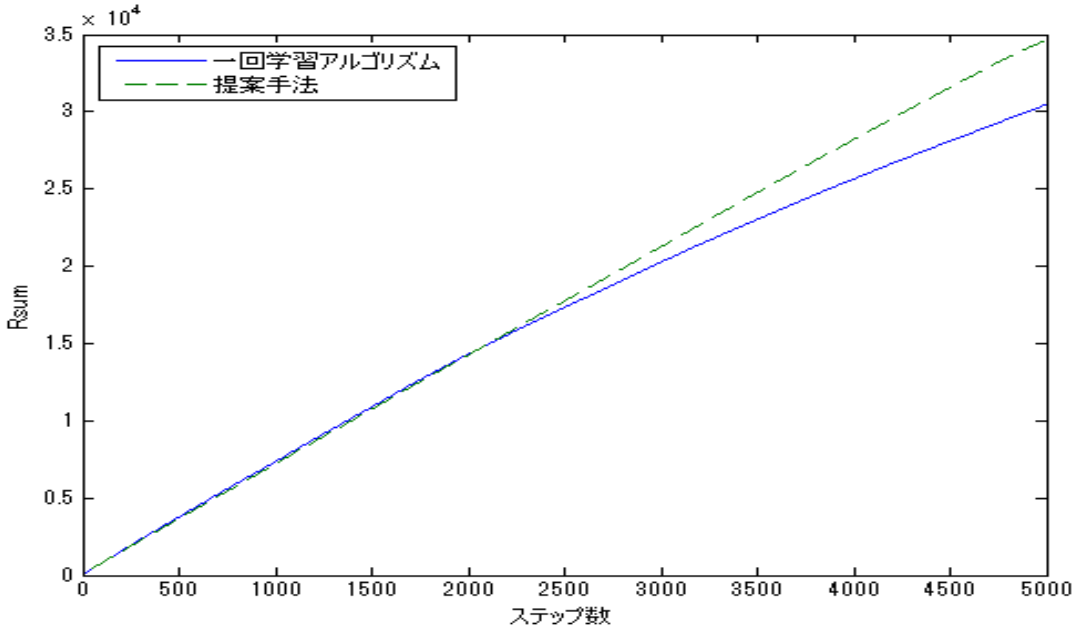

図 3　学習率 $\epsilon = 0.002$ のときの一回学習アルゴリズムと提案手法

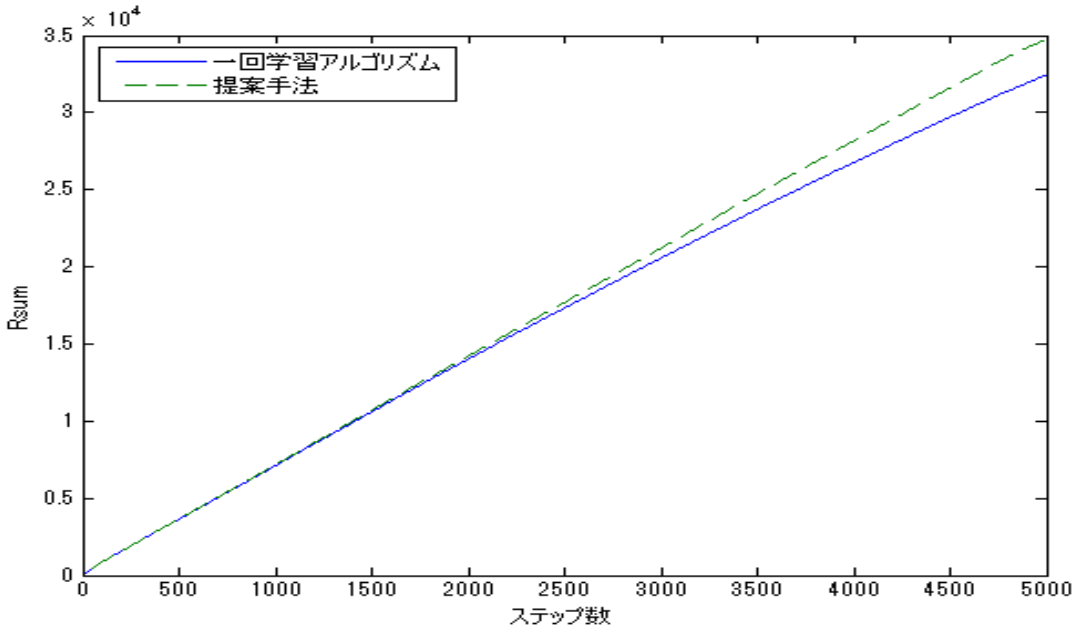

図 4　学習率 $\epsilon = 0.02$ のときの一回学習アルゴリズムと提案手法

表 2　各学習率に対する各手法の最終的な目的関数値

学習率	$\epsilon = 0.0002$	$\epsilon = 0.002$	$\epsilon = 0.02$
一回学習アルゴリズム	2.8021e+04 (100%)	3.0450e+04 (100%)	3.2442e+04 (100%)
提案手法	3.4259e+04 (122.26%)	3.4609e+04 (113.66%)	3.4709e+04 (106.99%)

図 2 から図 4 より, 学習率 ϵ が小さくなるにつれて, 提案手法の利益が, 一回学習アルゴリズムと比較して, 大きくなることが確認できる. これは, 一回学習アルゴリズムで良い潜在価格 $\boldsymbol{\lambda}^{\epsilon n}$ が得られなかった場合でもオンライン劣勾配法により, 最適な潜在価格に近づくよう更新され, より最適な決定を実現できたと考えられる. また, 表 2 からどの学習率 ϵ においても, 提案手法の方が利益が大きくなっていることが分かる.

この結果から, 学習データが少なく, 十分な学習が行えない場合でも, オンライン劣勾配法を用いることで, 最適な潜在価格に近づき, より利益の大きくなる, すなわち最適な潜在価格に近い $\boldsymbol{\lambda}$ が得られたことがわかる.

5.3　学習率 $\epsilon = 0$ と $\epsilon = 0.02$ の場合における提案手法の初期点による影響

本小節では, 初期点 $\boldsymbol{\lambda}^0$ に関して, 提案手法において $\epsilon = 0$ と $\epsilon = 0.02$ としたときの比較を行う. 以下の図 5 と 6 は初期点を $\boldsymbol{\lambda}^0 = \mathbf{0}, 10\boldsymbol{e}$ としたとき, 手法を比較したものである. 横軸はアルゴリズム中でのステップ数, すなわち到着した客数をあらわし, 縦軸は目的関数の累積値 R_{sum} をあらわす. また表 3 は, 初期点に対するアルゴリズムの反復終了時, すなわち $t = 5000$ のときの目的関数の累積値をあらわす. ただし () 内の数字は各 ϵ の一回学習アルゴリズムを 1 としたときの比率である.

表 3　各初期点に対する各 ϵ における提案アルゴリズムによる最終的な目的関数値

$\boldsymbol{\lambda}$ の初期値	$\boldsymbol{\lambda}^0 = \mathbf{0}$	$\boldsymbol{\lambda}^0 = 10\boldsymbol{e}$
$\epsilon = 0$ のときの提案手法	3.3696e+04 (100%)	2.4527e+04 (100%)
$\epsilon = 0.02$ のときの提案手法	3.4127e+04 (101.28%)	3.4077e+04 (138.94%)

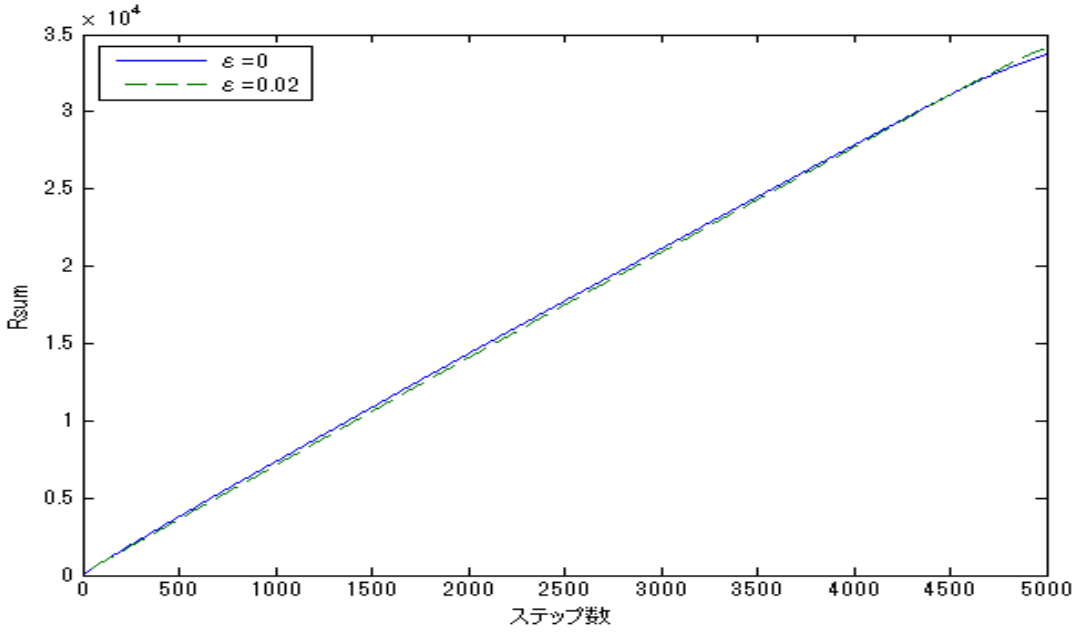

図 5　初期点 $\boldsymbol{\lambda}^0 = \mathbf{0}$ のときの各 ϵ に対する提案手法

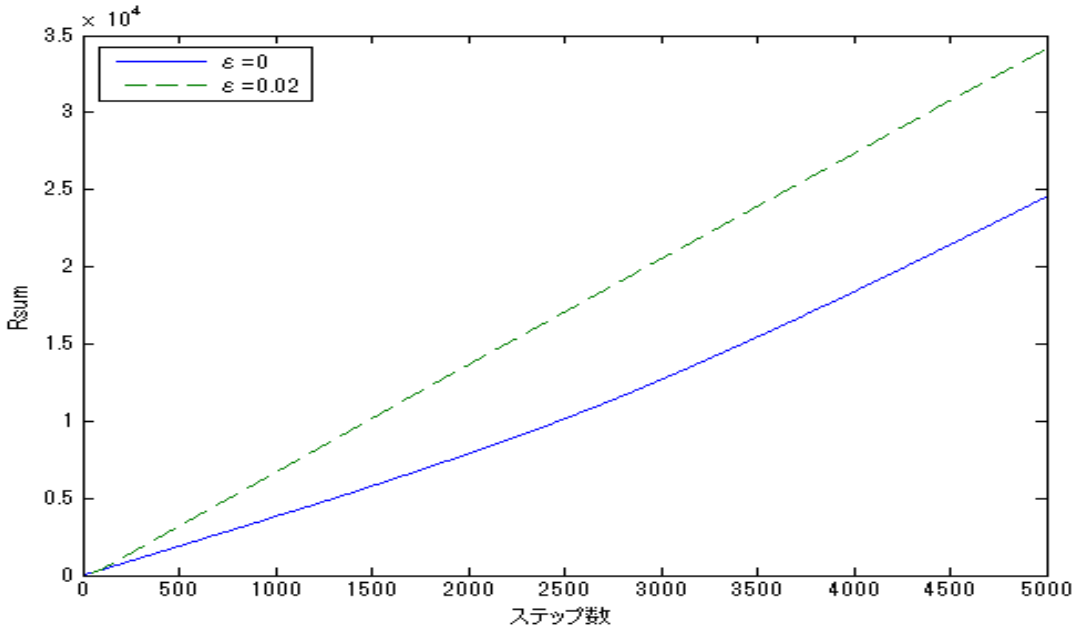

図 6　初期点 $\boldsymbol{\lambda}^0 = 10\boldsymbol{e}$ のときの各 ϵ に対する提案手法

図 5,6 と表 3 からわかるように, $\boldsymbol{\lambda}^0 = \mathbf{0}$ のときは, 提案手法は ϵ の値に関わらずほぼ同じ利益となる. しかし, $\boldsymbol{\lambda}^0 = 10\boldsymbol{e}$ のとき, 提案手法の方が利益が大きくなる. これは, 初期点 $\boldsymbol{\lambda}^0$ が最適な潜在価格に近い状態でアルゴリズムの反復が開始したときは, 一回学習の影響も小さく, 二つの手法にほとんど差が見られないためと考えられる. しかし, 初期点 $\boldsymbol{\lambda}^0$ が最適な潜在価格から離れていた場合, $\epsilon = 0$ とした提案手法では, 反復の間に最適な潜在価格に近づけず, 最適とは遠い推薦をしてしまっている. 一方, $\epsilon = 0.02$ とした提案手法では初期点 $\boldsymbol{\lambda}^0$ が最適な潜在価格から離れていても, ϵn 人目が到着したときの一回学習で近似的に最適な潜在価格に近づき, $\epsilon = 0$ のときの提案手法と比較して良い推薦が行えていると考えられる.

以上より, $\epsilon = 0.02$ のときの提案手法は $\boldsymbol{\lambda}^0$ が最適な潜在価格から離れていても, 一回学習により, 初期の段階で近似的に最適な潜在価格に近づくことができると言える.

6　結論と今後の課題

本研究では, 一回学習アルゴリズムで重みを学習した後, オンライン劣勾配法でこの重みを顧客が到着するごとに更新する手法を提案した. また, その有意性を確かめるため, 実際に数値実験を行った. 数値実験からわかるように, 単純な一回学習アルゴリズムと比較して利益

が大きくなることが確認できた.

今後の課題として, 次のようなものが挙げられる. 一つは, 実データを用いて数値実験を行うことである. 本報告書では利益やコストはすべて乱数で適当に定め, 目的関数の最大化を行った. これを, 実際のデータに対しても, 提案手法を適用することにより, 利益が増加することを確認したい. またそれに伴い, 実データから利益やコストをどのように決定するかという問題が挙げられる.

謝辞

日頃から御教授下さり, 本報告書の作成にあたっては細部に至るまで様々なご指摘と適切なご指導を賜った山下信雄教授に深く感謝致します. また, 日頃からお世話になっている福田秀美助教, 博士課程の先輩である山川雄也さん, ならびに最適化数理研究室の皆様に厚く御礼申し上げます.

参考文献

[1] S. Agrawal, Z. Wamg, Y. Ye, *A Dynamic Near-Optimal Algorithm for Online Linear Programming*, Operations Research, volume 62 (2014), pp.876–890.

[2] M. Babaioff, N. Immorlica, D. Kempe, R. Kleinberg, *Online auctions and generalized secretary problems*, SIGecom Exchange, volume 7, no 2 (2008), pp.1–11.

[3] N. Buchbinder, J. Naor, *Online primal-dual algorithms for covering and packing*, Operations Research, volume 34, no 2 (2009), pp.270–286.

[4] B. コルテ, J. フィーゲン，組合せ最適化, シュプリンガー・ジャパン, 2009.

[5] N. Devanur, T. Hayes, *The Adwords Problem:Online Keyword Matching with Budgeted Bidders under Random Permutations*, ACM Conference on Electronic Commerce, Stanford, Califolnia, USA, 2009.

[6] 福島雅夫，数理計画入門, 朝倉書店, 1996.

[7] J. Hiriart-Urruty, C. Lemaréchal, *Convex Analysis and Minimization Algorithms I*, Springer-Verlag, 1993.

[8] 神嶌敏弘，推薦システムのアルゴリズム (1), 人工知能学会誌, volume 22, no 6 (2007), pp.826–837.

[9] S. Shalev-Shwartz, *Online Learning and Online Convex Optimization*, Foundations and trends in machine learning, volume 4, no 2 (2011), pp. 107–194.