Lagrange Multiplier Method for Multi-campaign Assignment Problem

Yong-Hyuk Kim and Byung-Ro Moon

School of Computer Science & Engineering, Seoul National University
Shillim-dong, Kwanak-gu, Seoul, 151-742 Korea
{yhdfly, moon}@soar.snu.ac.kr

Abstract. It is crucial to maximize marketing efficiency and customer satisfaction in personalized marketing. In this paper, we raise the multiple recommendation problem which occurs when performing several personalized campaigns simultaneously. We formulate the multi-campaign assignment problem to solve this issue and propose methods for solving the problem. The most notable element is the Lagrange multiplier method. Roughly speaking, Lagrange multiplier reduces problem complexity with a minor impact on optimality. However, it is not easy to find Lagrange multipliers in exact accord with problem constraints. We use a genetic algorithm for finding optimal Lagrange multipliers. Through the experiments, we verify the effectiveness of the problem formulation and our genetic approach.

1 Introduction

Customer Relationship Management (CRM) is crucial in acquiring and maintaining loyal customers. To maximize revenue and customer satisfaction, companies try to provide personalized services for customers. A representative effort is one-to-one marketing. The fast development of Internet and mobile communication has enhanced the market for one-to-one marketing. A personalized campaign targets the most attractive customers with respect to the subject of the campaign. So it is important to predict customer preferences for campaigns. Collaborative Filtering (CF) [11] and various data mining techniques including clustering [9] and nearest neighbor algorithm [6] are used to predict customer preferences for campaigns [3]. Especially, since CF is fast and simple, it is widely used for personalization in e-commerce [10] [8]. There have been a number of customer-preference estimation methods based on CF [13] [7] [1]. A survey for recommender systems in e-commerce is given in [12].

As personalized campaigns are frequently performed, several campaigns often happen to run simultaneously. It is often the case that an attractive customer for a specific campaign tends to be attractive for other campaigns. If we perform separate campaigns without considering this problem, some customers may be bombarded by a considerable number of campaigns. We call this the multiple recommendation problem. The larger the number of recommendations for a customer, the lower the customer interest for campaigns. In the long run, the customer response for campaigns drops. It lowers the marketing efficiency as well as customer satisfaction, which diminishes the customer loyalty and sometimes results in the disastrous "churning." Unfortunately, traditional methods only focused on the effectiveness of individual campaigns and did not consider the problem with respect to the multiple recommendations. In the situation that several campaigns are performed at the same time, it is necessary to find the optimum campaign assignment to customers considering the recommendations in other campaigns.

In this paper, we define the multi-campaign assignment problem (MCAP) considering the multiple recommendation problem and propose three methods for the issue. We show that one can solve the MCAP to optimality by a dynamic programming algorithm. Although the dynamic programming algorithm guarantees optimal solutions, it becomes intractable for large problems. We thus propose a constructive heuristic that not only has practical time complexity but also shows good performance. However, since it is a heuristic, it does not guarantee optimal solutions. Finally, we propose the Lagrange multiplier method with the advantages of both the dynamic programming method and the constructive heuristic. It has linear time complexity for the fixed number of campaigns and sacrifices little on the optimality. Furthermore, the Lagrange multiplier method also provides a good upper bound and can be used to measure the suboptimality of other heuristics. But, randomly generated Lagrange multipliers do not satisfy problem constraints. It is not easy to find Lagrange multipliers in exact accord with problem constraints. Since it is important to find good Lagrange multipliers, we use genetic algorithms to optimize Lagrange multipliers. We also verify the effectiveness of the proposed genetic algorithm with field data.

The remainder of this paper is organized as follows. In Section 2, we describe the multi-campaign assignment problem. The detailed description of response suppression function, the key element for the problem, is given in Section 3. In Section 4, we propose three algorithms for the problem: a dynamic programming algorithm, a heuristic algorithm, and the Lagrange multiplier method. In Section 5, we propose a genetic algorithm for optimizing Lagrange multipliers. We show experimental results in Section 6 and finally make conclusions in Section 7.

2 Problem Formulation

The multi-campaign assignment problem is to find customer-campaign assignments that maximize the effects of campaigns. The main difference with independent campaigns lies in that the customer response for campaigns is influenced by multiple recommendations.

Let N be the number of customers, $C = \{1, 2, \ldots, N\}$ be the set of customers, and K be the number of campaigns. In the following, we describe the input, output, constraints, and evaluation function for the problem.

Input: For each customer, the preference for each campaign is given. Each campaign has a weight. A response suppression function R related to multiple recommendations is given.

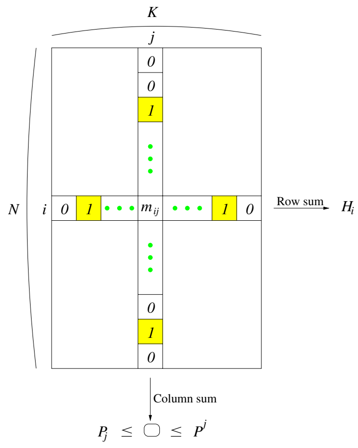

Fig. 1. The campaign assignment matrix $M = (m_{ij})$

– $w_1, w_2, \ldots, w_K$: the weight of each campaign ($w_j > 0$ for each campaign j).
– $f_1, f_2, \ldots, f_K : C \longrightarrow [0, \infty)$: the preference function of each campaign.
– $R : \mathcal{N} \longrightarrow [0, 1]$: the response suppression function with respect to the number of recommendations.

The preferences for a campaign can be gotten from some existing method such as CF. If H_i is the number of multiple recommendations for customer i and $f_j(i)$ is the preference of customer i for campaign j, the actual preference of customer i for campaign j becomes $R(H_i) \cdot f_j(i)$.

Constraints: The maximum and minimum numbers of recommendations for each campaign are enforced. Let P^j be the maximum number of recommendations for campaign j, and P_j be the minimum number of recommendations for campaign j. Then the number of recommendations in campaign j is between P_j and P^j. These constraints can be tight or loose according to the situation of business.

Output: The output is a binary campaign assignment matrix $M = (m_{ij})$ in which m_{ij} indicates whether campaign j is assigned to customer i. Figure 1 shows an example campaign assignment matrix.

Evaluation: The *campaign preference* for campaign j is defined to be the actual preference sum of recommended customers for campaign j as follows: $\sum_{i \in C; m_{ij}=1} R(H_i) \cdot f_j(i)$. The fitness $F(M)$ of a campaign assignment matrix M is the weighted sum of campaign preferences.

$$F(M) = \sum_{j=1}^{K} \left(w_j \cdot \sum_{i \in C; m_{ij}=1} R(H_i) \cdot f_j(i) \right) .$$

The objective is to find a matrix M that maximizes F.

Nomenclature: The nomenclature that would be used in the remainder of this paper is given in the below:

$\boldsymbol{w} = (w_1, w_2, \ldots, w_K) \in \mathcal{R}^K$: the campaign weight vector
$f_j : C \longrightarrow [0, \infty)$: the preference function for each campaign j
$R : \mathcal{N} \longrightarrow [0, 1]$: the response suppression function
(for convenience, we assume $R(0) = 0$)
$\boldsymbol{p}^* = (P^1, P^2, \ldots, P^K) \in \mathcal{N}^K$: the upper bound constraint vector
$\boldsymbol{p}_* = (P_1, P_2, \ldots, P_K) \in \mathcal{N}^K$: the lower bound constraint vector
$M = (m_{ij})$: the $N \times K$ binary campaign assignment matrix
$M' = (m'_{ij})$: the $N \times K$ real matrix where $m'_{ij} = f_j(i) \cdot m_{ij}$
$\boldsymbol{m_i} = (m_{i1}, m_{i2}, \ldots, m_{iK})$: the i^{th} row vector of the matrix M
$\boldsymbol{m'_i} = (m'_{i1}, m'_{i2}, \ldots, m'_{iK})$: the i^{th} row vector of the matrix M'
$\mathbf{1}_n$: an n-dimensional vector $(1, 1, \ldots, 1)$
$H_i = \boldsymbol{m_i} \cdot \mathbf{1}_K^T$: the number of multiple recommendations for customer i
$\sigma_i = \boldsymbol{m'_i} \cdot \boldsymbol{w}^T$: the weighted sum of preferences of customer i for recommended campaigns

Formal Definition: More formally, we define the multi-campaign assignment problem (MCAP) as follows.

Definition 1 *The* multi-campaign assignment problem *is to find a campaign assignment matrix $M = (m_{ij})$ that maximizes $\langle \boldsymbol{w}, \boldsymbol{R}(\mathbf{1}_K M^T) \cdot M' \rangle$ subject to $\boldsymbol{p}_* \leq \mathbf{1}_N M \leq \boldsymbol{p}^*$, where $\boldsymbol{R}(x_1, x_2, \ldots, x_n) = (R(x_1), R(x_2), \ldots, R(x_n))$.*

3 The Response Suppression Function

In case of multiple campaign recommendations, the customer response rate drops as the number of recommendations grows. We introduce the response suppression function for the response-rate degradation with multiple recommendations. Finding a reasonable response suppression function is the subject to be tuned over abundant field data. The optimal response suppression function depends on situations and it is a longterm research topic. Instead, we devised a number of suppression functions. Definitely, the function should be monotonic nonincreasing.

Figure 2 shows a basic response suppression function. The function is nonnegative monotonic nonincreasing with the maximum value one. It was derived from the Gaussian function. By the function, the preference for a campaign drops to, e.g., one third when four campaigns are performed simultaneously to a customer. In the experiment, we used the function R as the response suppression function.

4 Methods

4.1 Dynamic Programming

We can find the optimal campaign assignment matrix of the MCAP using dynamic programming (DP). DP has been useful for diverse problems [2] [5]. We

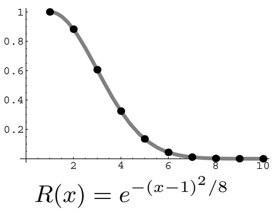

Fig. 2. Basic response suppression function ($R(x) = 0$ for $x \geq 11$)

```
DP($R$, $P_*$, $P^*$)
{
    $S_0(\mathbf{0}_K) = 0$;
    **for** each $\boldsymbol{v}$ such that $\mathbf{0}_K \neq \boldsymbol{v} \leq P^*$, $S_0(\boldsymbol{v}) = -\infty$;
    **for** $i = 1$ **to** $N$
        **for** each $\boldsymbol{v} \leq P^*$ {
            $S_i(\boldsymbol{v}) = max_{\boldsymbol{m_i};\ \forall j, m_{ij} \leq v_j}(S_{i-1}(\boldsymbol{v} - \boldsymbol{m_i}) + R(H_i) \cdot \sigma_i)$;
            $L_i(\boldsymbol{v}) = argmax_{\boldsymbol{m_i};\ \forall j, m_{ij} \leq v_j}(S_{i-1}(\boldsymbol{v} - \boldsymbol{m_i}) + R(H_i) \cdot \sigma_i)$;
        }
    $optimum\_value = max_{\boldsymbol{v};\ \boldsymbol{p_*} \leq \boldsymbol{v} \leq \boldsymbol{p^*}} S_N(\boldsymbol{v})$;
    $\boldsymbol{rv} = argmax_{\boldsymbol{v};\ \boldsymbol{p_*} \leq \boldsymbol{v} \leq \boldsymbol{p^*}} S_N(\boldsymbol{v})$;
    **for** $i = N$ **to** 1 {
        $\boldsymbol{m_i} = L_i(\boldsymbol{rv})$;
        $\boldsymbol{rv} = \boldsymbol{rv} - \boldsymbol{m_i}$;
    }
    **return** *optimum_value* and $M$;
}
```

$\mathbf{0}_n$ is an n-dimensional vector $(0, 0, \ldots, 0)$ and $\boldsymbol{v} = (v_1, v_2, \ldots, v_K) \in \mathcal{N}^K$

Fig. 3. A dynamic programming algorithm for MCAP

devised a DP algorithm for MCAP. Figure 3 shows the pseudo-code. In the algorithm, $S_i(\boldsymbol{v})$ means the optimum fitness of the multi-campaign assignment problem with $P_* = P^* = \boldsymbol{v}$ and the customer set $\{1, 2, \ldots, i\}$. The algorithm requires $O(NK \cdot \Pi_{j=1}^{K} P^j)$ space. Since the maximum number of $\boldsymbol{m_i}$ configurations is 2^K, it takes $O(NK2^K \cdot \Pi_{j=1}^{K} P^j)$ time. If K is a fixed number, this is a polynomial-time algorithm. However, when K is not small and almost all P^j's are $\Omega(N)$, it is nearly intractable. An optimum assignment matrix is obtained by backward links $L_i(\boldsymbol{v})$ stored during the process of dynamic programming. The proposed DP algorithm is only applicable to problems with small K, N pairs. Thus we need heuristics for large problems. Since the DP algorithm guarantees optimal solutions, it is useful in evaluating the suboptimality of other heuristics proposed in the next subsection.

```
$M = O$;
Create an AVL tree;
**for** each campaign $j$
        Find topmost $\alpha$ customers with high gain values, and
            for each customer $i$ of them, insert the node ($g_{ij}$, ($i$, $j$))
            into the AVL tree;
**do** {
        Choose a maximum gain node $(i, j)$ and delete it from the AVL tree;
        **if** ($g_{ij}$ is not positive and every campaign satisfies
            its constraint on the minimum number of recommendations)
                **then** break;
        **if** (the campaign $j$ is not full) **then** {
                Recommend the campaign $j$ to the customer $i$; // $m_{ij} \leftarrow 1$
                **for** each not-full campaign $k$
                        Update the gain values $g_{ik}$ in the AVL tree;
        }
} **until** (the AVL tree is empty or every campaign is full)
```

$*$ We set α to be $N/2$ in our experiments.

Fig. 4. The constructive assignment algorithm

4.2 Heuristic Algorithm

Starting at the initial situation that no customer is recommended any campaigns, we iteratively assign campaigns to customers by a greedy method. We call this algorithm Constructive Assignment Algorithm (CAA). Define the *gain* g_{ij} of a pair (customer i, campaign j) to be the amount of fitness gain by assigning campaign j to customer i. Initially, the gain g_{ij} is equal to $w_j f_j(i)$, the product of campaign weight and the preference of customer i for campaign j. Generally the gain is formulated as: $g_{ij} = R(H_i + 1) \cdot (\sigma_i + w_j f_j(i)) - R(H_i) \cdot \sigma_i$. We use an AVL tree for the efficient management of real-valued gains. Figure 4 shows the template of the CAA. It chooses the most attractive α customers for each campaign and inserts them into the AVL tree. Next, it iteratively performs the following. It chooses a pair (customer i, campaign j) with the maximum gain and, if the gain is positive and campaign j does not exceed the maximum number of recommendations, it recommends campaign j to customer i. If a recommendation is done, it updates the gains of customer i for the other campaigns. We naturally assume that the response suppression function is monotonic nonincreasing. In that case, any gain cannot be positive after the maximum gain drops below zero. When the maximum gain is not positive, the algorithm terminates as far as every campaign satisfies the constraint on the minimum number of recommendations. There are $O(NK)$ nodes in the AVL tree during a run of the algorithm. Hence, both the insertion and deletion of a node in the AVL tree take $O(log(NK))$, i.e., $O(logN)$. The time complexity of the algorithm is $O(NK^2logN)$. If the algorithm is implemented without an AVL tree, it would take $O(N^2K^3)$.

```
LM($R$, $\boldsymbol{\lambda}$)
{
        for $i = 1$ to $N$ {
                $F_i = max_{\boldsymbol{m_i} \in \{0,1\}^K}(R(H_i) \cdot \sigma_i - \langle \boldsymbol{\lambda}, \boldsymbol{m_i} \rangle)$;
                $\tilde{\boldsymbol{m_i}} = argmax_{\boldsymbol{m_i} \in \{0,1\}^K}(R(H_i) \cdot \sigma_i - \langle \boldsymbol{\lambda}, \boldsymbol{m_i} \rangle)$;
        }
        $\boldsymbol{p} = \sum_{i=1}^N \tilde{\boldsymbol{m_i}}$;
        $optimum\_value = \sum_{i=1}^N F_i + \langle \boldsymbol{\lambda}, \boldsymbol{p} \rangle$;
        return $optimum\_value$, $\tilde{M} = (\tilde{m_{ij}})$, and $\boldsymbol{p}$;
}
```

$\boldsymbol{\lambda} = (\lambda_1, \lambda_2, \ldots, \lambda_K) \in \mathcal{R}^K$ and $\boldsymbol{p} = (p_1, p_2, \ldots, p_K) \in \mathcal{N}^K$

Fig. 5. Lagrange multiplier method for MCAP

4.3 Lagrange Multipliers

Since the multi-campaign assignment problem is a constraint optimization problem, it can be solved using Lagrange multipliers as in the following:

$$Max\ \{\langle \boldsymbol{w}, \boldsymbol{R}(\mathbf{1}_K M^T) \cdot M' \rangle - \langle \boldsymbol{\lambda}, \mathbf{1}_N M \rangle\}.$$

However, since it is a discrete problem which is not differentiable, it is formulated to a restricted form. Figure 5 shows the pseudo-code of the Lagrange multiplier method for MCAP. The following proposition guarantees its optimality.

Proposition 1 *(Optimality) Given the constraint vector $\boldsymbol{p}$, Lagrange multiplier method outputs an optimum matrix $\tilde{M} = (\tilde{m_{ij}})$.*

Proof: Suppose that $\tilde{M} = (\tilde{m_{ij}})$ is not an optimum matrix. Let $\bar{M} = (\bar{m_{ij}})$ be an optimum matrix with $\boldsymbol{p}$ as the constraint vector. Let $\bar{F_i}$ be $R(H_i) \cdot \sigma_i - \langle \boldsymbol{\lambda}, \bar{\boldsymbol{m_i}} \rangle$ for each i in campaign assignment matrix $\bar{M}$. For each i, by the optimality of F_i, $F_i \geq \bar{F_i}$. Since $\boldsymbol{p}$ is given, $\sum_{i=1}^N F_i + \langle \boldsymbol{\lambda}, \boldsymbol{p} \rangle \geq \sum_{i=1}^N \bar{F_i} + \langle \boldsymbol{\lambda}, \boldsymbol{p} \rangle$. This contradicts that $\tilde{M} = (\tilde{m_{ij}})$ is not an optimum matrix. Therefore, $\tilde{M} = (\tilde{m_{ij}})$ is an optimum matrix. ■

Given a Lagrange multiplier vector $\boldsymbol{\lambda}$, we can get the constraint vector $\boldsymbol{p}$ and the optimum result with $\boldsymbol{p}$ by the Lagrange multiplier method.

Proposition 2 *Let $\boldsymbol{\lambda} = (\lambda_1, \lambda_2, \ldots, \lambda_K)$ and $\boldsymbol{\lambda}' = (\lambda'_1, \lambda'_2, \ldots, \lambda'_K)$. Suppose that $\lambda_i = \lambda'_i$ for $i \neq k$ and $\lambda_k \neq \lambda'_k$. Then, if $\lambda_k < \lambda'_k$, $p_k(\boldsymbol{\lambda}) \geq p_k(\boldsymbol{\lambda}')$ and if $\lambda_k > \lambda'_k$, $p_k(\boldsymbol{\lambda}) \leq p_k(\boldsymbol{\lambda}')$.*

Proof: Suppose that $\boldsymbol{\lambda}$ and $\boldsymbol{\lambda}'$ correspond to $M = (m_{ij})$ and $M' = (m'_{ij})$, respectively. By the optimality of $\boldsymbol{\lambda}$, $\sum_{i=1}^N F_i(\boldsymbol{\lambda}) \geq F'_i(\boldsymbol{\lambda})$. By the optimality of $\boldsymbol{\lambda}'$, $\sum_{i=1}^N F'_i(\boldsymbol{\lambda}') \geq F_i(\boldsymbol{\lambda}')$. From the summation of above two formulas,

$$\sum_{i=1}^N -\langle \boldsymbol{\lambda}, \boldsymbol{m}_i \rangle + \sum_{i=1}^N -\langle \boldsymbol{\lambda}', \boldsymbol{m}'_i \rangle \geq \sum_{i=1}^N -\langle \boldsymbol{\lambda}, \boldsymbol{m}'_i \rangle + \sum_{i=1}^N -\langle \boldsymbol{\lambda}', \boldsymbol{m}_i \rangle$$

$$\begin{aligned}
&\iff \sum_{i=1}^{N} -\lambda_k m_{ik} + \sum_{i=1}^{N} -\lambda'_k m'_{ik} \geq \sum_{i=1}^{N} -\lambda_k m'_{ik} + \sum_{i=1}^{N} -\lambda'_k m_{ik} \\
&\iff -\lambda_k p_k(\boldsymbol{\lambda}) - \lambda'_k p_k(\boldsymbol{\lambda}') \geq -\lambda_k p_k(\boldsymbol{\lambda}') - \lambda'_k p_k(\boldsymbol{\lambda}) \\
&\iff (\lambda'_k - \lambda_k)(p_k(\boldsymbol{\lambda}) - p_k(\boldsymbol{\lambda}')) \geq 0. \quad \blacksquare
\end{aligned}$$

Proposition 3 *(Sensitivity) Suppose that $\boldsymbol{\lambda}_0$ and $\boldsymbol{\lambda}_1$ correspond to $\{M_0, \boldsymbol{p}_0\}$ and $\{M_1, \boldsymbol{p}_1\}$, respectively. Then, the following inequalities are satisfied.*

$$\langle \boldsymbol{\lambda}_0, \boldsymbol{p}_0 - \boldsymbol{p}_1 \rangle \leq F(M_0) - F(M_1) \leq \langle \boldsymbol{\lambda}_1, \boldsymbol{p}_0 - \boldsymbol{p}_1 \rangle.$$

In particular, for any assignment matrix M, $F(M) - F(M_0) \leq \langle \boldsymbol{\lambda}_0, \mathbf{1}_N M - \boldsymbol{p}_0 \rangle$.

Proof: By the optimality of $\boldsymbol{\lambda}_0$, $F(M_0) - \langle \boldsymbol{\lambda}_0, \boldsymbol{p}_0 \rangle \geq F(M_1) - \langle \boldsymbol{\lambda}_0, \boldsymbol{p}_1 \rangle$. Hence, $F(M_0) - F(M_1) \geq \langle \boldsymbol{\lambda}_0, \boldsymbol{p}_0 - \boldsymbol{p}_1 \rangle$. By the optimality of $\boldsymbol{\lambda}_1$, $F(M_1) - \langle \boldsymbol{\lambda}_1, \boldsymbol{p}_1 \rangle \geq F(M_0) - \langle \boldsymbol{\lambda}_1, \boldsymbol{p}_0 \rangle$. Hence, $F(M_0) - F(M_1) \leq \langle \boldsymbol{\lambda}_1, \boldsymbol{p}_0 - \boldsymbol{p}_1 \rangle$. $\blacksquare$

Fact 1 *F is a nondecreasing function of $\boldsymbol{p}$ (i.e., $\boldsymbol{p} < \boldsymbol{p}' \Rightarrow F \leq F'$). This implies $\boldsymbol{\lambda} \geq \mathbf{0}_K$.*

From Proposition 2, p_i inversely grows with λ_i. However, it is not easy to find $\boldsymbol{\lambda}$ satisfying $\boldsymbol{p}_* \leq \boldsymbol{p} \leq \boldsymbol{p}^*$. Proposition 3 shows that our Lagrange multiplier method also gives good upper bounds. By using the Lagrange multiplier method, the problem of finding the optimum campaign assignment matrix becomes that of finding a K-dimensional real vector $\boldsymbol{\lambda}$ with Lagrange multipliers. The Lagrange multiplier method takes $O(NK2^K)$ time. It is more tractable than the original problem. Roughly, for a fixed number K, the problem size is lowered from $O(N^{K+1})$ to $O(N)$.

5 A Genetic Algorithm for Optimizing Lagrange Multipliers

We propose a genetic algorithm (GA) for optimizing Lagrange multipliers. It conducts a search using an evaluation function with penalties for violated constraints. The search space with N customers and K campaigns has $\Pi_{i=0}^{K} \Pi_{j=P_i}^{P^i} \binom{N}{j}$ elements if all possibilities are considered. But, too large a problem size may make the search intractable. Our GA provides an alternative search method to find a good campaign assignment matrix by optimizing K Lagrange multipliers instead of directly dealing with the campaign assignment matrix.

5.1 Genetic Operators

The general framework of a typical steady-state genetic algorithm is used in our GA. In the following, we describe each part of the GA.

- *Encoding:* Each solution in the population is represented by a chromosome. Each chromosome contains K Lagrange multipliers. A real encoding is used for representing the chromosome $\boldsymbol{\lambda}$. From Fact 1, each Lagrange multiplier is nonnegative. So we set a gene corresponding to a Lagrange multiplier to be a real value between 0.0 and 1.0.
- *Initialization:* The GA first creates p real vectors between $\mathbf{0}_K$ and $\mathbf{1}_K$ at random. We set the population size p to be 100.
- *Selection and crossover:* To select two parents, we use a proportional selection scheme where the probability for the best solution to be chosen is four times higher than that for the worst solution. A crossover operator creates a new offspring by combining parts of the parents. We use the uniform crossover [14].
- *Mutation:* After the crossover, mutation operator is applied to the offspring. We use a variant of Gaussian mutation. We devised it considering the relation between λ_i and p_i given in Proposition 2. For each selected gene λ_i, we choose a Gaussian random real number t normally distributed with parameters $\mu = 0$ and $\sigma^2 = 1$ (i.e., $t \sim N(0, 1)$). If the corresponding constraint p_i is less than P^i, we set λ_i to $\lambda_i + (1 - \lambda_i) \cdot |t|/\gamma$. Otherwise, we set λ_i to $\lambda_i - \lambda_i \cdot |t|/\gamma$. Mutation rate is 0.05 and the constant γ is 2.58.
- *Replacement and stopping condition:* After generating an offspring, our GA replaces the worse of the two parents with the offspring. It is called *preselection* replacement [4]. Our GA stops when one of the following two conditions is satisfied: i) the number of generations reaches 15,000, ii) when the fitness of the worst chromosome is equal to the fitness of the best one.

5.2 Evaluation Function

Our evaluation function is to find a Lagrange multiplier vector $\boldsymbol{\lambda}$ that has high fitness satisfying the constraints as *much* as possible. In our GA, the following evaluation function is used: $F(M) - \sum_{j=1}^{K} c_j \cdot excess(j)$ where c_j is the campaign penalty and $excess(j)$ is the number of exceeded recommendations for campaign j. In this paper, we used K times the weighted average preference of campaign j as the campaign penalty c_j (i.e., $c_j = K \cdot w_j \cdot \frac{1}{N} \sum_{i=1}^{N} f_j(i)$).

6 Experimental Results

6.1 Inputs and Parameters

We used the preference values estimated by CF from a set of field data with 48,559 customers and 10 campaigns. We used the function R which was derived from Gaussian function as the response suppression function. The weight for each campaign was given equally 0.1. The maximum number of recommendations for each campaign was set to 7,284, equal to 15% of the total number of customers. The minimum number of recommendations was set to 0.

The average preference of customers for a campaign was 4.74. We examined the Pearson correlation coefficient of preferences for each pair of campaigns.

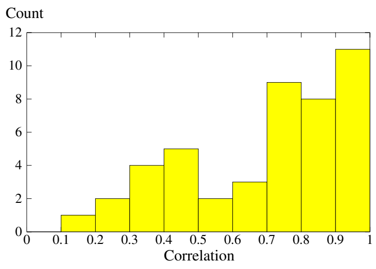

Fig. 6. Histogram for Pearson correlation coefficient of each campaign pair

Table 1. Comparison for Small Data Sets ($K = 3$)

Method	$N = 200$	$N = 400$	$N = 600$	$N = 800$	$N = 1000$
CAA (Time*)	733.61 (0.0010)	1303.98 (0.0021)	2030.10 (0.0032)	2725.59 (0.0045)	3479.30 (0.0058)
DP† (Time*)	734.64 (145.82)	1304.43 (2336.41)	2031.22 (11553.92)	2726.99 (28838.84)	3483.02 (71479.06)

Maximum 50% and minimum 0% recommendation for each campaign.
Equally weighted campaigns; i.e., $w_j = 0.33$ for each campaign j.
† The optimum value.
$*$ CPU seconds on Pentium III 1 GHz.

Figure 6 shows its histogram. Thirty three pairs (about 73%) out of totally 45 pairs showed higher correlation coefficient than 0.5. This property of field data provides a good reason for the need of MCAP modeling.

6.2 Analysis of Results

First, we compare the proposed heuristic algorithm with the DP algorithm which guarantees optimality. Due to the running time of dynamic programming, we restricted the instances with up to 1,000 customers and three campaigns. Table 1 shows their performance. We chose three campaigns among the 10 campaigns, and the sets of customers were sampled at random to prepare the sets with sizes from 200 to 1,000. The maximum number of recommended customers was set to be a half the total number of customers. The minimum number of recommendations was set to 0. The CAA was much faster than the dynamic programming algorithm while its results reached fairly close to the optimal solutions. This implies that CAA is an attractive practical heuristic.

Table 2 shows the performance of the independent campaign and various multi-campaign algorithms in the multi-campaign formulation. We use all the 48,559 customers and 10 campaigns here. The figures in the table represent the fitness values $F(M)$ described in Section 2. The results of "Independent" campaign are from K independent campaigns without considering their relationships with others. Although the independent campaign was better than the "Random"

Table 2. Comparison of Algorithms

Method	$K = 7$	$K = 8$	$K = 9$	$K = 10$
Independent	38546.25	36779.68	30259.65	26452.75
Random	32487.36	29385.14	27187.26	25951.19
CAA	85565.05	79885.40	71863.76	66473.63
LM-Random†	45401.90	53638.58	52351.99	46990.46
LM-GA†	**86474.30**	**80287.72**	**72518.22**	**66932.79**
LM Upper Bound*	87776.38	81428.48	73446.60	67654.77

† Average over 10 runs.

∗ Upper bound by Proposition 3 and the results of LM-GA.

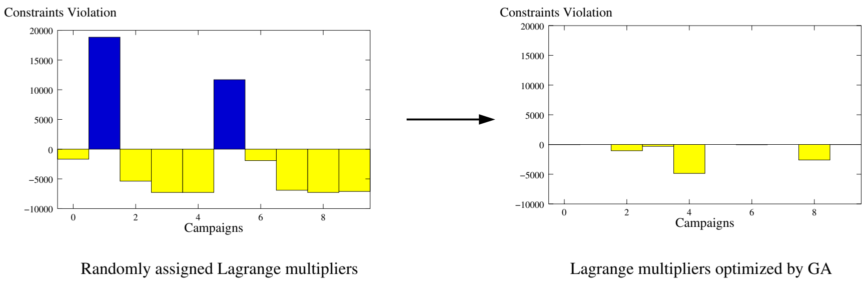

Fig. 7. An example of optimized Lagrange multipliers

assignment in multi-campaign formulation, it was not comparable to the other multi-campaign algorithms. The results of Random assignment are the average over 100 runs. The solution fitnesses of CAA heuristic were overall more than two times higher than those of the independent campaign.

Next, we compare the proposed heuristic algorithm with the Lagrange multiplier method. Table 2 shows their performance, too. LM-Random means the best result among randomly generated 15,000 Lagrange multiplier vectors. Even in the best result, there were constraints violations for some campaigns. However, when Lagrange multipliers are optimized by a genetic algorithm (LM-GA), we always found the best-quality solutions satisfying all constraints. Moreover, LM-GA performed better than the proposed deterministic heuristic, CAA. Figure 7 shows an example of optimized 10 Lagrange multipliers. For a typical randomly generated Lagrange multiplier vector, we can observe that excessive constraints violations for some campaigns (the left figure of Fig. 7). However, with the Lagrange multipliers optimized by GA, we can see that all constraints were satisfied (the right figure of Fig. 7).

As a final solution of the GA with $K = 10$ and fitness 66980.76, we got the following constraint vector: $\boldsymbol{p} = \{7254, 7269, 6238, 6982, 2422, 7284, 7237, 7282, 4675, 7260\}$ (on the right side of Fig. 7). When the vector $\boldsymbol{p}$ is used as the upper bound constraint vector $\boldsymbol{p}^*$ of the MCAP, CAA produced the campaign assignment matrix with a fitness value 65819.80. This gives another usage of the

Lagrange multiplier method. The suboptimality of heuristic algorithms can be measured by using the optimality of the Lagrange multiplier method.

7 Conclusions

The representative contributions of this paper are as follows. First, we proposed and formulated the multi-campaign assignment problem (MCAP). Second, we presented a dynamic programming algorithm and a constructive heuristic algorithm for MCAP. Finally, we proposed Lagrange multiplier method for the MCAP and optimized it using a genetic algorithm. Their performance was examined with experiments.

Our Lagrange multiplier method is fast and outputs optimal solutions. But, it is not easy to find Lagrange multipliers satisfying all constraints. When combined with the genetic algorithm, we could find high-quality Lagrange multipliers. The Lagrange multiplier method may be combined with other metaheuristics as well such as evolutionary strategy, simulated annealing, tabu search, and large-step Markov chains. Experimentation with respect to these methods is left for future study.

Acknowledgments. This study was supported by Brain Korea 21 Project. The ICT at Seoul National University provided research facilities for this study.

References

1. C. C. Aggarwal, J. L. Wolf, K. L. Wu, and P. S. Yu. Horting hatches an egg: A new graph-theoretic approach to collaborative filtering. In *Knowledge Discovery and Data Mining*, pages 201–212, 1999.
2. R. Bellman. *Dynamic Programming*. Princeton University Press, 1957.
3. M. Berry and G. Linoff. *Data Mining Techniques For Marketing, Sales, and Customer Support*. John Wiley & Sons, Inc, 1997.
4. D. Cavicchio. *Adaptive Search Using Simulated Evolution*. PhD thesis, University of Michigan, Ann Arbor, MI, 1970.
5. S. E. Dreyfus and A. M. Law. *The Art and Theory of Dynamic Programming*. Academic Press, 1977.
6. C. Feustel and L. Shapiro. The nearest neighbor problem in an abstract metric space. *Pattern Recognition Letters*, 1:125–128, 1982.
7. D. Greening. Building consumer trust with accurate product recommendations. Technical Report LMWSWP-210-6966, LikeMinds White Paper, 1997.
8. J. L. Herlocker, J. A. Konstan, A. Borchers, and J. Riedl. An algorithmic framework for performing collaborative filtering. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 230–237, 1999.
9. A. K. Jain and R. C. Dubes. *Algorithms For Clustering Data*. Prentice Hall, 1988.
10. J. A. Konstan, B. N. Miller, D. Maltz, J. L. Herlocker, L. R. Gordan, and J. Riedl. GroupLens: applying collaborative filtering to usenet news. *Communications of the ACM*, 40:77–87, 1997.

11. P. Resnick, N. Iacovou, M. Sushak, P. Bergstrom, and J. Riedl. GroupLens: An open architecture for collaborative filtering of netnews. In *Proceedings of the 1994 Computer Supported Collaborative Work Conference*, pages 175–186, 1994.
12. J. B. Schafer, J. A. Konstan, and J. Riedi. Recommender systems in e-commerce. In *ACM Conference on Electronic Commerce*, pages 158–166, 1999.
13. U. Shardanand and P. Maes. Social information filtering: Algorithms for automating "word of mouth". In *Proceedings of ACM CHI'95 Conference on Human Factors in Computing Systems*, volume 1, pages 210–217, 1995.
14. G. Syswerda. Uniform crossover in genetic algorithms. In *Third International Conference on Genetic Algorithms*, pages 2–9, 1989.