多目的意思決定のための最適組合せ問合せ

郭　　茜† 　石川　佳治††,†††

† 名古屋大学情報科学研究科　〒 464–8601 名古屋市千種区不老町
†† 名古屋大学情報基盤センター　〒 464–8601 名古屋市千種区不老町
††† 国立情報学研究所　〒 101—-0003 東京都千代田区一ツ橋 2 丁目 1-2
E-mail: †guoxi@db.itc.nagoya-u.ac.jp, ††ishikawa@itc.nagoya-u.ac.jp

あらまし　意思決定においては，しばしば多目的の最適化が行われる．本論文では，目的とする多次元ベクトルにできるだけ近くなるようなオブジェクトの組合せを求める，多目的の最適組合せ問合せの概念を提案し，その効率的問合せのためのアルゴリズムを示す.
キーワード　目的とする多次元ベクトル，組合せ，意思決定，R-木.

Multi-Objective Optimal Combination Queries

Xi GUO† and Yoshiharu ISHIKAWA††,†††

† Graduate School of Information Science, Nagoya University
Furo-cho, Chikusa-ku, Nagoya, 464–8601 Japan
†† Information Technology Center, Nagoya University
Furo-cho, Chikusa-ku, Nagoya, 464–8601 Japan
††† National Institute of Informatics,
2-1-2, Hitotsubashi, Chiyoda, Tokyo 101—0003, Japan
E-mail: †guoxi@db.itc.nagoya-u.ac.jp, ††ishikawa@itc.nagoya-u.ac.jp

Abstract　We propose a new problem called a *multi-objective optimal combination problem* (*MOC*) which finds out object combinations close to a given objective vector. A combination dominates another combination if it is not worse than anther one in all attributes and better than another one in one attribute at least. The optimal combinations are the ones which cannot be dominated by any other combinations. We propose an efficient algorithm to find out optimal combinations based on an R-tree by using a lower bound reduction method and an upper bound reduction method. Our experimental results show that the proposed algorithm is both effective and efficient.
Key words　Multi-objective, combination, decision-making, R-tree.

1. Introduction

In decision-making problems, we need objects which are optimal w.r.t. several objectives rather than a single objective. Such problems are categorized into *multi-objective optimization problems* [1] or *skyline query problems* [2]. In this paper, we propose a new variation which finds out optimal object combinations considering multiple objectives. We name it a *multi-objective optimal combination* (*MOC*) problem. Let us consider an example of the MOC problem first.

［Example 1］　Assume we want to synthesize a healthy food containing appropriate nutrition contents. A food is a mixture of several ingredients. Table 1 lists 25 available ingredients g_1 to g_{25} with their contents in the nutrition N_1 and the nutrition N_2. The nutrition contents in the synthesized food are aggregations of the nutrition contents in its ingredients. For example, the synthesized food $\{g_9, g_{10}, g_{21}\}$ has nutrition contents $(45, 13)$ which is the aggregations of the nutrition contents in $g_9 = (13, 3)$, $g_{10} = (14, 2)$ and $g_{21} = (18, 8)$.

Given an ideal nutrition contents $(50, 15)$, let us consider which food is healthier. Table 1 also shows six synthesized foods f_1 to f_6 consisting of three ingredients. Food $f_3 = (16, 22)$ is a bad one because it is beyond the requirement $(50, 15)$ in N_2. The other foods are not bad because they are within the requirement $(50, 15)$ in both N_1 and N_2.

Let us pick up the better ones then. Food $f_4 = (6, 11)$ is worse than $f_1 = (45, 13)$ because f_1 is closer to $(50, 15)$ than f_4 in both N_1 and N_2. We say that f_4 is dominated by f_1 and f_4 is not an optimal food. The optimal food should be a combination which cannot be dominated by any other combinations. Thus, f_1, f_2, f_5 and f_6 are optimal ones. ■

表 1 Ingredients And Synthesized Foods

Ingredients

g_i	N_1	N_2	g_i	N_1	N_2
g_1	2	3	g_{14}	6	13
g_2	3	5	g_{15}	2	14
g_3	1	3	g_{16}	12	11
g_4	4	2	g_{17}	10	14
g_5	1	5	g_{18}	15	14
g_6	7	3	g_{19}	13	12
g_7	9	9	g_{20}	11	16
g_8	12	8	g_{21}	18	8
g_9	13	3	g_{22}	12	18
g_{10}	14	2	g_{23}	4	17
g_{11}	7	10	g_{24}	15	9
g_{12}	2	9	g_{25}	17	12
g_{13}	4	11			

Foods

Food	N_1	N_2
$f_1\{g_9, g_{10}, g_{21}\}$	45	13
$f_2\{g_{10}, g_{10}, g_{21}\}$	46	12
$f_3\{g_5, g_9, g_{15}\}$	16	22
$f_4\{g_1, g_2, g_3\}$	6	11
$f_5\{g_9, g_9, g_{21}\}$	44	14
$f_6\{g_9, g_9, g_{24}\}$	41	15

Given an object set G where each object g has m attributes $(g^1, g^2, \cdots, g^m)$, we focus on combinations consisting of h objects. The attributes of a combination are attribute aggregations of its h elements. In other words, a combination $p = \{g_1, g_2, \cdots, g_h\}$ also has m attributes $(p^1, p^2, \cdots, p^m)$ where $p^j = \Sigma_{i=1}^{h} g_i^j$ $(j \in 1, 2, \cdots, m)$. A MOC problem is to find out good combinations which are close to an objective vector $\vec{b} = (b^1, b^2, \cdots, b^m)$. A combination p is closer to $\vec{b}$ than another combination p' if $b^k - p^k < b^k - p'^k$ $(k \in 1, 2, \cdots, m)$ and $b^j - p^j \leqq b^j - p'^j$ $(j \in 1, 2, \cdots, m$ and $j \neq k)$ where $p^i \leqq b^i$ and $p'^i \leqq b^i$. We also say that $\vec{p}$ *dominates* $\vec{p'}$. If a combination cannot be dominated by any other combinations, it is an *optimal combination.*

In this paper we focus on the h-MOC problem which refers to combinations consisting of h objects and the h is a fixed natural number given by a user. It is easy to extend the h-MOC problem to a general case which is to find out optimal combinations consisting of x objects where x is a natural number varying in $[1, N]$. The h-MOC problem, which we focus on in this paper, is a sub-problem of the general case. Obviously, we can obtain the solutions for the general case by solving sub-problems one by one. In the rest part of this paper, we simply call the h-MOC problems MOC problems for short without causing confusions.

A naïve method to find solutions for a MOC problem is to enumerate all possible combinations consisting of h objects and then determine whether they are dominated by any other combinations. The non-dominated combinations are determined to be the optimal combinations. However, this method is time-consuming. We propose an efficient algorithm to find out optimal combinations using the facilities of the R-tree index. An R-tree splits space by nested minimum bounding rectangles (MBR) and indexes them hierarchically [11]. We retrieve promising MBR combinations tier by tier to generate candidates for optimal combinations at the leaf tier. We use a reduction method to eliminate MBR combinations which are unpromising to generate optimal combinations. We only expand the promising MBR combinations using its child nodes. Thus, we can perform dominance tests on a small number of candidate combinations rather than on a huge number of possible combinations.

The rest of this paper is organized as follows. We first present some studies related to the proposed MOC problem in Section 2.. We give a formal definition of the MOC problem in Section 3.. Next, we talk about algorithms to answer MOC queries in Section 4.. In Section 5., we present some experimental results of the proposed algorithm and then conclude the paper in Section 6..

2. Related Work

In databases area, multi-objective optimization problems have received considerable attentions since the first work [2] proposed a skyline query problem. The skyline query problem aims at finding out optimal objects which cannot be dominated by any other objects. One object dominates another object if it is not worse than another one in all attributes and better than another one in one attribute at least. Many subsequent algorithms are proposed to improve performances of the skyline query, like BBS [8], SFS [12] and LESS [13]. Our MOC query problem is different from the classical skyline query problem because it focuses on object combinations rather than objects themselves. Though an object combination can be regarded as an object with aggregation attribute values of its elements, it is time consuming to use an existing algorithm to solve the MOC problem because there will be a huge number of object combinations to be processed.

The research of skyline queries on object combinations is limited. To the best of our knowledge, the first and only work on this topic is "top-k combinatorial skyline queries" [3]. This research was motivated by the investment portfolio which finds out optimal stock combinations considering profit and risk attributes. The authors studied how to find out top-k non-dominated combinations which rank from 1 to k before other non-dominated ones according to a given preference order in attributes. They constructed non-dominated combinations incrementally considering the preference order and terminates as soon as the top-k results have been found.

However, our MOC query problem simply focuses on finding out non-dominated combinations rather than a top-k query incorporating with any preference orders.

Our MOC query problem seems alike to the zero-one knapsack problem [14] which is in the linear integer programming category [6]. Suppose each object has a value attribute and a weight attribute, a knapsack problem is to find out the best object combination with a maximum total value and within a total weight limitation. The knapsack problem aims at optimizing the value attribute within a weight constraint. However, our MOC problem is to find out trade-offs between the value attribute and the weight attribute. For example, let us consider the combinations shown in Table **??**. Assume that N_1 is the value attribute and N_2 is the weight attribute. Among the six combinations, f_2 with a maximum value 46 is the best solution for a knapsack problem given a weight limitation 15. However, f_2, f_5 and f_6 are solutions for a MOC problem given an objective vector $(50, 15)$.

Another interesting work [4] focuses on selecting maximal combinations which consist of objects with a single attribute (e.g. price). A valid combination should have a total value (e.g. total price) within a single constraint (e.g. budget). It becomes a maximal one if it will be beyond the constraint by adding any new object to it. The proposed algorithms present k representative maximal combinations to the user which can generate the most sub-combinations. While our MOC query thinks about combinations consisting of objects with multiple attributes. Our objective vector can be regarded as multi-constraints like the single constraint in [4]. We construct optimal combinations which are closer to the multi-constraints.

In order to solve our MOC query problem, we organize objects using the R-tree index [11] and retrieve object combinations using a lower bound reduction method (Section 4.2). Our lower bound reduction method employs the basic idea of the forward checking (FC) algorithm [7] which constructs combinations incrementally to answer structural queries in spatial objects databases. A structural query asks for object combinations which have a spatial structure similar to a required structure.

3. Problem Definition

Given an object set G where each object has m attributes $(g^1, g^2, \cdots, g^m)$, a combination $p = \{g_1, g_2, \cdots, g_h\}$ consisting of h objects has attributes $(p^1, p^2, \cdots, p^m)$ where $p^j = \Sigma_{i=1}^{h} g_i^j$ $(j \in 1, 2, \cdots, m)$. Given an objective vector $\vec{b} = (b^1, b^2, \cdots, b^m)$, the distance from a combination p to $\vec{b}$ is $(d^1, \cdots, d^m)$ where $d^j = b^j - \Sigma_{i=1}^{h} g_i^j$. If $d^j \geqq 0$ for all j, the combination p is *eligible* to be an optimal combination.

[Definition 1] (Dominate) Given an objective vector $\vec{b}$, one eligible combination $\vec{p}$ *dominates* another eligible one $\vec{p'}$ if $d^k < d'^k (k \in 1..m)$ and $d^j \leqq d'^j (j \in 1..m$ and $j \neq k)$. □

[Definition 2] (Multi-Objective Optimal Combination) If a h-combination cannot be dominated by any other combinations $p_i \in P - \{p\}$ where $|p_i| = h$, it is *optimal.* □

[Problem 1] (MOC Query Problem) Given an object set G, an objective vector $\vec{b}$ and a combination cardinality h, a *MOC query problem* is to find out a combination set $S = \{s_1, s_2, \cdots, s_l\}$ where s_i $(i \in 1, 2, \cdots, l)$ is optimal. □

4. Algorithms

A naive method to solve the MOC query problem is to enumerate all combinations comprising h objects from the object set G, retain eligible combinations within a given objective vector $\vec{b}$, and then identify optimal ones which cannot be dominated any others. Obviously, the process is time-consuming because there may be a huge number of eligible combinations generated from G with respect to a moderate $\vec{b}$ and each eligible one needs comparisons with all the others to determine whether it is optimal or not.

4.1 Depth-First Combination Construction

Given objects indexed by an R-tree, we construct MBR combinations tier by tier and terminates at the leaf tier where the MBRs containing real objects. Each combination at tier i can be expanded by using its child MBRs and generate new combinations at tier $(i + 1)$. In this way, we can retrieve all possible object combinations using the R-tree structure.

[Example 2] Fig. 1 shows an R-tree index of objects in Table 1. Assume we want to construct combinations consisting of 3 elements. Starting from the root tier, we get 10 different MBR combinations $\{v_1v_2v_3 | v_1, v_2, v_3 \in \{A, B, C\}\}$. Each combination can be expanded to new combinations at the next tier by using their child MBRs. Let us take the combination ABC as an example. It can be expanded to 18 different combinations $\{v_1v_2v_3 | v_1 \in \{d, e\}, v_2 \in \{f, g, h\}, v_3 \in \{i, j, k\}\}$ at tier 2. Let us expand one combination dfi. It can be expanded to $\{v_1v_2v_3 | v_1 \in \{1, 2, 3, 5\}, v_2 \in \{14, 15, 23\}, v_3 \in \{17, 18, 20, 22\}\}$ at tier 3 which is also a leaf tier. At the leaf tier, the new generated combinations, like $\{1, 14, 17\}$, consist of real objects. At that time, we decide whether they are optimal or not by dominance tests. In other words, we have constructed an object combination following the path $\{A, B, C\} \to \{d, f, i\} \to \{1, 14, 17\}$ in a depth-first way. In such a depth-first way, we can construct all object combinations by retrieving a R-tree index. ■

Constructing combinations tier by tier on an R-tree provides us an opportunity to reduce the search space by eliminating unpromising MBR combinations. If we can eliminate unpromising MBR combinations before they are expanded to real object combinations, we may obtain fewer combination

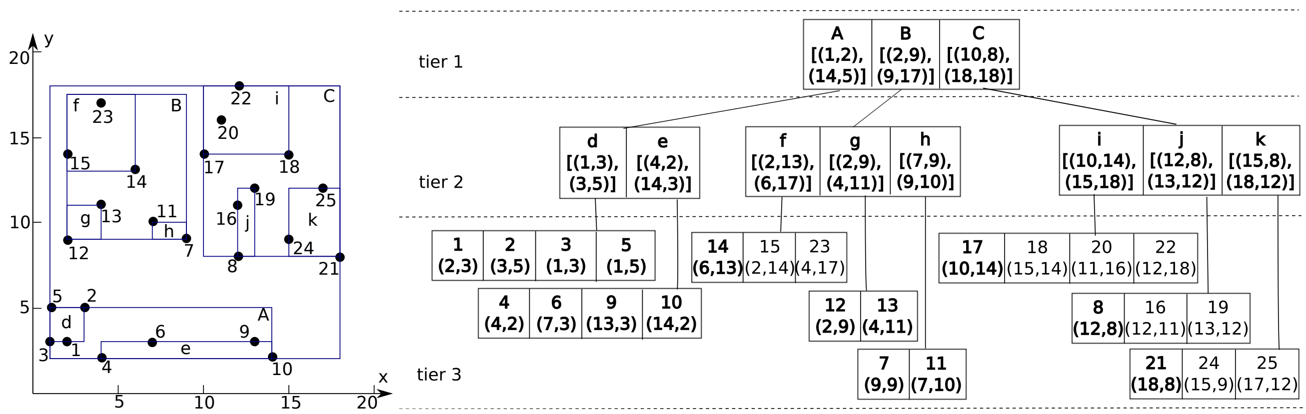

図 1　R-tree of objects in Table 1

candidates which need to be decided optimal or not at the leaf tier. A lower bound reduction method is proposed to eliminate the unpromising MBR combinations. The reduction method considers whether an MBR combination is an eligible one which should be within the objective vector $\vec{b}$.

4.2 Lower Bound Reduction Method

An MBR combination has a lower bound which is an aggregation on the lower bounds of its elements. For example, the combination ABC has a lower bound $ABC^{\vdash} = (13, 19)$ which is an aggregation on the lower bounds of its elements A, B and C, namely, $ABC^{\vdash} = A^{\vdash} + B^{\vdash} + C^{\vdash} = (1, 2) + (2, 9) + (10, 8)$. We define it formally as follows.

［Definition 3］(Lower Bound of An MBR Combination)　An MBR combination $p = \{e_1, e_2, \cdots, e_h\}$ has a *lower bound* $p^{\vdash}$ which is an aggregation on the lower bounds of its elements, namely, $p^{\vdash} = \Sigma_{i=1}^{h} e_i^{\vdash}$ where $e_i^{\vdash}$ is the lower bound of e_i. □

An MBR combination with a lower bound beyond an objective vector $\vec{b}$ cannot be expanded to object combinations within $\vec{b}$. An object combination beyond $\vec{b}$ is not eligible to be an optimal one. We conclude a theorem to eliminate such MBR combinations as follows.

［Theorem 1］　Given an objective vector $\vec{b} = (b^1, b^2, \cdots, b^m)$, an MBR combination p cannot be expanded to optimal object combinations, if its lower bound $p^{\vdash}$ beyond the objective vector $\vec{b}$, namely, $p^{i\vdash} > b^i$ $(i \in 1, 2, \cdots, m)$. □

［Proof 1］　We expand an MBR combination $p = \{e_1, e_2, \cdots, e_h\}$ using child MBRs of e_1 to e_h until we reach the leaf tier. In other words, we select objects enclosed in e_i $(i \in 1, 2, \cdots, h)$ to construct object combinations. Every object g_i selected from e_i has attribute values $g_i^j \geqq e_i^{j\vdash}$ $(j \in 1, 2, \cdots, m)$. An object combination consisting of these objects has attribute values $\Sigma_{i=1}^{h} g_i^j \geqq p^{j\vdash}$ where $p^{j\vdash} = \Sigma_{i=1}^{h} e_i^{j\vdash}$. If $p^{j\vdash} > b^j$, the combination is not eligible to be an optimal one because its attribute value $\Sigma_{i=1}^{h} g_i^j > b^j$. □

Let us consider an MBR combination ABC with a lower bound $(13, 19)$ as an example. Given an objective vector $(50, 15)$, we have to eliminate this combination because it is beyond $(50, 15)$ on the second attribute $(19 > 15)$. In other words, we will not expand ABC further.

Suppose we have an MBR combination $e_1 e_2 \cdots e_h$ and we want to expand it. The new combinations generated are denoted as $v_1 v_2 \cdots v_h$ $(v_1 \in C_1,\ v_2 \in C_2, \cdots,\ v_h \in C_h)$ where C_i is child MBRs of e_i. All combinations are enumerated by instantiating variables v_1 to v_h w.r.t. C_1 to C_h. For example, the new combinations generated by expanding AAB are $v_1 v_2 v_3$ $(v_1 \in \{d, e\},\ v_2 \in \{d, e\},\ v_3 \in \{f, g, h\})$. According to Theorem 1, we eliminate the combinations which are not eligible to generate optimal object combinations. We propose a method called *forward checking* [7] to incrementally generate eligible MBR combinations with lower bounds within $\vec{b}$ while expanding a parent MBR combination $e_1 e_2 \cdots e_h$.

While expanding a parent MBR combination $e_1 e_2 \cdots e_h$, we instantiate variables v_1 to v_h using child MBRs C_1 to C_h. After instantiating variables $v_1 v_2 \cdots v_{l-1}$ as $c_1 c_2 \cdots c_{l-1}$, we say that the process is at the l^{th} *instantiation level* where we will instantiate v_l. Feasible MBRs for v_l should have a lower bound smaller than a threshold $T = \vec{b} - \Sigma_{i=1}^{l-1} c_i^{\vdash}$. If we choose an MBR with a lower bound beyond T, the final obtained combination will be not eligible. For example, given an objective vector $\vec{b} = (50, 15)$, let us expand a parent combination AAB. Assume that we have instantiated variables $v_1 v_2$ as dd. Now we are at the 3^{rd} instantiation level and we need to select an MBR from B's child MBRs $\{f, g, h\}$ to instantiate v_3. The threshold T of selecting MBR for v_3 is $(50, 15) - (1, 3) - (1, 3) = (48, 9)$. If we select MBR f with a lower bound $(2, 13)$ beyond $(48, 9)$, we will obtain an ineligible combination ddf (i.e. $ddf^{\vdash} = (4, 19)$). Thus, we should use either g or h to instantiate v_3 which has lower bounds within T. Let us use d_{li} to denote the domain for instantiating v_i at an instantiation level l. According to $T = (48, 9)$, we can decide $d_{33} = \{g, h\}$ and avoid generating ddf.

The basic idea of *forward checking* is to update domains for an instantiation level l according to a threshold T_l which is decided by $c_1 c_2 \cdots c_{l-1}$ where c_i is an MBR instance for v_i.

While expanding $e_1e_2\cdots e_h$, we initialize d_{1i} $(i \in 1,2,\cdots,h)$ for each variable v_i using C_i. At the 1^{st} level, we instantiate v_1 as $c_1 \in d_{11}$ and get a threshold $T_2 = \vec{b} - c_1^{\vdash}$ for the next level. According to T_2, we get domains d_{2i} $(i \in 2,3\cdots,h)$ by eliminating MBRs from d_{1i}. The eliminated ones have lower bounds beyond T_2. Next, at the 2^{nd} level, we instantiate v_2 as $c_2 \in d_{22}$ and get a threshold $T_3 = \vec{b} - \Sigma_{i=1}^{2} c_i^{\vdash}$. According to T_3, we get domains d_{3i} $(i \in 3,4\cdots,h)$. In this way, we instantiate variables one by one avoid generating ineligible MBR combinations.

[Example 3] Table 2 shows the process of expanding AAB using the forward checking method. Let us consider step 0 to step 3. At step 0, we instantiating d_{11}, d_{12} and d_{13} using child MBRs $\{d,e\}$ w.r.t. A and child MBRs $\{f,g,h\}$ w.r.t. B. At the 1^{st} instantiation level (step 1), we instantiate v_1 using $d \in d_{11}$ and get a threshold T_2. According to T_2, we prepare domains d_{22} and d_{23} for the next level. We eliminate f from d_{23}, namely, $d_{23} = d_{13} - \{f\}$, because $f^{\vdash} = (2,13)$ beyond T_2. At the 2^{nd} level (step 2), we instantiate v_2 using $d \in d_{22}$ and get a threshold T_3. According to T_3, we get domain d_{33} for the next level which is the same with d_{23} because no MBR is beyond T_3. At the 3^{rd} level (step 3), we instantiate the last variable v_3 using $g \in d_{33}$ and obtain a complete combination ddg. ■

表 2 Expand AAB Using Lower Bound Reduction

Step	Instantiate	Threshold	Domains	Combination
0		$T_1 = (50,15)$	$d_{11} = \{d,e\}$ $d_{12} = \{d,e\}$ $d_{13} = \{f,g,h\}$	$v_1v_2v_3$
1	$v_1 \leftarrow d \in d_{11}$	$T_2 = (49,12)$	$d_{22} = \{d,e\}$ $d_{23} = \{g,h\}$	dv_2v_3
2	$v_2 \leftarrow d \in d_{22}$	$T_3 = (48,9)$	$d_{33} = \{g,h\}$	ddv_3
3	$v_3 \leftarrow g \in d_{33}$			ddg
4	$v_3 \leftarrow h \in d_{33}$			ddh
5	$v_2 \leftarrow e \in d_{22}$	$T_3 = (45,10)$	$d_{33} = \{g,h\}$	dev_3
6	$v_3 \leftarrow g \in d_{33}$			deg
7	$v_3 \leftarrow h \in d_{33}$			deh
8	$v_1 \leftarrow e \in d_{11}$	$T_2 = (46,13)$	$d_{22} = \{d,e\}$ $d_{23} = \{f,g,h\}$	ev_2v_3
9	$v_2 \leftarrow d \in d_{22}$	$T_3 = (45,10)$	$d_{33} = \{g,h\}$	edv_3
10	$v_3 \leftarrow g \in d_{33}$			edg
11	$v_3 \leftarrow h \in d_{33}$			edh
12	$v_2 \leftarrow e \in d_{22}$	$T_3 = (42,11)$	$d_{33} = \{g,h\}$	eev_3
13	$v_3 \leftarrow g \in d_{33}$			eeg
14	$v_3 \leftarrow h \in d_{33}$			eeh

At the last level h^{th}, we can obtain a complete combination by instantiating the last variable v_h. It is necessary to backtrack to the partial combination $c_1c_2\cdots c_{h-1}v_h$ to see whether there exist other MBRs in d_{hh} which can be used to instantiate v_h. If so, we use these MBRs to instantiate v_h and obtain other complete combinations. When MBRs in d_{hh} have been used up, we backtrack to the partial combination $c_1c_2\cdots c_{h-2}v_{h-1}v_h$ at the previous level $(h-1)^{th}$. We instantiate v_{h-1} using the unused MBRs in $d_{h-1,h-1}$ and repeat the forward checking process for variables $v_{h-1}v_h$. The whole process terminates when MBRs in d_{11} have been used up. At that time, we find out all eligible combinations .

[Example 4] Let us continue with the Example 3 and consider the backtrack process starting from step 4. After obtaining ddg at step 3, there exists an unused MBR h in d_{33}. We can use h to instantiate v_3 and obtain another complete combination ddh. After step 4, all MBRs in d_{33} have been used up. We backtrack to the partial combination dv_2v_3 at the 2^{nd} level and use another MBR to instantiate v_2 instead of d which we have used before. As step 5 shows, we use another MBR $e \in d_{22}$ to instantiate v_2 and repeat the forward checking process which is preparing a new d_{33} for v_3 according to a new T_3. One can follow the rest steps and obtain all eligible combinations as Table 2 shows. ■

Note that there are duplicate combinations generated during the lower bound reduction process. Two combinations are duplicates if they have same elements regardless of their element orders. As Table 2 shows, the finally generated combinations are ddg, ddh, deg, deh, edg, edh, eeg and eeh. Combinations deg with edg and combinations deh with edh are duplicates. It is easy to remove such duplicates and we will not talk it too much for the space limitation.

Algorithm 1 concludes the process of MOC queries using the lower bound reduction method. We start a query process by calling a function MOC_query$(p,\vec{b},h,S)$ where $p = \{root,\ root,\ root\}$ and $S = \emptyset$. We first initialize the threshold T as $\vec{b}$, initialize d_{1i} $(i \in 1,2,\cdots,h)$ as child MBRs of e_i using a function get_children(e_i), and initialize the current instantiation level identifier l as 1 (from line 3 to 6). Next, we expand the combination p (line 7 to line 30).

From line 9 to 18, we instantiate the variable v_l. We select an MBR from d_{ll} to instantiate v_l using a function get_MBR(d_{ll}) (line 10). The function get_MBR(d_{ll}) removes the selected MBR from d_{ll}. If d_{ll} is empty, we backtrack to the level $(l-1)$ (line 12 to 18). Note that we will not do the backtrack operation if the current level is 1 (line 12 to 13).

From line 19 to 24, we prepare domains for the next instantiation level $(l+1)$ using the function forward_check(). After updating the threshold T considering the instantiated variables (line 21), we call a function forward_check(T,l,i) (line 31 to 38). In the function, we initialize domains $d_{l+1,j}$ $(j \in i+1,i+2,\cdots,h)$ as domains $d_{l,j}$ at the previous level l. We check each MBR in $d_{l+1,j}$ and remove the ones which have lower bounds beyond T (line 35 to 37).

In the function MOC_query(), if we are not expanding a combination at the leaf tier, we recursively call the function MOC_query() to expand a newly generated combination p' (line 29). If not, we update the optimal object combination set S (line 27). A function update_optimal_set(p', S) decides whether a new object combination p' can be dominated by an existing combination in S. We add it into S, if it cannot be dominated by any combinations in S. We also removes the combinations in S which is dominated by p'.

5. Experiments

We implemented Algorithm 1 in GNU C++ and conducted experiments on an Intel Core2 Duo 2.40 GHz PC (2.0 GB RAM) with a Fedora 12 Linux 2.6.32. The algorithm was implemented based on a R-tree interface provided by a spatial index library SaIL ([9], [10]). The R-tree has a block size 8,192 bytes and a fill factor 70%.

We evaluated performances of Algorithm **??** with three experimental sets. The first set evaluated the algorithm with respect to different data distributions, say, independent distribution, correlated distribution, and anti-correlated distribution. The second set evaluated the algorithm with respect to different m's where m is the number of attributes. The third set evaluated the algorithm with respect to different h's where h is the number of objects in a combination. We will show the experimental results of the three sets in Section 5. 1, Section 5. 2, and Section 5. 3 respectively.

5. 1 Performances with Different Data Distribution

When we evaluate algorithm performances with different data distributions, we use five synthetic data sets $D_{0.6}$, $D_{0.9}$, $D_{-0.6}$, $D_{-0.9}$ and $D_{0.0}$ with different correlation coefficients 0.6, 0.9, -0.6, -0.9 and 0.0. We generated these data sets using the method in [2]. Each data set has 100 objects with two attributes ranging from 0 to 1000. We use 15 different objective vectors $\vec{b_1}$ to $\vec{b_{15}}$ to evaluate the algorithm. Each objective vector $\vec{b_i}$ ($i \in 1, 2, \cdots, 15$) has attribute values (b_i^1, b_i^2) where $b_i^1 = b_i^2 = 400 + 200 \times i$. Given objective vectors $\vec{b_1}$ to $\vec{b_{15}}$, we executed MOC queries to find out optimal combinations consisting of 3 objects on five data sets $D_{0.6}$, $D_{0.9}$, $D_{-0.6}$, $D_{-0.9}$ and $D_{0.0}$.

Fig. 3 (a), Fig. 4 (a) and Fig. 5 (a) show the number of optimal combinations found w.r.t. different objective vectors $\vec{b_1}$ to $\vec{b_{15}}$. The vertical axis represents the number of optimal combinations and the horizontal axis represents $\vec{b_1}$ to $\vec{b_{15}}$. Fig. 3 (b), Fig. 4 (b) and Fig. 5 (b) show algorithm performances. The left vertical axis is the CPU cost with second unit in a log scale. The right vertical axis is the number of checked MBR combinations (CMC) also in a log scale.

Let us consider the number of optimal combinations varying with different objective vector $\vec{b}$'s (Fig. 3 (a), Fig. 4 (a) and Fig. 5 (a)). It increases first, reaches a peak value, and then falls down. In the beginning, the $\vec{b_i}$ has small attribute values (e.g. $\vec{b_1} = (600, 600)$). Thus, we have to use objects with small attribute values to construct combinations (e.g. objects in the area $[(0,0), (600,600)]$). While the $\vec{b_i}$ grows, we can use more objects in a larger area and more optimal

Algorithm 1 MOC Query Using Lower Bound Reduction

1: **procedure** MOC_query($p, \vec{b}, h, S$) {$p = e_1 e_2 \cdots e_h$ is a combination to be expanded; S contains optimal object combinations.}
2: $p' := v_1 v_2 \cdots v_h$; {Expand p to p' which have h variables to instantiate.}
3: $T := \vec{b}$; {Initialize threshold T as $\vec{b}$.}
4: **for** $i := 1$ **to** h **do**
5: $d_{1i} :=$ get_children(e_i); {Initialize domains d_{1i}.}
6: $l := 1$; {Start from the 1^{st} instantiation level.}
7: **while** *true* **do**
8: **begin**
9: **if** $d_{ll} \neq \emptyset$ **then** {MBRs in d_{ll} are not used up.}
10: $v_l :=$ get_MBR(d_{ll}); {Select an MBR from d_{ll} to instantiate v_l.}
11: **else** {MBRs in d_{ll} are used up.}
12: **if** $l = 1$ **then**
13: return; {Terminate the expansion of p.}
14: **else**
15: **begin**
16: $l := l - 1$;
17: continue; {Backtrack to level $(l-1)$.}
18: **end**
19: **if** $l < h$ **then** {At a level before the last level h.}
20: **begin**
21: $T := T - v_l^{\vdash}$; {Update T.}
22: forward_check(T, l, i); {Prepare domains for level $(l+1)$.}
23: $l := l + 1$; {Start the instantiation for level $(l+1)$}
24: **end**
25: **else** {At the last level h.}
26: **if** at_leaf_tier(p) **then**
27: update_optimal_set(p', S); {Update S considering p'.}
28: **else**
29: MOC_query($p', \vec{b}, h, S$); {Expand p'.}
30: **end**
31: **procedure** forward_check(T, l, i)
32: **for** $j := i + 1$ **to** h **do**
33: **begin**
34: $d_{l+1,j} = d_{l,j}$; {Initialize domains at level $l+1$.}
35: **for** $k := 1$ **to** n **do** {$d_{l+1,j} = \{c_k | k \in 1, 2, \cdots, n\}$.}
36: **if** is_beyond($c_k^{\vdash}, T$) **then** {$c_k^{\vdash}$ is beyond T.}
37: $d_{l+1,j} := d_{l+1,j} - \{c_k\}$; {Eliminate c_k from $d_{l+1,j}$.}
38: **end**

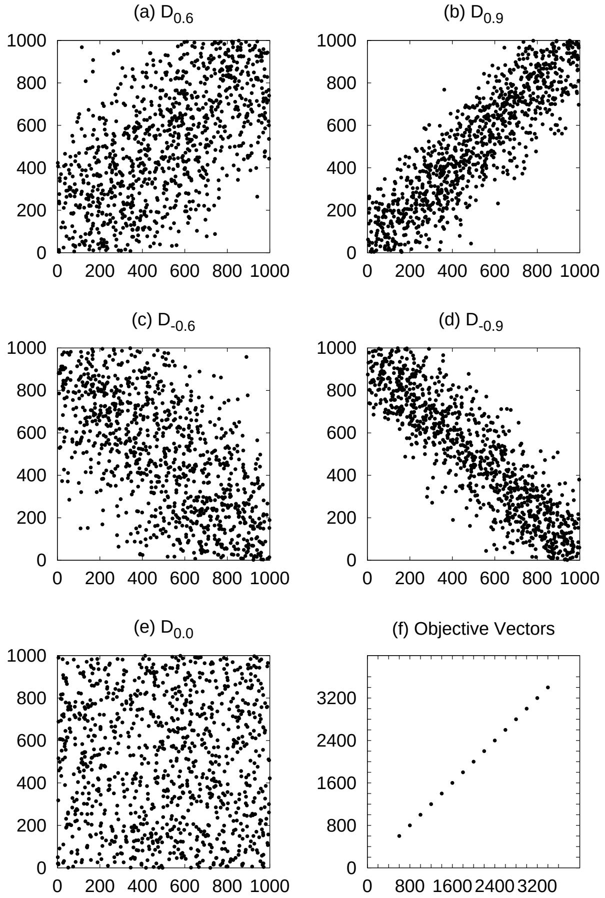

図 2 Object Distributions and Objective Vectors

combinations are found. When the $\vec{b_i}$ increases to be moderate (e.g. $\vec{b_6} = (1600, 1600)$), we can use objects in the whole area to construct combinations. At that time, the number of optimal combinations reaches a peak value. From then on, the $\vec{b}$ continues increasing while the area for selecting objects does not enlarge. In order to construct an optimal combination, we have to use objects which are close to the $\vec{b}$. The number of optimal combinations falls down and reach a constant value at last (e.g. $\vec{b_{13}}$ $\vec{b_{14}}$ $\vec{b_{15}}$).

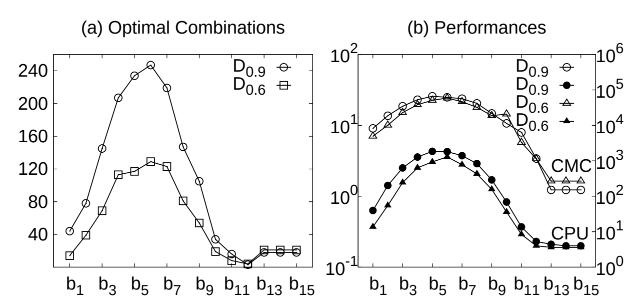

図 3 Results and Performances on Correlated Data Set

Let us consider the number of optimal combinations varying with different correlated degrees. In Fig. 3 (a), the high correlated data set $D_{0.9}$ has more optimal combinations than the low correlated data set $D_{0.6}$. Fox example, the objects in $D_{0.9}$ are more concentrated along the objective vectors $\vec{b_1}$ to $\vec{b_{15}}$ than the objects in $D_{0.6}$. Thus, we can use more objects to construct optimal combinations when we execute MOC queries on $D_{0.9}$. Given the $\vec{b}$ as $(600, 600)$, we can use more objects because there are more objects in the area $[(0, 0), (600, 600)]$ in $D_{0.9}$. On the other hand, an object close to the $\vec{b_i}$ is easier to become an element of an optimal combination. Likewise, we can understand the varying numbers of optimal combinations on $D_{-0.9}$ and $D_{-0.6}$ in Fig. 4 (a).

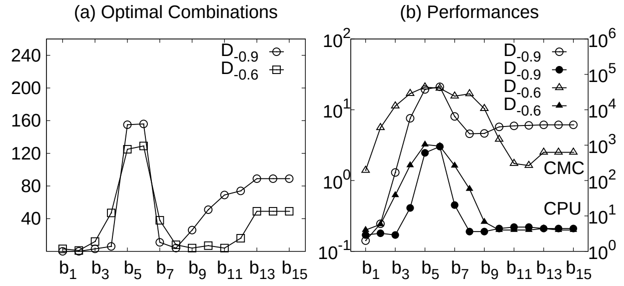

図 4 Results and Performances on Anti-Correlated Data Set

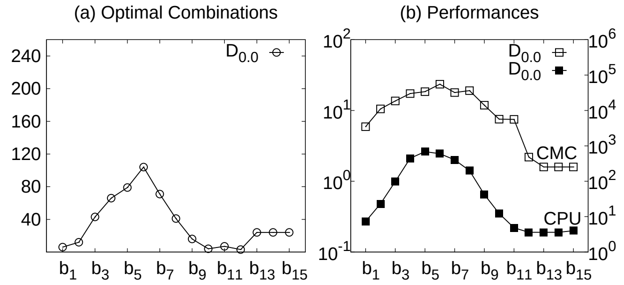

図 5 Results and Performances on Uniform Data Set

Fig. 3 (b), Fig. 4 (b) and Fig. 5 (b) show the algorithm performances. The CPU cost depends on how many MBR combinations (CMC) we have checked during the MOC queries. The number of CMCs increases first, reaches a peak value, and then decreases. More CMCs are generated w.r.t. a relative larger $\vec{b_i}$. When we can use objects in the whole area, fewer CMCs are generated w.r.t. a relative larger $\vec{b_i}$. The reasons are the same with what we have stated above for explaining the number of optimal combinations.

5.2 Performances with Different Attribute Number m

When we evaluate algorithm performances with different attribute number m, we use three data sets D_2, D_3 and D_4 where $m = 2$, $m = 3$ and $m = 4$ respectively. The objects in the three data sets follow uniform distributions. Each data set contains 100 objects with attribute values ranging from 0 to 1000. We use 15 objective vectors $\vec{b_i}$ $(i \in 1, 2, \cdots, 15)$ where $b_i^1 = b_i^2 = \cdots = b_i^m = 400 + 200 \times i$ $(m = 2, 3, 4)$. Given the objective vector $\vec{b_i}$, we execute MOC queries on

D_2, D_3 and D_4 in order to find out optimal combinations consisting of 3 objects.

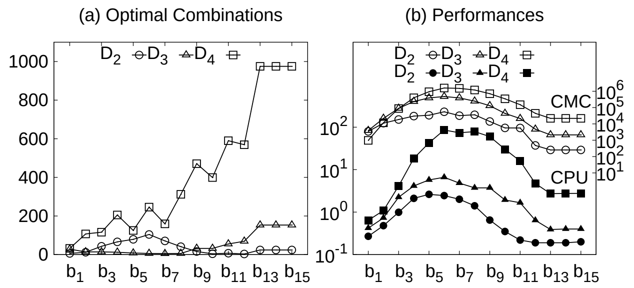

図 6 Results and Performances on Data Sets D_2, D_3 and D_4

Fig. 6 (a) shows the number of optimal combinations on data sets D_2, D_3 and D_4. The vertical axis represents the number and the horizontal axis represents objective vectors $\vec{b_1}$ to $\vec{b_{15}}$. The data set with a larger m (e.g. D_4) has more optimal combinations than the data set with a smaller m (e.g. D_2) because it is difficult for one combination dominates another combination if there are more attributes to compare.

The Fig. 6 (b) shows the algorithm performances on data sets D_2, D_3 and D_4. The left vertical axis represents CPU cost with a second unit in a log scale while the right vertical axis represents the number of CMCs also in a log scale. The CPU cost depends on the number of CMCs. The data set with a larger m (e.g. D_4) checks more MBR combinations than the data set with a smaller m (e.g. D_2) because the R-tree has more MBRs in a high-dimensional space.

5.3 Performances with Different Cardinality h

When we evaluate algorithm performances with different number of objects in a combination, say, different h's, we use the uniform distribution data set $D_{0.0}$. Given the objective vector $\vec{b} = (500, 500)$, we execute MOC queries to find out optimal combinations.

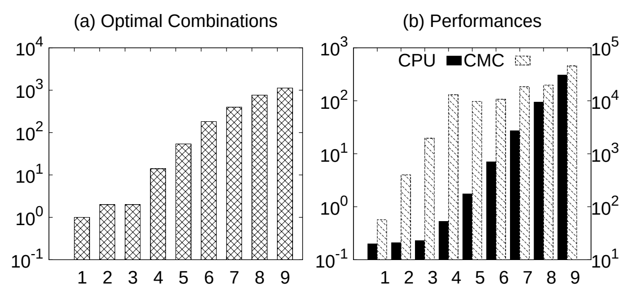

図 7 Results and Performances on Data Sets $D_{0.0}$ with $h = 1, 2, \cdots, 9$

Fig. 7 (a) shows the number of optimal combinations with different h's. The horizontal axis represents the h from 1 to 9 and the vertical axis represents the number in a log scale. The number increases while h increases because a same object set can generate more object combinations with a larger cardinality (e.g. $h = 9$).

Fig. 7 (b) shows the algorithm performances with different h's. The left vertical axis represents the CPU cost while the right vertical axis represents the number of CMCs. The CPU cost depends on the number of CMCs as well as the number of candidates. The number of CMC grows with h because a same R-tree can generate more MBR combinations which have a larger cardinality (e.g. $h = 9$). At the leaf tier of the R-tree, we decide whether a popped candidate object combination is an optimal one. It takes much more time to do dominance tests for a larger number of candidates due to a larger cardinality h.

6. Conclusions

In this paper, we propose a new multi-objective optimization problem called MOC problem which is to find out optimal combinations w.r.t. an objective vector $\vec{b}$. We propose the lower bound reduction and the upper bound reduction methods to answer MOC queries efficiently.

Acknowledgments

This research is partly supported by the Grant-in-Aid for Scientific Research, Japan (21013023, 22300034).

文 献

[1] K. Deb, “Multi-objective optimization using evolutionary algorithms, ” pp. 13-46, John Wiley and Sons, 2001.
[2] S. Börzsönyi, D. Kossmann, and K. Stocker, “The Skyline Operator,” ICDE, pp. 421-430, 2001,
[3] I.-F. Su, Y.-C. Chung, and C. Lee, “Top-k Combinatorial Skyline Queries,” DASFAA, pp. 79-93, 2010.
[4] S.B. Roy, S.A. Yahia, A. Chawla, G. Das, and C. Yu, “Constructing and Exploring Composite Items,” SIGMOD, pp. 843-854, 2010.
[5] D. Papadias, N. Mamoulis, and V. Delis, “Algorithms for Querying by Spatial Structure,” VLDB, pp. 546-557, 1998.
[6] D. Bertsimas, J. N. Tsitsiklis, “Introduction to Linear Optimization,” pp. 451-531, 1997.
[7] D. Papadias, N. Mamoulis and V. Delis, “Algorithms for Querying by Spatial Structure,” VLDB, pp. 546-557, 1998.
[8] D. Papadias, Y. Tao, G. Fu and B. Seeger, “Progressive skyline computation in database systems,” ACM Trans. Database Syst, 30(1), pp. 41-82, 2005.
[9] M. Hadjieleftheriou, E. Hoel and V. J. Tsotras, “SaIL: A Spatial Index Library for Efficient Application Integration,” Geoinformatica, 9(4), pp.367-389, 2005.
[10] M. Hadjieleftheriou, “Spatial Index Library (SaIL)”, `http://www2.research.att.com/~marioh/spatialindex/`.
[11] A. Guttman, “R-trees: A Dynamic Index Structure for Spatial Searching,” SIGMOD, pp. 47-57, 1984.
[12] J. Chomicki, P. Godfrey, J. Gryz and D. Liang, “Skyline with Presorting,” ICDE, pp. 717-719, 2003.
[13] P. G. Ryan, R. Shipley, and J. Gryz, “Maximal Vector Computation in Large Data Sets,” VLDB, pp. 229-240, 2005.
[14] Wikipedia, “Knapsack Problem” `http://en.wikipedia.org/wiki/Knapsack_problem`.