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
guoxi@db.itc.nagoya-u.ac.jp, ishikawa@itc.nagoya-u.ac.jp
http://www.db.itc.nagoya-u.ac.jp/en/

Abstract. Multi-objective optimization problem finds out optimal objects w.r.t. several objectives rather than a single objective. We propose a new problem called a *multi-objective optimal combination problem* (*MOC problem*) which finds out object combinations w.r.t. multiple objectives. 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 combinations, which cannot be dominated by any other combinations, are optimal. We propose an efficient algorithm to find out optimal combinations by reducing the search space with a lower bound reduction method and an upper bound reduction method based on the R-tree index. We implemented the proposed algorithm and conducted experiments on synthetic data sets.

Keywords: Multi-objective, combination, R-tree, domination.

1 Introduction

The *multi-objective optimization problem* [1, 2] finds out objects which are optimal w.r.t. several objectives rather than a single objective. In this paper, we propose a new variation which finds out optimal object combinations w.r.t. multiple objectives. We name it a *<u>m</u>ulti-objective <u>o</u>ptimal <u>c</u>ombination* (*MOC*) problem. Let us consider an example first.

Example 1 *A user wants to buy a breakfast consisting of three foods. Her budget is 1300JPY and her calorie demand is 1600kcal. Assume that six different foods are available in Fig. 1 (a)*[1]*. All 3-item food combinations are shown in Fig. 1 (b) with (cost, calorie) and are also shown in Fig. 1 (c) as points. We need to recommend better combinations for her.*

The combinations $\{AED\}$, $\{ACB\}$, $\{AEB\}$, $\{ACE\}$, $\{CED\}$, $\{AEF\}$ *and* $\{CEB\}$ *are within the user's requirements* $(13, 16)$ *as Fig. 1 (c) shows. They are possible answers for the user. The combination* $\{CEB\} = (13, 14)$ *is better than the combination* $\{ACE\} = (9, 12)$ *because* $\{CEB\}$ *is closer to the requirements* $(13, 16)$. *We say that* $\{CEB\}$ *dominates* $\{ACE\}$.

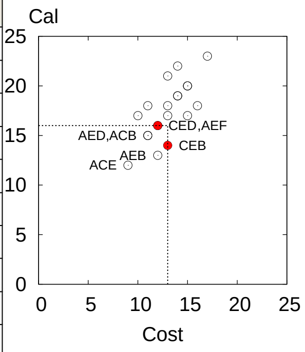
[1] The cost unit is 100 JPY and the calorie unit is 100kcal.

Food	(Cost, Cal)
A	(2,4)
B	(6,6)
C	(3,5)
D	(5,8)
E	(4,3)
F	(6,9)

(a) Food

Comb'n	(Cost, Cal)	Comb'n	(Cost, Cal)
ACB	(11,15)	ABF	(14,19)
ACD	(10,17)	ACF	(11,18)
AEB	(12,13)	ADF	(13,21)
AED	(11,15)	AEF	(12,16)
CEB	(13,14)	BCF	(15,20)
CED	(12,16)	BDF	(17,23)
ABD	(13,18)	BEF	(16,18)
CBD	(14,19)	CDF	(14,22)
EBD	(15,17)	CEF	(13,17)
ACE	(9,12)	DEF	(15,20)

(b) Three-Foods Combination



(c) Optimal Food

Fig. 1. MOC Problem Example

Suppose there combinations, which cannot be dominated by any other combinations, are optimal ones to be recommended. In this example, $\{CED\}$, $\{AEF\}$ and $\{CEB\}$ (solid points) cannot be dominated. We return them to the user as the results. The challenge of the problem is that there will be a huge number of combinations consisting of elements from a given object set and we need to identify the optimal combinations. This example is going to act as a running example in the rest of this paper.

■

There is an object set G where each object has m attributes $(g^1, g^2, \cdots, g^m)$. An h-item combination $p = \{g_1, g_2, \cdots, g_h\}(g_i \in G)$ 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 $\boldsymbol{b} = (b^1, b^2, \cdots, b^m)$, the *distance* from a combination p to $\boldsymbol{b}$ is $(d^1, \cdots, d^m)$ where $d^j = b^j - \Sigma_{i=1}^{h} g_i^j$. If $d^j \geq 0$ for all j, the combination p is *eligible* to be an optimal combination.

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

Definition 2 (Multi-Objective Optimal Combination) *If an h-item combination cannot be dominated by any other combinations $p_i \in P - \{p\}$, it is a multi-objective optimal combination (MOC).* □

Problem 1 (MOC Query) *Given an object set G, an objective vector* $\boldsymbol{b}$ *and a combination cardinality h, an* MOC query *finds out the MOC set $S = \{s_1, s_2, \cdots, s_l\}$ where s_i $(i \in 1, 2, \cdots, l)$ is an optimal combination consisting of h objects.* □

A naïve method to solve the MOC problem is to enumerate all possible h-item combinations and decide whether they are dominated or not. The non-dominated ones are returned as optimal combinations. However, this method is very time-consuming. In this paper, we propose an efficient algorithm to find out optimal ones using a lower bound and an upper bound reduction methods. The two reduction methods are based on the R-tree which indexes objects using hierarchical minimum bounding rectangles (MBRs) [11]. We construct combinations by searching an R-tree in a depth-first way. Considering lower bounds and upper bounds of MBRs, we reduce the search space and obtain candidates quickly. Finally, we find out the optimal ones from the candidates.

We first review some studies related to the proposed MOC problem in Section 2. Next, we propose the algorithm to answer MOC queries in Section 3. In Section 4, we report experimental results and conclude the paper in Section 5.

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 the performances of skyline queries, like BBS [8], SFS [12] and LESS [13]. Our MOC query problem, however, 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 with some preference orders.

One may think that our MOC query problem seems alike to the zero-one knapsack problem [14] which is in the linear integer programming category [6]. Given each object has a value attribute and a weight attribute. A knapsack problem finds 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.

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 and an upper bound reduction method. 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 databases. A structural query asks for object combinations which have a spatial structure similar to a required structure. Our upper bound reduction method employs the basic idea of the BBS algorithm [8] which is an efficient solution for classical skyline queries [2] on objects rather than on object combinations in our MOC query problem.

3 Algorithms

Given objects indexed by an R-tree, we construct MBR combinations in a depth-first way until reaching the leaf level where the MBRs are real objects. Each MBR combination can be expanded using its child MBRs. Let us use Example 2 to illustrate how to retrieve combinations using the R-tree index.

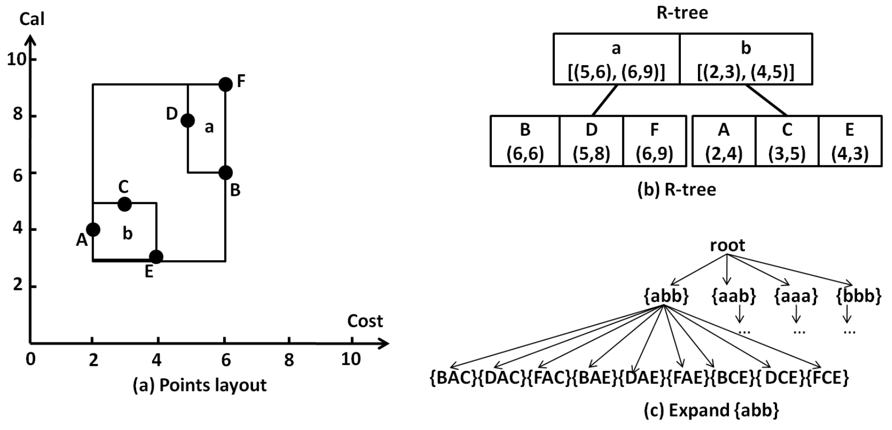


Fig. 2. Construct combinations using R-tree

Example 2 *Fig. 2 (a) and Fig. 2 (b) show the R-tree index of objects in the running example. Let us construct 3-item combinations using the R-tree. There are two MBRs a and b at the root level. We can select one object from MBR a and select two objects from MBR b to construct a 3-item combination. For simple, we use MBR combination {abb} to denote this selection pattern. As Fig. 2 (c) shows, we can expand pattern {abb} using objects involved in MBR a and objects involved in MBR b. Nine combinations (i.e., {BAC} to {FCE}) are obtained following pattern {abb}. In the same way, we can generate object combinations following patterns {aab}, {aaa} and {bbb}.* ■

Example 2 illustrates that we can construct h-item combinations easily by retrieving the R-tree in a depth-first way. The depth-first retrieval provides us an opportunity to reduce the search space by eliminating non-promising MBR combinations (i.e., patterns). If we can eliminate non-promising MBR combinations before they are expanded to real object combinations, we need fewer comparisons for object combinations at the leaf level. A lower bound reduction method and an upper bound reduction method are proposed to eliminate the non-promising MBR combinations.

3.1 Lower Bound Reduction

An MBR combination has a lower bound which is an aggregation on the lower bounds of its elements. For example, in Figure 2 the combination $\{abb\}$ has a lower bound $\{abb\}^{\perp} = (9, 12)$ which is an aggregation on the lower bounds of its elements one a and two b, namely, $\{abb\}^{\perp} = a^{\perp} + b^{\perp} \times 2 = (5, 6) + (2, 3) \times 2$. We define it formally as follows.

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

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

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 level. 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 $g_i^j \geq e_i^{j\perp}$ $(j \in 1, 2, \cdots, m)$. An object combination consisting of these objects has attribute values $\Sigma_{i=1}^{h} g_i^j \geq p^{j\perp}$ where $p^{j\perp} = \Sigma_{i=1}^{h} e_i^{j\perp}$. If $p^{j\perp} > b^j$, the combination is not eligible to be an optimal one because its attribute value $\Sigma_{i=1}^{h} g_i^j > b^j$.* ■

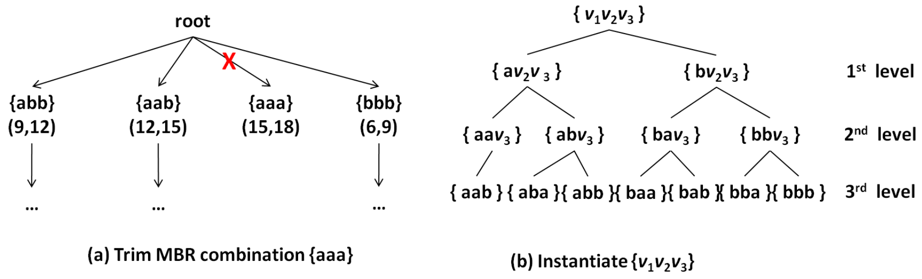


Fig. 3. Lower Bound Reduction

Example 3 *Let us think about constructing 3-item combinations again. Fig. 3 (a) shows the lower bounds of MBR combinations. Given an objective vector $(13, 16)$, we prune the MBR combination $\{aaa\}$ because it has a lower bound $[15, 18]$ which is beyond of $(13, 16)$. Combination $\{aaa\}$ will not be expanded.* ■

In order to generate MBR combinations with lower bounds within the objectives, we incrementally obtain them using a method inspired by the *forward checking* algorithm used in [7]. An h-item combination can be denoted as $v_1v_2\cdots v_h$ ($v_1 \in C_1,\ v_2 \in C_2, \cdots,\ v_h \in C_h$) where C_i is the domain of v_i. We instantiate variable v_i by selecting an MBR or an object from domain C_i. Our instantiation order is from v_1 to v_h. We obtain a combination $c_1c_2\cdots c_h$ after instantiating v_h at last.

During the process, after instantiating an intermediate variable v_{l-1}, we obtain a partial combination $c_1c_2\cdots c_{l-1}v_l\cdots v_h$ where $c_i \in C_i$ and $i \in [1, l-1]$. The process is at the l^{th} *instantiation level* where v_{l-1} has been instantiated while v_l needs to be instantiated. The domain C_l for v_l is decided by the current partial combination $c_1c_2\cdots c_{l-1}v_l\cdots v_h$. The MBRs in domain C_l should have lower bounds within $T = \boldsymbol{b} - \Sigma_{i=1}^{l-1} c_i^{\perp}$.

Example 4 *Fig. 3 (b) shows the process of instantiating a combination* $v_1v_2v_3$*. Let us take the leftmost branch as an example. At the* 1^{st} *level, we instantiate* v_1*. Given the objective vector* $(13, 16)$*, domain* C_1 *is* $\{a, b\}$*. After setting MBR* a *to variable* v_1*, we obtain a partial combination* $\{av_2v_3\}$*. Next, at the* 2^{nd} *level, we instantiate* v_2 *and objects belongs to* C_2 *should have lower bounds within* $(8, 10) = (13, 16) - (5, 6)$ *where* $a^{\perp} = (5, 6)$*. Domain* C_2 *is* $\{a, b\}$ *and we set MBR* a *to variable* v_2*. Now the partial combination is* $\{aav_3\}$*. Next, at the* 3^{rd} *level, we instantiate* v_3 *and objects belongs to* C_3 *should have lower bounds within* $(3, 4) = (8, 10) - (5, 6)$*. Domain* C_3 *is* $\{b\}$ *and we set MBR* b *to variable* v_3*. Finally, we obtain a combination* $\{aab\}$*. In the same way, we can obtain other combinations.*

Notice 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 (e.g. $\{aab\}$ *and* $\{baa\}$*). It is easy to remove such duplicates and we will not talk it too much for the space limitation.* ■

Algorithm 1 shows the process of MOC queries using the lower bound reduction method. We start a query process by calling a function MOC_query($p, \boldsymbol{b}, h, S$) where $p = \{root,\ root,\ root\}$ and $S = \emptyset$. We use d_{ji} to denote the domain C_i for variable v_i at the j^{th} instantiation level. We first initialize the threshold T as $\boldsymbol{b}$, 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). At the same time, 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).

Algorithm 1 MOC Query Using Lower Bound Reduction

```
1:  procedure MOC_query($p, \boldsymbol{b}, h, S$)    {$p = e_1 e_2 \cdots e_h$ 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:  $T := \boldsymbol{b}$;    {Initialize threshold $T$ as $\boldsymbol{b}$.}
4:  for $i := 1$ to $h$ do
5:      $d_{1i}$ := get_children($e_i$);    {Initialize domains $d_{1i}$.}
6:  $l := 1$;    {Start from the 1st 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:         $T := T - v_l^{\perp}$;    {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_level($p$) then
27:             update_optimal_set($p', S$);    {Update $S$ considering $p'$.}
28:         else
29:             MOC_query($p', \boldsymbol{b}, h, S$);    {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($c_k^{\perp}, T$) then    {$c_k^{\perp}$ is beyond $T$.}
37:             $d_{l+1,j} := d_{l+1,j} - \{c_k\}$;    {Eliminate $c_k$ from $d_{l+1,j}$.}
38:     end
```

Let us go back to the function MOC_query(). If we are not expanding a combination at the leaf level, 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. The combinations in S, which is dominated by p', are removed.

3.2 Upper Bound Reduction

We obtain optimal object combinations and inserting them into the set S while retrieving the R-tree in a depth first way as Algorithm 1 shows. An MBR combination is promising if it has an upper bound which cannot be dominated by any combinations in S. This *upper bound reduction* method avoids expanding MBR combinations which will generate combinations dominated by others.

Definition 4 (Upper Bound for MBR Combination) *An MBR combination $p = \{e_1, e_2, \cdots, e_h\}$ has an* upper bound *$p^\top$ which is an aggregation on the upper bounds of its elements e_1 to e_h, namely, $p^\top = \Sigma_{i=1}^{h} e_i^\top$ where $e_i^\top$ is the upper bound of e_i.* □

Definition 5 *Given an objective vector $\boldsymbol{b}$, an MBR combination p is* dominated *by an object combination s if its upper bound $p^\top$ is dominated by s, namely, $d^k_{p^\top} < d^k_s$ $(k \in 1, 2, \cdots, m)$ and $d^j_{p^\top} \leq d^j_s$ $(j \in 1, 2, \cdots, m$ and $j \neq k)$.* □

Theorem 2 *An MBR combination p cannot be expanded to optimal object combinations if it is dominated by a combination s.*

Proof 2 *An MBR combination p with an upper bound $p^\top$ can be expanded to an object combination p' which have upper bounds within $p^\top$, namely, $p'^i \leq p^{i\top}$ $(i \in 1, 2, \cdots, m)$. If p is dominated by an object combination s, p' is also dominated by s because $d^k_{p'} < d^k_s$ $(k \in 1, 2, \cdots, m)$ and $d^j_{p'} \leq d^j_s$ $(j \in 1, 2, \cdots, m$ and $j \neq k)$.* ■

Example 5 *Let us consider the upper bound reduction process in Fig. 4 (a). The upper bounds of $\{abb\}$, $\{aab\}$, $\{aaa\}$ and $\{bbb\}$ are shown the figure. At the leaf level, we have obtained object combinations (i.e., $\{BAC\}, \ldots, \{FCE\}$) by expanding the MBR combination $\{abb\}$. Their attributes are shown in the figure. Considering the Theorem 2, the MBR combination $\{bbb\}$ is non-promising to generate optimal combinations because its upper bound $(12, 15)$ is dominated by $\{DCE\} = (12, 16)$ already found.* ■

We use a min-heap to organize the MBR combinations which are waiting to be expanded like the well-known BBS algorithm in [8]. Each time we pop and expand the top one and then push its expansions into the min-heap. The top one should have a minimum *Manhattan distance* to an objective vector $\boldsymbol{b}$.

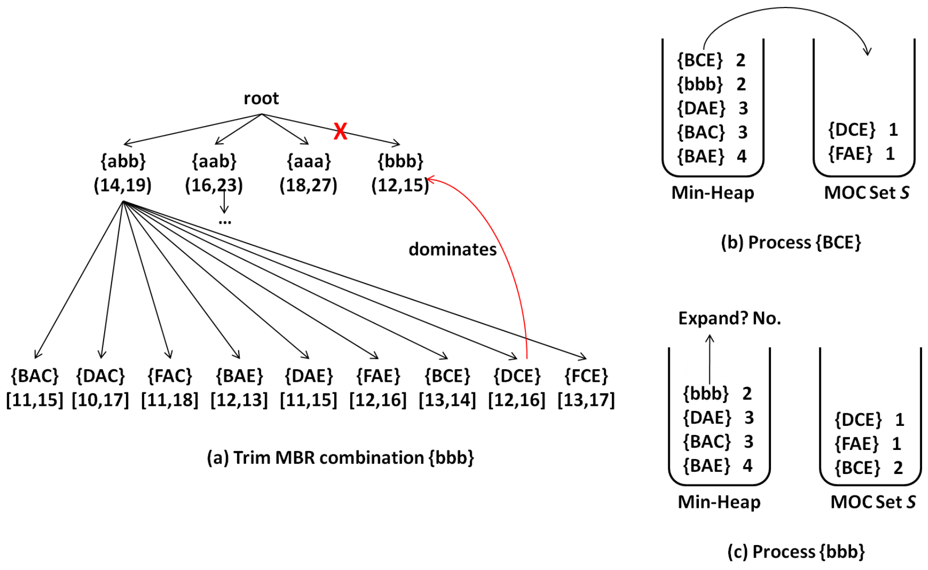


Fig. 4. Upper Bound Reduction

Definition 6 (Manhattan Distance of An MBR Combination) *Given an MBR combination p with lower bounds $p^{i\perp} \leq b^i$ $(i \in 1, 2, \cdots, m)$, the* Manhattan distance *of the combination p is $md(p) = \Sigma_{j=1}^{r} d^{j\top}$ where $d^{j\top} = b^j - p^{j\top}$ $(p^{j\top} \leq b^j$ and $r \leq m)$.* □

For example, an MBR combination $\{bbb\}$ has a lower bound $(6, 9)$ and an upper bound $(12, 15)$. Its lower bound and upper bound are within the objective vector $\boldsymbol{b} = (13, 16)$. Its Manhattan distance is $md(\{bbb\}) = (13 - 12) + (16 - 15) = 2$. Notice that MBR combinations like $\{aab\}$ are special. The MBR combination $\{aab\}$ has a lower bound $(12, 15)$ within $(13, 16)$ but an upper bound $(16, 23)$ beyond of $(13, 16)$. We set its Manhattan distance to $md(\{aab\}) = 0$.

Theorem 3 *An object combination s, which is extended from an MBR combination p, cannot dominate another MBR combination p', if p' has a smaller or equal Manhattan distance with p, namely, $md(p') \leq md(p)$.*

Proof 3 *The object combination s has a Manhattan distance $md(s) = \Sigma_{i=1}^{m} d_s^i$ where d_s^i is its distance to* **b** *at the i^{th} attribute. The MBR combination p, where s comes from, has a Manhattan distance $md(p) \leq md(s)$. Assume that the object combination s can dominate another MBR combination p', say, $d_s^k < d_{p'\top}^k$ $(k \in 1, 2, \cdots, m)$ and $d_s^j \leq d_{p'\top}^j$ $(j \in 1, 2, \cdots, m$ and $j \neq k)$. Then p' has a Manhattan distance $md(p') > md(p)$ because $md(p') > md(s)$ and $md(p) \leq md(s)$. It contradicts with the condition $md(p') \leq md(p)$ in Theorem 3.* ■

According to Theorem 3, we maintain a min-heap with respect to Manhattan distances of combinations. The top one has a minimum Manhattan distance. Other combinations in the heap cannot generate a combination which will dominate the top one. Each time we pop and expend the top one to new combinations. If a new combination is an object combination, we update the current optimal combination set S. If a new combination is still an MBR combination, we decide whether it is dominated by current optimal combinations in S. We only push the non-dominated ones into the min-heap and throw away the dominated ones. After rebuilding the min-heap, we pop a new top one and expand it by repeating the process stated above until the min-heap is empty. Notice that if a top one is dominated by any current optimal combination in S, we throw it away directly.

Example 6 *Fig. 4 (b) shows the scene when an combination $\{BCE\}$ is on the top of the min-heap. The number shown behind of each combination is its Manhattan distance. The combination $\{BCE\}$ is an object combination and we compare it with two optimal combinations $\{DCE\}$ and $\{FAE\}$ which have been found already. It cannot be dominated neither by $\{DCE\}$ nor $\{FAE\}$. We pop out combination $\{BCE\}$ and inserted it into the MOC set S.*

After popping out combination $\{BCE\}$, an MBR combination $\{bbb\}$ is on the top of the min-heap as Fig. 4 (c) shows. We pop out $\{bbb\}$ but do not expand $\{bbb\}$ because it is dominated by $\{DCE\} \in S$. ■

Algorithm 2 shows an MOC query using the lower bound reduction as well as the upper bound reduction. Algorithm 2 is similar to Algorithm 1 except for several differences annotated by comments. In the beginning, we decide whether a combination p is dominated by combinations in S using a function is_domed(p, S) (line 2). If it cannot be dominated, the process continues. We update S if p is an object combination (line 4 to 5). We expand p if it is an MBR combination (line 7 to line 38). If a complete instantiated combination p' cannot be dominated by combinations in S, we calculate its Manhattan distance $md(p')$ using a function calculate_md(p′) and push it into the min-heap Q (line 32 to 36). Note that the min-heap rebuild itself after push or pop operations. When the min-heap Q is not empty, we pop and use a new top one to execute the function MOC_query().

4 Experiments

We implemented Algorithm 2 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 the R-tree provided by a spatial index library SaIL ([9, 10]). The R-tree has a block size 512 bytes and a fill factor 70%.

We evaluated performances of Algorithm 2 with four 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 different sizes of data sets. The third set evaluated the algorithm with respect to different m's where m is the number of attributes. The fourth set evaluated the algorithm with respect to

Algorithm 2 MOC Query Using Lower Bound Reduction and Upper Bound Reduction

```
1: procedure MOC_query($p, \boldsymbol{b}, h, S, Q$)                {$Q$ is the min-heap}
2: if is_domed($p, S$) then             {$p$ is dominated by combinations in $S$.}
3:     return;
4: if is_leaf_combination($p$) then                {$p$ is an object combination.}
5:     update_optimal_set($p, S$);
6: else                                         {Expand an MBR combination $p$.}
7:     begin
8:     $p' := v_1 v_2 \cdots v_h$;
9:     $T := \boldsymbol{b}$;
10:    for $i := 1$ to $h$ do
11:        $d_{1i} :=$ get_children($e_i$);
12:    $l := 1$;
13:    while $true$ do
14:        begin
15:        if $d_{ll} \neq \emptyset$ then
16:            $v_l :=$ get_MBR(d_ll);
17:        else
18:            if $l = 1$ then
19:                break;
20:            else
21:                begin
22:                $l := l - 1$;
23:                continue;
24:                end
25:        if $l < h$ then
26:            begin
27:            $T := T - v_l^{\perp}$;
28:            forward_check($T, l, i$);
29:            $l := l + 1$;
30:            end
31:        else
32:            if $\neg$is_domed($p', S$) then
33:                begin
34:                calculate_md(p');         {Calculate Manhattan distance of $p'$.}
35:                push($Q, p'$);         {Push the new combination $p'$ into $Q$.}
36:                end
37:        end
38:    end
39: if $Q \neq \emptyset$ then
40:    begin
41:    $p =$ pop($Q$);                               {Pop the top combination.}
42:    MOC($p, \boldsymbol{b}, h, S, Q$);                {Use $p$ to do a new MOC query.}
43:    end
```

different cardinalities h's where h is the number of objects in a combination. We will show the experimental results of the three sets in Section 4.1, Section 4.2, Section 4.3, and Section 4.4 respectively.

4.1 Performances on Different Data Distributions

When we evaluate algorithm performances with different data distributions, we use five synthetic data sets $D_{-0.6}$, $D_{-0.4}$, D_0, $D_{0.4}$ and $D_{0.6}$ with different correlation coefficients -0.6, -0.4, 0.0, 0.4 and 0.6. We generated these data sets using the method in [2]. Objects in data sets $D_{0.4}$ and $D_{0.6}$ follow the correlated distribution while object in data sets $D_{-0.4}$ and $D_{-0.6}$ follow the anti-correlated distribution. Objects in the data set D_0 follows the uniform distribution. Each data set has 10K objects with two attributes ranging from 0 to 10000. We randomly select 50 different objective vectors ranging in $[1000, 9000] \times [1000, 9000]$ to evaluate the algorithm. After executing queries to find out 3-MOCs on these five data sets, we summarized average results of the random 50 different queries as OC which is the number of optimal combinations; CMC which is the number of checked MBR combinations; CAD which is the number of candidate object combinations for optimal ones; CPU which is the cost of running time with one second as a unit.

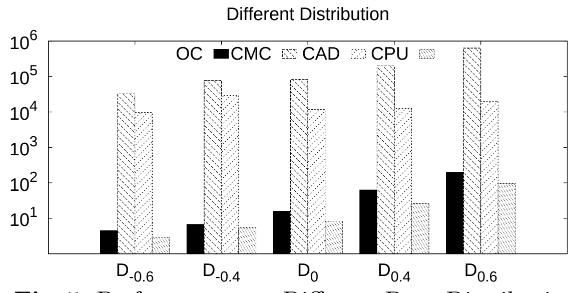


Fig. 5. Performances on Different Data Distribution

Fig. 5 shows the experimental results of data sets with different object distributions. Note that OC, CMC, CAD and CPU are all in their log scales. The correlated data sets (i.e. $D_{0.6}$ and $D_{0.4}$) have more OCs than the anti-correlated data sets (i.e. $D_{-0.6}$ and $D_{-0.4}$). The uniform distribution data set (i.e. D_0) has a middle size OCs. The number of candidate object combinations CAN is not influenced by the distributions of data sets. The CPU cost depends on how many MBR combinations (CMC) we have checked during the MOC queries. We have to check more CMCs for the correlated data sets while check fewer CMCs for the anti-correlated data sets.

4.2 Performances on Different Data Sizes

When we evaluate algorithm performances with different data sizes, we use five synthetic data sets D_{1K}, D_{2K}, D_{5K}, D_{10K} and D_{15K} containing 1K objects, 2K objects, 5K objects, 10K objects and 15K objects respectively. Objects in each data set have two attributes and follow a uniform distribution. We also randomly select 50 different objective vectors to evaluate the algorithm. After executing queries to find out 3-MOCs on these five different data sets, we summarized average results of the random 50 different queries as OC, CMC, CAD and CPU in Fig. 6. Note that OC, CMC, CAD and CPU are all in their log scales.

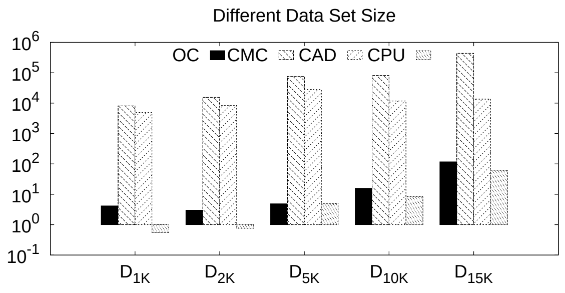


Fig. 6. Performances on Different Data Sizes

The data set (e.g. D_{15K}) containing more objects has more OCs than the data set (e.g. D_{1K}) containing fewer objects. The number of candidates (CAN) of optimal combinations is not influenced by the sizes of data sets. We have to check more MBR combinations (CMC) for a large data set (e.g. D_{15K}) than for a small data set (e.g. D_{1K}). The CPU cost depends on the CMC and it increases with the growth of the data set size.

4.3 Performances on Different Numbers of Attributes

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 $\boldsymbol{b_i}$ ($i \in 1, 2, \cdots, 15$) where $b_i^1 = b_i^2 = \cdots = b_i^m = 400 + 200 \times i$ ($m = 2, 3, 4$). Given the objective vector $\boldsymbol{b_i}$, we execute MOC queries on D_2, D_3 and D_4 in order to find out optimal combinations consisting of 3 objects.

Fig. 7 (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 $\boldsymbol{b_1}$ to $\boldsymbol{b_{15}}$. 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. 7 (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

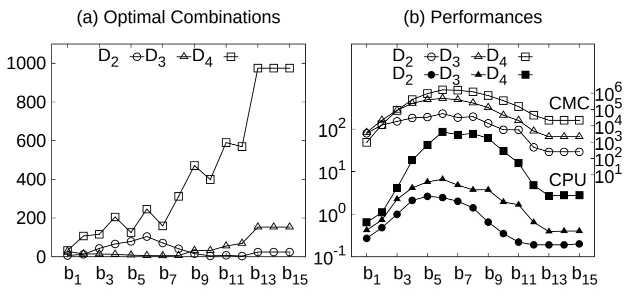


Fig. 7. Performances on Data Sets D_2, D_3 and D_4

(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.

4.4 Performances on Different Cardinality

When we evaluate algorithm performances on different cardinalities of a combination, say, different h's, we use the uniform distribution data set D_{1K}. Given the objective vector $\boldsymbol{b} = (500, 500)$, we execute MOC queries to find out optimal combinations with cardinalities $h = 1, 2, \cdots, 9$.

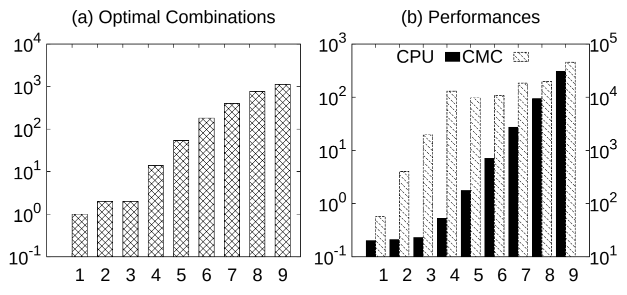


Fig. 8. Performances on Data Set D_{1k} with $h = 1, 2, \cdots, 9$

Fig. 8 (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. 8 (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 level 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.

5 Conclusions

In this paper, we propose a new multi-objective optimization problem called MOC problem. The MOC problem is to find out optimal combinations consisting of h objects with respect to a given objective vector $\boldsymbol{b}$. The optimal combinations cannot be dominated by any possible combinations. We organize objects using the R-tree index and do MOC queries efficiently with two reduction methods, say, the lower bound reduction and the upper bound reduction. We evaluated the proposed MOC query algorithm on different data sets with different objective vectors and parameter settings.

Acknowledgments. This research was partly supported by Grant-in-Aids for Scientific Research (#22300034) from JSPS and the Funding Program for World-Leading Innovative R&D on Science and Technology (First Program).

References

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`.