Applied Mathematics, **2012, 3, 78-85**
http://dx.doi.org/10.4236/am.2012.31013 Published Online January 2012 (http://www.SciRP.org/journal/am)

Combined Algorithms of Optimal Resource Allocation

Valery I. Struchenkov
Moscow State Technical University of Radio Engineering, Electronics and Automation, Moscow, Russia
Email: str1942@mail.ru

Received October 17, 2011; revised November 19, 2011; accepted November 28, 2011

ABSTRACT

Under study is the problem of optimum allocation of a resource. The following is proposed: the algorithm of dynamic programming in which on each step we only use the set of Pareto-optimal points, from which unpromising points are in addition excluded. For this purpose, initial approximations and bilateral prognostic evaluations of optimum are used. These evaluations are obtained by the method of branch and bound. A new algorithm "descent-ascent" is proposed to find upper and lower limits of the optimum. It repeatedly allows to increase the efficiency of the algorithm in the comparison with the well known methods. The results of calculations are included.

Keywords: Dynamic Programming; The Pareto Set; Branch and Bound Method; The Curse of Dimensionality; Algorithm "Descent-Ascent"

1. Introduction

The problem of optimum distribution of limited resource *R* between *n* consumers was solved by R. Bellman more than 50 years ago [1]. His method of dynamic programming allows to find the optimal path (trajectory) and lead for *n* steps of the system from the given initial state to the final one.

While using R. Bellman's algorithm, which became classical, the problem comes to finding the optimal trajectory, connecting nodes of a regular grid, which actually define the set of states on each step prior to the beginning of the account.

Later on other realizations of dynamic programming have been offered, where the task of regular grid of states was not necessary. At the same time as we move from the initial point to the end we consider only achievable states (points). And from all paths that lead to each state remain only the best [2,3].

The characteristic feature of traditional algorithms of dynamic programming is intensive growth of volume of calculations with growth *n* and *R* which has the name "the curse of dimensionality".

So, if possible resource values for each consumer are not integer and on a numerical axis are irregularly located then application of traditional Bellman's algorithm is conjugated with essential computing difficulties because of necessity of introduction small discrete and accordingly a great number of states. If number of consumers and number of possible values of resource for each of them reaches some hundreds then time needed for the decision of real problems can be unacceptable. That is why the search of more effective algorithms especially for the big dimension problem which built in multiply repeated cycle of calculation is actual. This is the purpose of this work.

2. Problem Statement

Let's consider the following problem: to find the maximum of the sum

$$\sum_{i=1}^{n} g_i(x_i) \tag{1}$$

subject to

$$\sum_{i=1}^{n} f_i(x_i) \le R,\ x_i \in X_i,\ i = 1,2,\cdots,n \tag{2}$$

where X_i—finite sets, $f_i(x_i) \ge 0$, $g_i(x_i) \ge 0$ and $R > 0$. Functions $f_i(x_i)$ and $g_i(x_i)$ may not be integer. It is supposed that the set of feasible solutions (2) is nonempty.

Variety of problems which can be written in this form, comes to allocation given resource *R* among *n* consumers. Functions $f_i(x_i)$ characterize a resource, and $g_i(x_i)$—is efficiency of it's use. If $f_i(x_i) = p_i x_i$ and $g_i(x_i) = c_i x_i$ we have the problem of optimum loading of vehicle objects, which weights p_i, costs c_i, and quantities $x_i\ (x_i = 0,1,2,\cdots)$ [1]. If additionally $x_i = \{0,1\}$, we obtain the well-known problem of "knapsack" [2].

It might be needed to look for minimum in (1). In this case functions $f_i(x_i)$—a resource, and $g_i(x_i)$—expenses. For example, $f_i(x_i)$—resources of time, and $g_i(x_i)$—expenses for realization of some project.

Non-integer values of the resource arise in a problem surface protection from aggressive influence of environment [4,5]. The surface consists of n elements, for protection of each of them m various ways can be used. Their efficiency and corresponding expenses are various. In this problem, $f_i(x_i)$—permissible damage from incomplete protection of i-th element of surface, and $g_i(x_i)$—expenses.

The problem on maximum is reduced to an equivalent problem on minimum by replacement in (1) $g_i(x_i)$ on $M - g_i(x_i)$ where $M = \max\limits_i g_i(x_i)$.

Further the problem (1,2) is being considered on minimum and the following designations are used:

$x_i^j\ (i = 1,\cdots,n;\ j = 1,\cdots,\ k_i)$—values $x_i \in X_i, f_i^j = f_i(x_i^j)$ and $g_i^j = g_i(x_i^j)$ —the corresponding values of the functions $f_i(x_i)$ and $g_i(x_i)$.

3. Algorithms with Elimination of States

Efficiency of dynamic programming can be essentially increased by elimination not only the paths leading to some state, but also actually unpromising states which obviously can't belong to an optimum trajectory, that is optimum sequence of states.

For the simple knapsack problem, this idea of elimination of the states is realized [2].

The algorithms realizing this idea for the decision of a more general problem (1,2), we'll consider a problem on minimum.

$x_i^j\ (i = 1,\cdots,\ n)$ we'll number so that $f_i^1 < f_i^2 < \cdots < f_i^{ki}$. If $f_i^j = f_i^{j+1}$, then we eliminate (f_i^{j+1}, g_i^{j+1}) if $g_i^j < g_i^{j+1}$, and we eliminate (f_i^j, g_i^j) if $g_i^j > g_i^{j+1}$.

However the pair (f_i^{j+1}, g_i^{j+1}) can be eliminated if $f_i^j < f_i^{j+1}$ and $g_i^j \le g_i^{j+1}$ as bigger resource should correspond smaller expense in the problem on minimum. In this case f_i^j will be always used instead of f_i^{j+1}. After elimination we get $g_i^1 > g_i^2 > \cdots > g_i^{ki}\ (i = 1,\cdots,\ n)$. Hence for any i the remained points with coordinates (f_i^j, g_i^j) form Pareto set on a plane (f, g), all other variants are unpromising.

At each step of dynamic programming by the state we mean total resource, which already was used. Accordingly after the first step the set of states f_1^j is such that pairs $(f_i^j,\ g_i^j)$ form Pareto set.

At each following step $m = 2,\cdots,\ n$ we'll consider set of points (F_m, G_m) of the following

$$F_m = \sum_{i=1}^m f_i(x_i),\ G_m = \sum_{i=1}^m g_i(x_i)$$

where vector $(x_1,\cdots,x_m)$ runs all values, satisfying the conditions

$$\sum_{i=1}^m f_i(x_i) \le R,\ x_i \in X_i,\ i = 1,\cdots,\ m..$$

Let for two points $(F'_m,\ G'_m)$ and $(F''_m,\ G''_m)$ where the bottom index is number of a step, and top index is point number , we have : $F''_m \ge F'_m$ and $G''_m > G'_m$. A path $(x'_1, x'_2,\cdots,x'_m)$ corresponds to point $(F'_m,\ G'_m)$ and to point $(F''_m,\ G''_m)$ corresponds $(x''_1, x''_2,\cdots,x''_m)$.

Point $(F''_m,\ G''_m)$ can be excluded from consideration because path $(x''_1, x''_2,\cdots,x''_m)$ can not be a part of optimal trajectory. In fact, let $(x''_{m+1}, x''_{m+2},\cdots,x''_n)$ be optimum continuation from point $(F''_m,\ G''_m)$, which expenses G^* correspond. Then for trajectory $(x'_1,\cdots,x'_m,\ x''_{m+1}, x''_{m+2},\cdots,x''_n)\ G'_m + G^* < G''_m + G^*$. And point $(F''_m,\ G''_m)$ is eliminated.

At $F''_m \ge F_m$ and $G'' = G'_m$ the point $(F''_m,\ G''_m)$ can also be excluded. It means that from the set of points $(F_m,\ G_m)$ on each step m it is possible to leave only Pareto subset, and from two congruent points remains only one. This rule also includes elimination of inefficient paths which lead to the same state, and elimination of actually unpromising states to which non Pareto points correspond.

Pareto subset of $(F_m,\ G_m)$ will be designated by $S_m = \{(F_m{}^k, G_m{}^k) | k = 1,\cdots,K_m\}$, numbering them according to increasing resource, *i.e.* $F_m^1 < F_m^2 < \cdots < F_m^{Km}$, $G_m^1 > G_m^2 > \cdots > G_m^{Km}$. Formation of step-by-step ordered Pareto sets S_m can be achieved in different ways. A variant where at first we form the whole set of admissible points and then leave only Pareto points turned to be ineffective. The algorithm has been realized:

The first step.

$$S_0 = \{(0,0)\};\ S_1 = \{(F_1^j = f_1^j,\ G_1^j = g_1^j) | j = 1,2,\cdots,k_1\}.$$

No elimination.

The general step. Assume that the set

$$S_{m-1} = \{(F_{m-1}^k,\ G_{m-1}^k) | k = 1,\cdots,K_{m-1}\}$$

is already constructed. At first stage (k = 1) we calculate

$$\{(F_m^j = F_{m-1}^1 + f_m^j,\ G_m^j = G_{m-1}^1 + g_m^j) | j = 1,\cdots,k_m\}$$

without elimination. At second stage for $k = 2,\cdots,\ K_{m-1}$ (the external cycle) we consistently calculate $P(F_{m-1}^k + f_m^j,\ G_{m-1}^k + g_m^j)\ j = 1,\cdots,k_m$ (an internal cycle). May be only three results of comparison each calculated point P with already available (the nearest on value of a resource):

1) P is not included in formed set as there is a dominating point in it;

2) P is included (with reserving the order) as there is no dominating point in relation to it, and it isn't domi-

nating;

3) New point P is included in the formed Pareto set, and one or probably more points in relation to them P is dominating, are eliminated from it.

Owing to orderliness Pareto set and $f_m^j \ge 0$ no necessity to analyse all points for search a dominating point. At $k = 2$ and $j = 1$ search begins with 1, *i.e.* points with F_m^1. At $k = 3, \cdots, K_{m-1}$ and $j = 1$ search begins with number received by a point with $F_{m-1}^{k-1} + f_m^1$ or dominating over it. At $k = 3, \cdots, K_{m-1}$ and $j = 2, \cdots, k_m$ search in an internal cycle begins with number, received by point with $F_{m-1}^k + f_m^{j-1}$ or dominating over it. Search goes always with increase of resource and comes to an end at achievement $F_{m-1}^k + f_m^j$, *i.e.* the resource of the new point—"candidate". In item 3 search of dominated points begins with $N + 1$ where N—number of the new point which was included and stops at the first unsuccessful attempt.

As a result we receive Pareto set of points, which are ordered on increase of a resource.

For problems of big dimension the number of Pareto points can be great, especially at non integer values $f_i^j \left(i = 1, \cdots, n; j = 1, \cdots, k_i\right)$ that has demanded working out of the new algorithms, allowing to eliminate some unpromising Pareto points. It is possible on each step of dynamic programming if to use the approximate decision (initial approach) as the top estimation of optimum with possibility of its clarification in the process of the account or to build bilateral prognostic estimations of optimum. In particular on each step Pareto set include points corresponding minimum resources and accordingly maximum expenses which through some steps can already exceed total expenses for all steps, corresponding to initial approach. Such situation comes for smaller number of steps in the presence of good initial approach.

Let's assume that some admissible vector $x^*\left(x_1^*, x_2^*, \cdots, x_n^*\right)$ is calculated and the corresponding value of the target function is

$$G_n^* = \sum_{i=1}^{n} g_i\left(x_1^*\right)$$

At the decision of a problem (1,2) on minimum on a step m Pareto point $\left(F_m^j, G_m^j\right) \in S_m$ is eliminated, if $G_m^j > G_n^*$, as it can't belong to optimum trajectory because on subsequent steps target function can increase only as $g_i\left(x_i\right) \ge 0$.

If for $j_k = \max j : F_m^j \le F_m^*$ and a value $\delta = G_m^* - G_m^{jk} > 0$, then $\left(F_m^*, G_m^*\right)$ is non Pareto point. It is replaced on $\left(F_m^{jk}, G_m^{jk}\right)$; for $i > m$ we remain x_i^* (accordingly f_i^* and g_i^*), therefore all $G_i^*\left(m < i \le n\right)$ we decrease on δ, and F_i^* we decrease on $F_m^* - F_m^{jk}$.

On subsequent steps $\left(m < i \le n\right)$ corrected values F_i^* and G_i^* are used for elimination Pareto points. Expenses for the rest part of a trajectory and accordingly the top border of total expenses can be calculated approximately for everyone Pareto point. We decrease G_n^*, using minimum of received values of total expenses, and receive an opportunity of additional elimination Pareto points. Correspondent algorithms use obtained values G_n^* as the top estimations of optimum, and the bottom estimation is equal to zero and, as a rule, it is far from optimum.

We use a combination of dynamic programming and a method of branches and borders for construction the improved bilateral estimations of optimum. Thus everyone Pareto point on each step is considered as a point of branching with construction of bilateral estimations of optimum (the bottom and top border). Efficiency of such method depends on quantity of states, computing expenses for borders calculation and their nearness to optimum.

Let's designate expenses for all trajectory, corresponding to initial approach, through E (in a method of branches and borders they are called as a record), and their bottom border through H. Expenses for the rest part of trajectory from a point $\left(F_m^j, G_m^j\right)$, corresponding to some admissible approach, we will designate through E_m^j, their bottom border through H_m^j. Then condition of elimination Pareto points $\left(F_m^j, G_m^j\right)$ will become $G_m^j + H_m^j > E$. If $G_m^j + E_m^j < E$, new value of a record is $G_m^j + E_m^j$. We modify initial approach accordingly.

If for some m and j $E_m^j = H_m^j$ there is no sense to consider $\left(F_m^j, G_m^j\right)$, as a branching point. If additionally $G_m^j + E_m^j < E$ this point and corresponding trajectory are remembered and stored until then yet won't be obtain value of a record smaller, than $G_m^j + E_m^j$. If "the record will stand" then corresponding decision is optimum. If on some step there will be no Pareto points the record is the required decision. On each step it is possible to correct the bottom border, replacing H on

$$h = \min\left(G_m^j + H_m^j\right),\ \left(j = 1, 2, \cdots, K_m\right) \text{ if } H < h,$$

and to stop calculation at $E - H < \varepsilon H$,
where ε is defined by demanded accuracy of the decision.

4. Construction of Initial Approach (The Top Border)

The simple algorithm of construction of initial approach consists in the following steps:

1) For $i = 1, 2, \cdots, n$ consecutive for $j = 1, \cdots, k_i$ it is postponed f_i^j on an axis of abscisses, and g_i^j on an axis of ordinates and we receive sequence of points, which defines strictly monotonously decreasing piecelinear function, because $\left(f_i^j, g_i^j\right)$ are Pareto points. We will consider that such functions are constructed for all i.

2) We calculate $R_{\min} = \sum_{i=1}^{n} f_i^j$ and $R_{\max} = \sum_{i=1}^{n} f_i^{ki}$.

If $R_{\min} > R$, the problem has no decision. If $R_{\max} \le R$, we have the trivial decision, choosing $j = k_i$ for everyone *i*. Let's designate $I = \{1, \cdots, n\}$. We make $k = n$.

3) We calculate $R_{av} = (R - R_{\min})/k$. If $\exists\, i \in I : f_i^1 + R_{av} > f_i^{ki}$,then we make $f_i^* = f_i^{ki}$, further we eliminate *i* from *I,* replace *k* on *k* − 1, *R* on $R - f_i^{ki}$ and repeat item 3, differently $f_i^* = f_i^1 + R_{av} \big| i \in I$ and we go to item 4.

4) For $i = 1, \cdots, n$ we consistently define $\max j : f_i^j \le f_i^*$ and replace f_i^* on f_i^j. Accordingly $g_i^* = g_i^j$.

5) Initial approach (trajectory)

$$F_m = \sum_{i=1}^{m} f_i^*,\ G_m = \sum_{i=1}^{m} g_i^* \quad (m = 1, \cdots, n)$$

Similarly, it is possible to build initial approach in the process of dynamic programming for the rest part of the trajectory, starting with any Pareto point.

5. Calculation of Bilateral Estimations of an Optimum

For construction of bilateral estimations of an optimum we use the piece-linear functions received in item 1. Those of them which aren't convex, we will replace on their convex shells $w_i(z_i)$ $(i = 1, 2, \cdots, n)$. As a result we receive a continuous estimated problem: to find a minimum

$$W(z) = \sum_{i=1}^{n} w_i(z_i), \tag{3}$$

where $z(z_1, \cdots, z_n)$ is an unknown vector

and

$$\sum_{i=1}^{n} z_i \le R,\ f_i^1 \le z_i \le f_i^{ki},\ i = 1, \cdots, n \tag{4}$$

Optimum of a continuous problem (3,4) which it is obvious no more an optimum of a problem (1,2), we will accept as required bottom border.

In this problem of nonlinear programming the target function and the system of restrictions have essential features which will be used for its decision by simple algorithm.

The ends of links of broken lines $w_i(z_i)$ we will designate through z_j^j $(j = 1, \cdots, p_i)$.

We have $z_i^1 = f_i^1,\ z_i^{pi} = f_i^{ki},\ p_i = k_i$, if $g_i(f_i)$ — convex function, differently $p_i < k_i$. Values

$$u_i^j = \left| \left(w_i^{j+1} - w_i^j \right) \middle/ \left(z_i^{j+1} - z_i^j \right) \right|$$
$$\left(i = 1, \cdots, n;\ j = 1, \cdots, p_i - 1 \right)$$

we will name biases. Owing to monotony and convexity functions $w_i(z_i)$ sequence of biases u_i^j $(i = 1, \cdots, n)$ is strictly monotonously decreasing.

The problem (3,4) has simple sense: it is how much necessary to go down on each broken line $w_i(z_i)$ from the initial point that without breaking restriction on the sum of abscisses, to receive the minimum sum of ordinates? The simple and obvious enough answer:- on each step it is necessary to go down along a link with the maximum bias. We will result a formal substantiation of this statement.

Let $z^*(z_1^*, \cdots, z_n^*)$—the decision of a problem (3,4). We will designate through
$M = \left\{ u_i^j \middle| i = 1, \cdots, n;\ j = 1, \cdots, p_i - 1 \right\}$—a set of all biases,

$$M_1(z^*) = \left\{ u_i^j : z_i^{j+1} \le z_i^* \mid i = 1, \cdots, n;\ j = 1, \cdots, p_i - 1 \right\},$$

$$M_2(z^*) = \left\{ u_i^j : z_i^j < z_i^* < z_i^{j+1} \middle| i = 1, \cdots, n;\ j = 1, \cdots, p_i - 1 \right\}$$

and

$$M_3(z^*) = \left\{ u_i^j : z_i^* \le z_i^j \middle| i = 1, \cdots, n;\ j = 1, \cdots, p_i - 1 \right\}$$

—accordingly sets of the biases completely used for descent, used partially and not used at descent.
$M = M_1 \cup M_2 \cup M_3$. $M_i \cap M_j = \varnothing$ for $i \ne j$ $(i, j = 1, 2, 3)$

$$W(z^*) = \sum_{i=1}^{n} \left(g_i^1 - \sum_{j=1}^{pi} u_i^j \delta_i^j \right) = \sum_{i=1}^{n} g_i^1 - \sum_{i=1}^{n} \sum_{j=1}^{pi} u_i^j \delta_i^j \tag{5}$$

where $\delta_i^j = z_i^{j+1} - z_i^j$, if $u_i^j \in M_1$; $\delta_i^j = z_i^* - z_i^j$, if $u_i^j \in M_2$ and $\delta_i^j = 0$, if $u_i^j \in M_3$.

Let $u_m^1 = \max\ u_i^1$ $(i = 1, \cdots, n)$. We will prove that $u_m^1 \in (M_1 \cup M_2)$, respectively $z_m^* > z_m^1$.

Let's assume that this statement isn't true. We take the minimum bias $u_k^r \in (M_1 \cup M_2)$, to it corresponds $\delta_k^r > 0$. Having reduced δ_k^r by some Δ, we receive target function increase on $u_k^r \Delta$ and possibility of its reduction by the big value $u_m^1 \Delta$ without violating restriction on the resource, as $u_m^1 > u_k^r$. Hence z^* not an optimum. The received contradiction proves necessity of use of the maximum bias. Necessity of whole using of the maximum from unused biases is similarly proved, the resource won't be settled yet.

As a result we receive the following algorithm of descent:

1) We take initial point $z\left(z_1^1, z_2^1, \cdots, z_n\right)$, which corresponds the maximum value of target function

$$\sum_{i=1}^{n} w_i\left(f_i^1\right) = \sum_{i=1}^{n} g_i^1$$

We fix the rest of resource $T = R - \sum_{i=1}^{n} f_i^1$.

2) From all links of all broken lines we take a link with

the maximum bias. Let it will be a link with number j of a broken line with number m. On the first step $j = 1$ owing to convex and monotony of broken lines. We will change only a variable z_m. Its increase gives reduction of target function with the greatest speed. We calculate a movement step $c = \min\left(z_m^{j+1} - z_m^j,\ T\right)$ and replace z_m^j on $z_m^j + c$. Target function will decrease on $u_m^j c$, and the remained resource will decrease on c.

3) From all remained links of all broken lines we choose again a link with the maximum bias and repeat item 2 while remained resource T won't be settled.

If there are several links with the maximum bias the priority is given to a link with the maximum length which the remained resource allows to use completely $\left(z_s^{j+1} - z_s^j \le T\right)$. If a resource is not enough for full use of any of such links, any of them is used.

Let's notice that in a minimum point only bias u_s^j, which used by last, can be used partially. If it also is used completely, *i.e.* on last step $z_s^{j+1} - z_s^j = T$, the received decision of continuous problem coincides with the decision of initial discrete problem and it is definitive. Otherwise it is initial approach, and value of target function in a minimum point is required bottom border H_0. If last considered a broken line initially was convex $\left(k_s = p_s\right)$ we receive the approached decision of initial discrete problem and accordingly record E by canceling last step (it explains the aspiration to make it less). Otherwise we define the absciss of the node nearest at the left on broken line $g_s\left(f_s\right)$ using optimum value z_s^*. Ordinate of this node, we will designate through g_s^k. The bottom border H_0 doesn't change, and record $E = H_0 + g_s^k - w_s\left(z_s^*\right)$.

Note that convex shells are used only to calculate the borders, but at formation of a set of states on each step, we consider all admissible states, *i.e.* initial broken lines.

For effective realization of the algorithm essential value has a way of search of the greatest biases. The simple way consists in sorting of all biases of all links of all broken lines. But expenses of operative memory for sorting can be excessive as the essential part of biases in general can not be demanded at calculation of the bottom border and a record. Instead of complete sorting only biases of the first elements of broken lines are ordered as it should be nonincreasing. With each of them number of a broken line and number of its link communicates. Initial number of its link is equal 1. In the presence of equal biases the priority is given to a link with the greatest length. The received array of biases we will designate through $U\left(u_1, u_2, \cdots, u_n\right)$. According to the stated algorithm descent process begins with use u_1 and continues with maximum biases. Further, if the current bias is used completely it is replaced in the array U on u_1', *i.e.* bias of the next link of the same broken line maintaining order in the array U. If we can not completely use the next bias because of the restriction on resource, it is used partially and the descent is finished.

On the first step $u_1 = u_m^1$ and $u_1' = u_m^2$, where $u_m^1 = \max u_i^1$; $\left(i = 1, \cdots, n\right)$. The bias u_1' is located in the array U so that the condition not increasing of its elements was satisfied. If $u_1' < u_2$, the elements of U more than u_1', move to the left, so on the first place always stands the greatest bias. If all links of some broken line s were used, the fictitious link with number k_s and a zero bias is entered, and process proceeds before resource exhaustion. The total array of biases, and also $\Delta b = z_s^* - z_s^j$, that is a part of a link of a broken line which was used by the last at descent is remembered.

Alternative to algorithm of descent is ascent from a point $\left(f_1^{k1}, f_2^{k2}, \cdots, f_n^{kn}\right)$ with the minimum value of target function $\sum_{i=1}^{n} w_i\left(f_i^{ki}\right)$, which increases on each step with a minimum speed because we use the next minimum bias. The used resource decreases from $R_{\max}$ to R. In this algorithm in case of exhaustion of all biases (links) of some broken line the fictitious link with number a zero is entered. Its bias is equal to a great value.

As a result we receive and remember the new array of biases $V\left(v_1, \cdots, v_n\right)$, and also $\Delta b_1 = z_s^{j+1} - z_s^*$—a part of a link of a broken line which was used by the last at ascent. Both algorithms give the same decision $\left(z_1^*, z_2^*, \cdots, z_n^*\right)$ and value of target function (the bottom border H_0), but arrays of biases U and V are different. In the array U—biases of links—applicants for the further descent, and in the array V—on ascent. Everyone i-th broken line is presented by a bias of one link, but number of this link in array U one more, as z_i^* is equal to an absciss of the left end of the link presented in the array U or the right end in the array V. The exception is made by the broken lines presented by biases, appeared as a result on the first places u_i^* and v_i^*. $u_i^* = v_i^*$, with them the same j-th link of a broken line is connected. This line was used by the last. Its number is s, thus $b_s^{j+1} - b_s^j = \Delta b + \Delta b^1$.

The algorithm of ascent is proved similarly as the algorithm of descent. It begins to work with ranking in order of decreasing biases of last links of all broken lines.

The algorithm of descent can be also used for calculation of borders of expenses on the rest part of a trajectory for everyone Pareto point.

Let's note two features of this algorithm:

1) An array of n biases received as a result and the data connected with it about numbers of broken lines and their links contains the necessary information for decision restoration z^* and corresponding optimum trajectory for a continuous problem (sequence of states $z_1^*, z_1^* + z_2^*, \cdots, z_1^* + z_2^* + \cdots + z_n^*$).

2) If instead of R we have $R' < R$ there is no sense to

begin descent anew, it can be continued from received for R an optimum point, using a total array and the remained biases. However at $R' > R$ for updating of the received decision we are forced to descend anew.

3) If from a total array exclude all biases, which concern the first broken line, then the resulting array will correspond the decision $\left(z_2^*,\cdots,z_n^*\right)$ of a problem:

$\min W(z) = \sum_{i=2}^{n} w_i(z_i)$, where $z(z_2,\cdots,z_n)$—an unknown vector, at restrictions

$$\sum_{i=2}^{n} z_i \le R - w_1\left(z_1^*\right),\ f_i^1 \le z_i \le f_i^{ki},\ i = 2,\cdots,n,$$

because for decision of this problem it should to use the maximum biases from all broken lines, except the first. In other words, the resulting array of biases corresponds to an optimum trajectory from a state which was received after the first step of algorithm $\left(z_1^*, w_1\left(z_1^*\right)\right)$ up to the end.

Similarly, passing to the next r-th step of algorithm of dynamic programming and excluding from array U biases $u_r^j\ (j = 1,\cdots,p_r - 1)$, we receive the array of biases, which corresponds z_r^* and therefore to a state $\left(F_r^*, G_r^*\right)$ on an optimum trajectory.

It is similarly possible to use the algorithm of ascent.

Essential lacks of the stated algorithms of descent and ascent are:

1) Initial points are far from an optimum.

2) Information about the optimal trajectory, which was found on the first step in solving the continuous problem, is not used.

3) As a rule, new values of a record turn out for the points located near to a point on an optimum trajectory. But these algorithms find a new value of a record already after points which could be eliminated at movement from a point on an optimum trajectory were passed.

New algorithm of calculation of borders which we name "descent-ascent" is free from these lacks. Its basic points:

1) The initial estimated problem (3,4) is solved both a method of descent and a ascent method. Arrays U and V, as base, and also Δb and Δb_1 are remembered. Each element of each array connected with number of a broken line and number of a link to which it corresponds. That gives the chance to restore the optimum point z^* and the optimum trajectory.

2) The optimum trajectory $\left(F_i^*, G_i^*\right)\ (i = 1,\cdots,n)$ and corresponding borders E_0 and H_0 are remembered. This is decision of an estimated problem, and a deviation from an optimum of an initial problem doesn't exceed $E_0 - H_0$.

3) We construct Pareto set after the first step of dynamic programming.

4) The biases corresponding to the first broken line are excluded from arrays U and V.

5) Starting with state $\left(F_1^*, G_1^*\right)$ descent for states with $F_1^k < F_1^*$, and ascent for states with $F_1^k > F_1^*$ are carried out consistently for definition of bottom and top border costs for the remainder of the trajectory and possible elimination of a state or record updating.

6) On the subsequent steps of dynamic programming both borders are defined similarly.

Special is s-th step of algorithm of dynamic programming, where s is a number of a broken line to which posesses a link used by last at construction of an optimum trajectory of a problem (3,4). Its bias was used partially, therefore at removing of this bias from base arrays of biases, Δb and Δb_1 are nulled.

In aforementioned special cases of a considered problem all calculations become simpler, as at $f_i(x_i) = p_i x_i$ and $g_i(x_i) = c_i x_i$ for everyone a broken line all links have one bias, and at $x_i = \{0,1\}$ each broken line consists of one link.

As the algorithm of descent-ascent demands considerable volume of calculations, for revealing of its efficiency in comparison with more simple algorithms experimental calculations have been executed.

6. Experimental Calculations

To compare the different algorithms they were implemented in the next computer programs:

P1—Dynamic Programming with elimination of the path which lead to the same state;

P2—Elimination only nonPareto states;

P3—Additional elimination of a part unpromising Pareto states with use initial approach;

P4—Additional elimination unpromising Pareto states with calculation of the bottom and top borders of an optimum on algorithm of descent-ascent.

Calculations were carried out on personal computer Intel Pentium 4, CPU 3.0 GHz, 512 MB the RAM.

In calculations abscisses and ordinates of broken lines are pseudo-casual real numbers from [1,100], but the number of tops of broken lines $K_i = K$ didn't depend from i.

Account time depends not only on number of steps n, value of K and from a preset value of resource R, but also from concrete values f_i^j and g_i^j.

In the first calculation small values $n = 50$ and $K = 10$ were set. Results are presented in **Table 1**. Designations: **sum**—total number of remembered states on all steps, **max**—maximum number of states on a separate step, ***T***—time of the account in seconds.

Calculations on P1 were carried out under an additional condition: states on each step are considered coinciding if they differ (on a resource) less, than on the set

Table 1. Calculations with n = 50 and K = 10.

	R = 1000			R = 2000			R = 3000			R = 4000		
N	sum	max	T	sum	max	T	sum	max	T	sum	max	T
P1	4,913,186	125,963	1485	8,573,228	285,081	2499	9,774,235	325,130	2767			>3600
P2	749,454	36,890	49	1,205,318	63,635	63	1,426,221	89,532	67	1,452,752	93,875	67
P3	639,420	31,212	36	752,111	23,399	23	690,340	29,526	14	528,481	19,492	8
P4	9786	436	0.1	1705	150	0.2	1748	117	0.3	78	8	0.4

value d.

Attempt to receive result with d = 0.001 was unsuccessful because of excessive for-expenditure machine time. In the table the result received at d = 0.005 is presented. At d = 0.01 account time was essentially less, but accuracy of calculation can appear insufficient as on deviation target function exceeded 0.1.

Calculations under other programs were satisfied without this additional condition, but at comparison of real values the constant 10^{-9} was used.

Account time on P4 was less than 0.1 sec, and at R = 4000 the account has come to the end on 30th step. It is interesting a sudden reduction of account time on P3 with increasing R. In this calculation on P3 uniform distribution of a resource between consumers appears more close to an optimum with increasing resource R as the maximum requirements for a resource at consumers differ slightly.

In further calculations program P1 wasn't used because of hopelessness of algorithm for a considered class of problems. The classical algorithm of dynamic programming (a method of a regular grid) is especially unpromising at a grid step d, equal to the value used at work with P1.

Essential influence on growth of time of the account renders growth K. So at n = 40 and K = 20 already at R = 2500 and use P2 **sum** = 2,748,038, **max** = 200,040, T = 809 sec, and at use P3 **sum** = 1,433,853, **max** = 81,889, T = 241 sec. At increase R the number remaining Pareto points becomes unacceptably big. There is a same situation, as with algorithm P1: operative memory is exhausted, and exchange with connected external memory is slow. It is characteristic that in the same calculation on P4 at R = 2500 **sum** = 2886, **max** = 152, T < 0.5 seconds, but at R = 1000 P4 gives **sum** = 6855, **max** = 500 and T = 1.2 second.

Similar results have been received under the same conditions, but with n= 100 and K = 40. The decision on P2 and P3 during comprehensible time managed to be received only at d = 0.002 and more. So at R = 2000 and d = 0.005 P3 gives **sum** = 7,367,886, **max** = 98,087 and T = 3208 sec. And at use in this calculation P4 increase R gave both increase, and reduction of number of the points which have remained after elimination. Accordingly account time both increase and reduce. Results are presented in **Table 2**.

It is characteristic that in this calculation small change of R has essentially affected on account time. It is visible from **Table 3**.

We explain the received results that at R = 2077 initial value of a record differs from an optimum on 0.29, and at R = 2078 on 0.34. As appears from resulted above algorithm, with increase R value of a bias u_s^j which at calculation of initial values of bottom border H_0 and record E_0 is used by the last, can only decrease. It gives smaller value $E_0 - H_0$, if $z_s^* - z_s^j$ is constant because $E_0 - H_0 = u_s^j \left(z_s^* - z_s^j \right)$. Here z_s^*—an absciss of the point received at descent on a link of a broken line with number s at calculation H_0, and z_s^j—an absciss of the top of this broken line nearest at the left, used at calculation E_0, that is at "rounding off" to the decision of a discrete problem. But reduction u_s^j can be compensated by difference increase $z_s^* - z_s^j$, which can both to increase, and to decrease with growth R that depends on the initial data $\left(f_i^j,\ g_i^j \right)$.

For revealing of possibilities of use P4 for the decision of problems of the big dimension calculations have been executed at K = 20 and n = 100, 200, 300, 400, 500. It is established that the number of remembered states and accordingly account time essentially depends from R. So at n = 400 and R = 28,000 **sum** = 108,893, **max** = 1099 and **T** = 2 sec. And at R = 2000 only in the same conditions **sum** = 544,554, **max** = 3335 and **T** = 1189 second. At n = 500 and R = 35,000 **sum** = 222475, **max** = 1740 and T = 12 second.

In general the increase n and K doesn't mean obligatory growth of time of the account as at "successful" R that is at small $z_s^* - z_s^j$ this time can be a little and in problems of the big dimension. At any initial data exists R at which initial approach appears the final decision $\left(z_s^* - z_s^j = 0 \right)$. For this purpose it is enough to repeat calculation, having reduced R on $z_s^* - z_s^j$ or having increased R on $z_s^{j+1} - z_s^*$.

If at use P4 to be limited to approximate solution, for example instead of d = 10^{-9} use d = 10^{-5}, then in the same conditions the number of states and account time

Table 2. The decision on P4 with various *R*.

	$R = 2000$			$R = 2100$			$R = 2500$			$R = 4000$		
N	sum	max	T	sum	max	T	sum	max	T	sum	max	T
P4	75,663	2192	91	98,270	3856	153	17,191	420	11	16,479	736	6

Table 3. The impact of small change of *R* on account time.

	$R = 2077$			$R = 2078$			$R = 2079$			$R = 2080$		
N	sum	max	T	sum	max	T	sum	max	T	sum	max	T
P4	20,177	475	12	99,514	2274	179	44,056	920	49	54,973	1399	74

essentially decreases. However more perspective appears the termination of account when $(E-H)/H<\varepsilon$ instead of increasing d.

If $\varepsilon = 10^{-5}$ in any of calculations, which were discussed above, the account time on P4 did not exceed 0.5 sec.

Thereupon problems with $n = 4000$ and $K = 40$, and then with $n = 5000$ and $K = 50$ in the same conditions of a choice f_i^j, g_i^j have been solved. At $R = 10^5$, $2*10^5$ and $3*10^5$ and $\varepsilon = 10^{-5}$ account stopped after the third step, and time of the account didn't exceed 8 seconds. Similar results have been received and at a casual choice f_i^j from [1,1000], preservation of all other parameters of calculation and at various values R.

In addition as an option in the program P4 descent algorithm was used in place of the descent-ascent algorithm. It is established that the descent algorithm in all the calculations required more time than the algorithm of descent-ascent. Moreover, the computing time on a P4 with descent algorithm in some calculations was greater than P3, because of the limitations descent algorithm noted above.

7. Conclusions

The results of comparison between different algorithms (P1, P2, P3, P4) leads to the following conclusions:

1) For the decision of problems (1,2) classical algorithms of dynamic programming can be considered as become outdated.

2) The most perspective is the combined algorithm (P4).

3) At use of the combined algorithm it is expedient to search for the approximate solutions, breaking the account at small a relative error of search of minimum ε. For this purpose it is possible to set acceptable value ε, but instead for the decision of problems of the big dimension it is possible to set obviously small ε, to display step-by-step values E, H and $(E-H)/H$ and finish process taking into account current results and elapsed time.

4) At growth R and n it is possible both increasing, and decreasing of account time. Actually algorithm P4 if doesn't overcome completely "a dimension damnation" does its action selective.

The problem (1,2) is considered as an example, but the algorithms using Pareto sets, although not universal, as well as dynamic programming at whole, are applicable and for the decision of other problems: a various kind two-parametrical problems of distribution of resources, storekeeping, calculation of plans of replacement of the equipment, a choice of suppliers and so on.

High-speed algorithm descent-ascent can be used to solve the problem of the form (1.2), in which there are several restricts (2). This more complicated problem is reduced to a multiple solution of (1,2) with one restriction.

Consideration of these problems is beyond of the present article.

REFERENCES

[1] R. Bellman and S. Drejfus, "Applied Dynamic Programming," Princeton University Press, Princeton, 1962.

[2] S. Martello and P. Toth, "Knapsack Problems: Algorithms and Computer Implementation," John Willey & Sons Ltd., Chichester, 1990.

[3] S. Dasgupta, C. H. Papadimitriou and U. V. Vazirani, "Algorithms," McGraw-Hill, Boston, 2006.

[4] V. I. Struchenkov " Dynamic Programming with Pareto Sets," *Journal of Applied and Industrial Mathematics*, Vol. 4, No. 3, 2010, pp. 428-430, doi:10.1134/S1990478910030154

[5] V. I. Struchenkov. "Optimization Methods in Applied Problems," Solon-Press, Moscow, 2009.