Mathematics and Statistics 8(3): 339-346, 2020
DOI: 10.13189/ms.2020.080313 http://www.hrpub.org

High-speed Dynamic Programming Algorithms in Applied Problems of a Special Kind

V. I. Struchenkov*, D. A. Karpov

Department of General Informatics, Institute of Cybernetics of the Russian Technological University (MIREA), Russia

Received March 24, 2020; Revised April 28, 2020; Accepted May 20, 2020

Abstract The article discusses the solution of applied problems, for which the dynamic programming method developed by R. Bellman in the middle of the last century was previously proposed. Currently, dynamic programming algorithms are successfully used to solve applied problems, but with an increase in the dimension of the task, the reduction of the counting time remains relevant. This is especially important when designing systems in which dynamic programming is embedded in a computational cycle that is repeated many times. Therefore, the article analyzes various possibilities of increasing the speed of the dynamic programming algorithm. For some problems, using the Bellman optimality principle, recurrence formulas were obtained for calculating the optimal trajectory without any analysis of the set of options for its construction step by step. It is shown that many applied problems when using dynamic programming, in addition to rejecting unpromising paths lead to a specific state, also allow rejecting hopeless states. The article proposes a new algorithm for implementing the R. Bellman principle for solving such problems and establishes the conditions for its applicability. The results of solving two-parameter problems of various dimensions presented in the article showed that the exclusion of hopeless states can reduce the counting time by 10 or more times.

Keywords Dynamic Programming, System State, Optimal Trajectory (path), Optimality Principle, Pareto Sets

1. Introduction

The main provisions of dynamic programming were formulated when considering a dynamic system, whose state is determined by one or more parameters. At fixed times the system is affected by external impacts (controls), which convert it from one state to another. It is required to find a sequence of impacts that transfers the system from a given initial state to a final state with minimal total cost. The sequence of actions determines the sequence of states, i.e. the trajectory (path) of the system.

R. Bellman [1,2,3,4] formulated the principle of optimality, the meaning of which is to go along the optimal trajectory from any state to the final state if the "past history" does not matter. The applicability conditions of dynamic programming are as follows:

1. The objective function (costs) can be calculated in stages, which are determined by the given moments of the impact.
2. The set of possible paths from each state does not depend on how the system got into this state.
3. The result of the impact on the system (step transition) in each of the states does not depend on how the system got into this state.

Back in the early 60s of the last century, R. Bellman and his colleagues realized their method in solving a number of practically important problems [1,2].

The method can be applied to the development of optimal solutions in multi-stage processes, including those cases when the division into stages is done artificially [2,3,4,5]. Currently, dynamic programming algorithms are successfully used to solve applied problems from various fields of practice [6].

One of the latest proposals is the approximation of plane curves by splines of complex structure [7].

In the original algorithm, it was proposed to carry out calculations from the final state to the initial one (backward-sweep method [1,2,4]). For this, all possible states of the system must be known. For this reason, it was recommended [1,3,4,8] to solve many applied problems using a regular grid of states.

In accordance with the principle of optimality, from each state to the final state it is necessary to go along the optimal path. But under the same conditions, each state

from the initial state should be reached along the optimal path. In other words, of all the paths leading to a certain state, we need to keep only the best one.

It corresponds to a direct - sweep method [3,4]. In this case, the set of possible states of the system can be constructed in stages and using of a regular grid of states is not necessary.

It should be noted that with an increase in the number of states at each stage and the number of stages, the number of analyzed (and memorized) transitions increases sharply, which can lead to unacceptable costs of computer time, even when using modern public computers. Therefore, the development of high-speed dynamic programming algorithms continues to be relevant, especially for the tasks in which the state is described by more than one parameter. This is especially important when developing systems in which dynamic programming is embedded in a repeatedly repeated calculation cycle.

Improving the performance of dynamic programming algorithms is achieved when rejecting not only hopeless paths leading to a specific state, but when rejecting hopeless states and all their incoming and outgoing paths.

2. Objectives

The main goals of this article are to show how the idea of rejecting hopeless states is realized in solving many applied problems for which traditional dynamic programming algorithms were previously proposed, formulate additional conditions for the applicability of the algorithm with rejecting states, and present the results of comparative calculations which confirm the effectiveness of the new algorithm. In addition, the article aims at showing how, in some applied problems, dynamic programming allows one to obtain recurrence formulas for calculating the optimal trajectory without enumeration of options at all.

3. Some Tasks that Allow the Calculation of the Optimal Trajectory without Enumerating Options

Let us consider the task of planning supplies (purchases) of equipment for the development of production (trade, etc.) in several stages (for example, months or quarters) for a given time. Let T be the number of stages, c_i - the cost of supplying a unit of equipment (price) at the i-th stage, x_i- the number of units of equipment delivered (i = 1,2, ..., T-1), r_i and s_i, the number of units used and equipment already supplied at the i-th stage respectively, ($s_i \geqslant r_i$ and $r_{i+1} \geqslant r_i$ for all i). h_i- the cost of storing a unit of unused equipment. It is required to find the sequence $x_i \geqslant 0$(i = 1,2, ..., T-1) for which the total costs for all stages are minimal. It should be noted that the operating costs of the equipment are not taken into account, since the number of used units of equipment r_i is given and does not depend on x_i (i = 1,2, ..., T-1). At the last stage r_T units of equipment are operated and there are no supplies.

Deliveries and commissioning refer to the end of the stage, therefore $s_{i+1} = s_i + x_i$ (i = 1,2, ..., T-1). We assume that $s_1 = r_1$.

The simplest solution is to supply the minimum amount of equipment for each stage, that is, take $x_{t+1} = r_{t+1}-r_t$, but this maybe a bad solution if equipment is more expensive at last stage. Let z_i denotes the costs at the i-th stage and Z_i the total costs for the i stages.

$$Z_1=c_1x_1.$$

$$Z_2=z_1+z_2= c_1x_1+ c_2x_2+ h_2(s_2-r_2)= \\ = s_2(c_1 -c_2+h_2)- c_1r_1- h_2r_2+c_2s_3. \quad (1)$$

Here we use $x_1= s_2 -s_1$; $x_2= s_3 -s_2$ and $s_1 =r_1$.

Of all the paths leading to the given state s_3, we must leave the one for which Z_2 is maximum. This means that from all the possible states of the second stage s_2, we need to take the one that, for a given s_3, gives a maximum of $Z_2 = Z_2$*. But it can be seen from formula (1) that Z_2 is a linear function of s_2 for a fixed s_3. Therefore, for $d_2 = c_1-c_2 + h_2 > 0$, we take the minimum $s_2 = r_2$, and for $d_2 <0$, we take the maximum $s_2 = s_3$, since $x_2 \geqslant 0$

In the first case, from (1) we obtain

$Z_2^* = c_1(r_2 - r_1)- c_2r_2+ c_2s_3=p_2+q_2s_3$. Here $q_2= c_2$ and $p_2= c_1(r_2 - r_1)- c_2r_2$. In the second case, $Z_2^* =- c_1r_1-h_2r_2 +(c_1+ h_2) s_3= p_2+q_2s_3$, but now $P_2=- c_1r_1-h_2r_2$ and $q_2= c_1+ h_2$. If $c_1 -c_2+h_2=0$ the obtained pairs of p_2 and q_2 coincide. In this case, we can take any value of s_2, satisfying the condition $r_2 \leqslant s_2 \leqslant s_3$.

If we now take $Z_i=p_i +q_is_{i+1}$, then for Z_{i+1} we get $Z_{i+1}= p_i +q_is_{i+1} +c_{i+1}(s_{i+2}- s_{i+1})+h_{i+1}(s_{i+1}- r_{i+1}) = p_i - h_{i+1} r_{i+1}+ s_{i+1}(q_i- c_{i+1}+ h_{i+1})+ c_{i+1}s_{i+2}$

For a fixed s_{i+2} it is necessary to find s_{i+1} at which the costs for Z_{i+1} stages are minimal. If $d_{i+1}= q_i- c_{i+1}+ h_{i+1}>0$ we take $s_{i+1}= r_{i+1}$, otherwise $s_{i+1}=s_{i+2}$. In any case, $Z_{i+1}^* =p_{i+1} + q_{i+1}s_{i+2}$, but in the first case ($d_{i+1}= q_i- c_{i+1}+ h_{i+1}>0$) we get $p_{i+1}= p_i+ r_{i+1}(q_i- c_{i+1})$ and $q_{i+1}= c_{i+1}$, and in the second case $p_{i+1}= p_i- h_{i+1}r_{i+1}$ and $q_{i+1}= q_i+ h_{i+1}$.

It turns out that in any state of s_k at any stage we need to go from the state of the previous stage with the lowest value (r_{k-1}) or with a value equal s_k.

Denoting $q_1=c_1$ and $p_1= -c_1r_1$, for i=1,2,…,T-2 we obtain the following algorithm:

1. i=1;
2. Calculate $d_{i+1}= q_i- c_{i+1}+ h_{i+1}$ and remember it.
3. If $d_{i+1} \geqslant 0$, calculate $p_{i+1}= p_i+ r_{i+1}(q_i- c_{i+1})$ and $q_{i+1}= c_{i+1}$; otherwise $p_{i+1}= p_i- h_{i+1}r_{i+1}$ and $q_{i+1}= q_i+ h_{i+1}$. Remember p_{i+1} and q_{i+1}.
4. If i<T-2 then increase i by 1 and go to step 2, otherwise calculate $Z_{T-1}= p_{T-1}+ q_{T-1} r_T$.

This is the minimum value of the objective function,

since at the stage with number T, the costs are equal to zero: there are no supplies and $s_T=r_T$, since $s_T<r_T$ is unacceptable, and $s_T>r_T$ means unnecessary costs for equipment which will not be used.

We will restore the optimal trajectory, passing sequentially from the last stage to the second (Fig. 1). If $d_{T-1}>0$, then go from r_T to r_{T-1}, otherwise save r_T, etc.

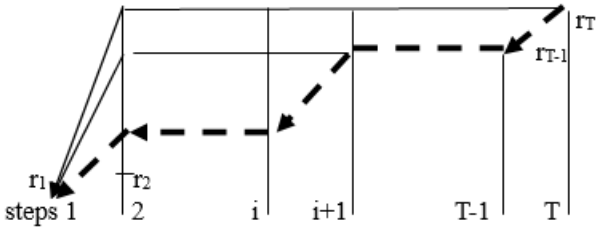

Figure 1. Recovery of the optimal trajectory

The problem under consideration can be solved using the traditional dynamic programming algorithm with a partition of a regular state grid, but the above algorithm is preferable. It allows, in addition to the initial data, to remember only the signs of all the calculated values of d_i and only the last calculated pair p_i,q_i. The amount of computation is also insignificant.

If raw materials or, for example, building materials and not equipment that is not consumed are to be supplied, the algorithm is almost the same, but the calculation formulas are different.

The needs for raw materials for work at each stage are denoted by r_i, and the actual deliveries at each stage through (i=1,2,…T).

We calculate the partial sums: $R_1=r_1$ and then $R_{i+1}= R_i + r_{i+1}$ for all i=1,2,…,T-1. The state of the system at the i-th stage will be S_i i.e. the total amount of raw materials delivered for all previous stages, including the current one. $S_1=s_1$;

$S_{i+1}= S_i + s_{i+1}$; i=1,2,…,T-1. The conditions $s_i \geq r_i$ and

$r_{i+1}\geq r_i$ are optional (except for $s_1 \geqslant r_1$). Instead, we have

$S_i \geqslant R_i$ (i=1,2,…,T-1) and $S_T = R_T$.

The costs of delivery (c_i) and storage (h_i) of a unit of raw materials at the i-th stage are known.

Costs for the first two stages $Z_2= c_1S_1 + h_1(S_1-R_1) + c_2(S_2-S_1) + h_2(S_2-R_2) = (c_1-c_2+h_1)S_1 + (c_2+h_2)S_2 - h_1R_1 - h_2R_2$.

It is required to find S_1, which for a fixed S_2 gives the minimum value of Z_2^*. If $d_2= c_1-c_2+h_1\geqslant 0$ we take $S_1= R_1$, otherwise $S_1= S_2$. In any case, Z_2^* is written in the form $Z_2^*=p_2 +q_2S_2$, but in the first case $p_2= R_1(c_1-c_2)- h_2R_2$ and $q_2= c_2+h_2$, and in the second case $p_2=-h_1R_1-h_2R_2$; $q_2= c_1+h_1+ h_2$.

Further, if $Z_i= p_i +q_iS_i$, then

$Z_{i+1}= p_i- h_{i+1} R_{i+1}+ (q_i- c_{i+1}) S_i+(c_{i+1}+h_{i+1}) S_{i+1}$.

The choice of S_i for fixed S_{i+1} is determined by the sign $d_{i+1}= q_i- c_{i+1}$. If $d_{i+1}\geqslant 0$ then $S_i=R_i$, otherwise $S_i=S_{i+1}$. In any case, $Z_{i+1}^*= p_{i+1}+ q_{i+1}S_{i+1}$. But if $d_{i+1}\geqslant 0$ we get $p_{i+1}= p_i+R_i(q_i- c_{i+1})-h_{i+1}R_{i+1}$ and $q_{i+1}= c_{i+1}+ h_{i+1}$, otherwise $p_{i+1}= p_i -h_{i+1}R_{i+1}$; $q_{i+1}= q_i+ h_{i+1}$. Setting $p_1=-h_1R_1$ and $q_1 =c_1+ h_1$, we can apply the above algorithm by sequentially calculating d_{i+1} and pairs p_{i+1},q_{i+1} using the formulas obtained for i=1,2...,T-1. After calculating all S_i we calculate $s_i= S_{i+1}- S_i$. (i=T-1,T-2, …,1).

4. Dynamic Programming with the Rejection of Hopeless States

We consider the integer linear programming problem, known as the knapsack problem [8,9]:

Find max $z=c_1x_1+c_2x_2+\ldots+c_nx_n$ for

$a_1x_1+a_2x_2+\ldots+a_nx_n\leq b$, $x_j \in \{0,1\}$,$a_j>0$; $c_j>0$; $j=1,\ldots,n$.

We can assume that the problem is the optimal distribution of a given resource b among n consumers, the j-th consumer receives a_j resource units or nothing. But we can talk about the problem of the optimal choice of n items with given costs c_j for loading a vehicle with carrying capacity b [3]. When choosing the j-th item, a resource is spent on the amount of a_j and the effect c_j is achieved. We need to choose so as not to consume a resource greater than b and achieve the maximum total effect z. The solution to this problem is given in the literature as an example of using dynamic programming [3,8]. The problem considered in [3] is in loading a car with a carrying capacity of 35 units of weight with an optimal set of six items, the weights and costs of which are shown in table 1.

Table 1. Source data

Item	I1	I2	I3	I4	I5	I6
Weight qi	4	7	11	12	16	20
Cost ci	7	10	15	20	27	34

It is required to choose such items, the total weight of which does not exceed 35, and the total cost is maximum.

The problem is solved by the backward-sweep method using a state grid with discrete d = 1. At each of the six stages, 36 states are considered [3].

Note that if we build a trajectory from “start to end”, then after the first stage there are only 2, after the second 4, etc., but not 36 states. Moreover if the weights of objects are non-integer (for example, 7.28; 11.65, etc.), then for the exact solution of the problem we will have to use d = 0.01, and the number of states (grid nodes) at each stage increases by two orders of magnitude with the same number of items, and dynamic programming becomes less efficient than method of full enumeration.

But the state grid is not needed at all, since it is possible to consider the already used resource as the state of the system, build all realistically achievable states in stages, calculating the total cost of the items taken for each of

them, and if two (or more) paths fall into the same state leave the one to which the maximum cost corresponds. In the problem under consideration, such a state occurs after the third stage: 11 units of the resource are used. Two ways are brought into this state: (0,0,11) take only the third item and (4,7,0) take only the first two. On the first path, the total cost is 15, and on the second path 17 (see table 1 above). The first path and all its extensions are rejected.

We will consider two states after the fourth stage: the path (4,0,0,12): only the first and fourth items were taken and the path (0,7,11,0): only the second and third items were taken. For the first of them, 16 units of weight were used and the total cost of 27 (7 + 0 + 0 + 20) was obtained. And for the second, 18 units of weight were used and the cost was 25 (0 + 10 + 15 + 0). The second state 18 (25) is unpromising and can be rejected, since it is easier to place the remaining items with a reserve of the load-carrying capacity resource, and the opportunities of choice on the next stages are the same.

At the fifth stage, state 22 (32) is unpromising (since there is state 20 (34)), as well as states 27 (42), 30 (45), and 34 (52). In general, after the fifth stage, only 15 (and not 32 and not 36) states will remain. Therefore, the algorithm with rejection of states in this problem is more efficient than the traditional algorithm. The presence of non-integer item weights does not complicate the task when using it.

In a more general case, there are several instances of each item, but the appearance of hopeless states is also possible.

For example, there are many items, each having a duplicate. If the weight of the first three items is 4; 7 and 12, and the cost is 8, 15 and 16, respectively, then at the third stage a hopeless state arises. Indeed, taking only the first and two second items, we get the total weight of 4 +7 +7 = 18 and the cost of 8 + 15 + 15 = 38.

And if we take only the first two and the third item, we get the weight 4 + 4 + 12 = 20 and the cost 8 + 8 + 16 = 32, that is, the weight is more, and the cost is less. State 20 (32) is unpromising, since at the same stage there is state 18 (38), and the possibilities and consequences of making decisions about all subsequent items are the same for them. With an increase in the number of instances of items, the number of hopeless states can increase significantly.

5. The Algorithm

Now we describe the dynamic programming algorithm with rejection of states and state the conditions for its applicability.

So far we assume that the state is determined by one parameter P. We will consider a two-criterion problem: the first criterion is parameter P. The second parameter Q is the value of the objective function corresponding to the variant of the trajectory leading to the given state. In resource allocation problems P is the amount of resource used (a minimum is needed), and for Q-, a maximum is needed. In the task of loading vehicles, P is the total weight of the selected items, and Q is their total cost.

1. States after the first stage (take a different number of copies of the first item), we order by one of the parameters, for example, by P, and calculate the Q value for each of them.
2. We form successively the states of the next stage, obtained as extensions of the paths leading to the states of the last considered stage while maintaining order. Several continuations can proceed from each state. We believe that they are ordered by incrementing the parameter P. In the problem with several instances of each item, this corresponds to the choice of 1,2,3 ... etc. copies of the item that corresponds to the stage under consideration. The formation of the next stage begins with a transition from each state with a minimum increment of parameter P.

As applied to the knapsack problem, the process is shown in Fig.2.

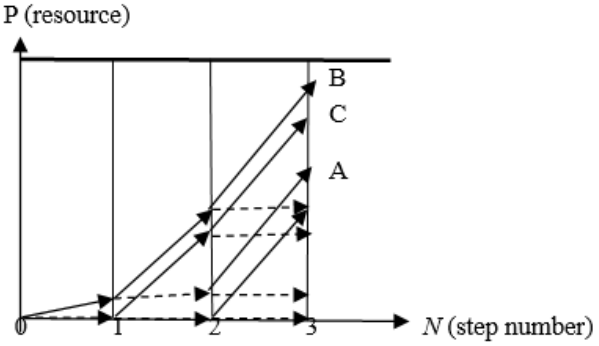

Figure 2. To the rejection of states in the knapsack problem

If q_i is the weight of the item i, then after the first step we have two states 0 and q_1, after the second step we have the same two states (do not take the second item) and two more new ones (take the second item without the first - q_2 or with the first $q_1 + q_2$). After the third step, we have the same 4 states (did not take the third item) and 4 new ones, a total of 8 states, etc.

For this task, the minimum increment P is zero (we don't take the next item), and all the states of the last formed stage are simply rewritten into the list of states at a new stage. Further, we sequentially consider transitions with the following increments параметра *P*. In the knapsack problem with each attempt to form a new state (point C in Fig. 2), the already formed states A and B (adjacent to it) were determined. They correspond to a smaller and larger, if any, value of P (Fig. 2.).

If a new state coincides with one of them by the criterion P, then the paths leading to it are compared by

the criterion Q and the best of them remains. In addition to coincidence, three more options are possible:

- for the new state, the second criterion is less than for A i.e. $Q_C \leqslant Q_A$. The new state is rejected, since it is not better than existing by any criterion.

Further comparisons are made with the closest state exceeding state C by criterion P (point B in Fig. 2). If $Q_C < Q_B$ or there is no such state B, then the new state C is included in the list of states of the stage being formed.

- if $Q_C \geqslant Q_B$, then state B is excluded from the list, state C takes its place. In this case, one or more states with higher values of P and lower values of Q compared to state C may be excluded. In other words, at each stage Pareto set of states is formed.

If not all stages are formed, then go to step 2; otherwise, from the Pareto set we take the state that is necessary for the meaning of the problem. In the problems considered, this is the state with the highest Q.

The algorithm is easily generalized if there are more state parameters and when replacing maximization with minimization. So, with the number of state parameters greater than 1, for example, when a vehicle is loaded with items of a given weight and volume and if there are restrictions on the total weight and total volume, a three-criteria problem is considered (minimum total weight and total volume and maximum cost). In any case, at each step, only the Pareto set of states should be left.

Applicability conditions for a dynamic programming algorithm with state rejection:

1. All conditions of applicability of the traditional method of dynamic programming must be satisfied;
2. In each of the states at every stage, the same actions (actions on the system) should lead to the same increments of the objective function and parameters that determine the state.
3. If there are restrictions on the state parameters, then for any admissible trajectory which starts from the dominated state, (which will be rejected) the corresponding increments of the parameters (step-by-step actions on the system) should lead to the permissible trajectory, if we apply these actions, starting from the dominant (which will be left) state.

If these conditions are met, then from the best (Pareto-dominant) state, we can always continue the best of the path leading to it, making the same decisions and, therefore, getting the same increments of the objective function as from the worst (dominated) state. This means that the advantage of the dominant state can be maintained until the end of the process. In other words, the dominated state is “forever behind” and therefore it is rejected and its continuations are not analyzed.

6. Results of Solving Two-parameter Tasks

An increase in the state parameters number sharply enlarges the amount of computation in the implementation of the traditional dynamic programming algorithm and can create computational difficulties even when using modern computers. To a large extent, these difficulties can be overcome in solving problems that allow the rejection of states. The efficiency of the state rejection algorithm was tested on two tasks. The first task is the optimal loading of the vehicle with items of various cost, weight and volume. The total volume and weight are limited, but the number of copies of each item is not limited.

The problem was solved using the outlined above algorithm with the rejection of states, that is, with the formation at each stage of the Pareto set of a three-criteria problem (weight, volume, cost). Comparative calculations were carried out using the new algorithm and the traditional R. Bellman algorithm (without splitting the regular grid of states and rejecting the hopeless paths leading to the same state). The number of objects consistently increased, and the calculation was carried out at a fixed load capacity W = 600 units of weight and capacity V = 500 units of volume. The source data was pseudo-random numbers. The calculations were carried out on a personal computer with an Intel Core 2 DUO2400 MHz processor and 2048 MB of RAM.

By modern standards, this is a low-power computer, but for comparative calculations it is enough.

A different number of items was considered. Table 2 summarizes some of the results.

Table 2. The results of the calculations

Number of items (pcs)	R. Bellman’s	algorithm	State rejection	algorithm
	Number of states	Counting time (sec)	Number of states	Counting time (sec)
10	23514	11,02	646	0,30
20	479248	121,51	17116	12,23
30	1816000	1339,43	36320	25,30
50	7409484	7314,21	145284	138,97
75	10589540	8580,46	203645	156,80
100	13871196	10773,10	256874	189,52
150	21588545	13108,23	392519	226,65

The second two-parameter problem is as follows.

An enterprise can produce N types of products using one (any) of two types of raw materials, the quantities of which are X and Y, respectively. For the production of one product of the i-th type, x_i or y_i units of raw materials of the first and second type are required, respectively. The number of products manufactured per month is limited by the production capacity a_i, regardless of the raw materials used. It is required to determine the quantity and the type of products manufactured per month, so that their total cost would be maximum. The formal statement of the problem is as follows.

$$\text{Find max} \quad \sum_{i=1}^{i=N} (k^x_i + k^y_i) c_i$$

$$\text{under restrictions} \quad \sum_{i=1}^{i=N} k^x_i x_i \le X, \quad \sum_{i=1}^{i=N} k^y_i y_i \le Y$$

$$k^x_i + k^y_i \leqslant a_i \ (1 \leqslant i \leqslant N).$$

Here

$c_i \geq 0$ - the price of one product of the i-th type,

x_i- the amount of raw materials X necessary for the manufacture of products of the i-th type,

y_i- the amount of raw materials Y necessary for the manufacture of products of the i-th type,

$a_i \geqslant 0$ -the maximum number of products of the i-th type, which can be made in a month,

$k_i^x \geqslant 0$-the number of products of the i-th type made from raw materials X,

$k_i^y \geqslant 0$- the number of products of the i-th type made from raw materials ***Y***.

The next step is to determine the number of products of the i-th type (i = 1,2, ..., N) in addition to those already manufactured, and the system states are the corresponding total quantities of raw materials of the first and second types, which were used for all already manufactured products.

In this problem, at every stage, we fix the number of products from raw materials of the second type and the corresponding amount of this raw material (starting from zero) and sequentially increase the number of products from raw materials of the first type from zero until the production capacity limit is satisfied. Next, we enlarge and fix the number of products from raw materials of the second type and sequentially increase the number of products from raw materials of the first type, etc.

This takes into account restrictions on the amount of raw materials of each type.

At every step, we obtain the Pareto set of solutions to the three-criteria problem, from which, after all steps, we select the solution with the maximum value of the objective function and, by returning, restore the trajectory and determine the unknowns k_i^x and k_i^y.

The effectiveness of the new algorithm was evaluated in comparison with the R. Bellman algorithm. In the calculations, the number of types of products consistently increased while maintaining fixed quantities of raw materials X = 261 and Y = 259. For x_i, y_i, a_i, pseudo-random numbers were used.

Some of the results are presented in Table 3, whereby it follows that the algorithm with the rejection of states turned out to be more efficient than the traditional dynamic programming algorithm. This difference increases with a large number of stages, since the costs of rejecting non-Pareto states are insignificant, and their number increases sharply with growing dimension of the problem.

With the number of types of products 100, the new algorithm worked 28 times faster than the traditional, and with 10 types of products - 14 times.

Table 3. The results of the calculations

The number of types of products (pcs)	R. Bellman's algorithm		State rejection algorithm	
	Number of states	Counting time (sec)	Number of states	Counting time (sec)
10	98451	212,33	7551	15,38
20	850710	1547,21	24306	59,14
30	2520180	2647,26	70005	108,34
50	3095198	3822,21	83654	147,63
75	3626036	4758,64	95422	186,71
100	4366722	6354,73	122519	226,55

7. Discussion

The above mentioned conditions for the applicability of the algorithm with the rejection of states are sufficient, but not all of them are necessary, since the advantage of the dominant state, identified at some step, can remain until the end of the process even if it decreases at the next steps.

The speed of the algorithm with the rejection of hopeless states depends on their number, which in turn depends not only on a specific task, but also on specific numerical values of the source data.

So in the knapsack problem, if for each item the values of weight and cost are numerically equal, then there is no rejection of states.

A large percentage of state rejection and consequently the reduction of the counting time in the above calculation results is explained by the fact that pseudo random numbers were used as the initial data. So items appeared to have less weight and volume, but a greater cost than many of the other objects.

In other words, this means that if, for example, in the task of loading vehicles, the weight, volume and cost of items are considered as criteria, then in the corresponding criteria set there are dominated points. Corresponding items cannot be selected for loading, if items that have advantages over them are in sufficient numbers. In general, if there is one point that dominates all the others, then no other items can be selected.

A feature of the algorithm with the rejection of states in the presence of dominated points in the initial data is that the counting time depends on the order in which items are examined. Starting from the dominant items, it is possible to reduce the counting time significantly due to earlier rejection of hopeless states.

It is also advisable to pre-process the source data in order to exclude dominant points.

For one-parameter problems, the time spent on searching and rejecting hopeless states when using the above algorithm is insignificant, but if such a rejection takes place, then reducing the counting time is significantly more than these costs.

For two-parameter problems, the detection and rejection of hopeless states is more complicated, but even in this case it can have a significant effect both in the presence and absence of dominant points among the initial data.

8. Conclusions

The implementation of dynamic programming in specific algorithms for solving applied tasks depends on the characteristics of the task. For example but not limited this in order to approximate plane curves, we have to use the traditional algorithm with a partition of the state grid [10,11]. But many tasks allow the use of more efficient algorithms.

Tasks, in which dynamic programming allows to obtain formulas for calculating the optimal trajectory without enumeration options, is relatively rare, in contrast to the rejection of options.

The list of tasks in which it is advisable to use dynamic programming with rejection of states is not limited to those considered ones in this article. For example the well-known problem of the optimal distribution of financial resources between competing investment projects is described by the same model as the knapsack problem and can be solved using the above algorithm. The same algorithm can be used to solve other optimization problems:

- calculation of the optimal timing of equipment replacement;
- the choice of methods (mechanisms) for the production of work;
- selection of suppliers of goods supplied in batches of various volumes and prices. This task differs from the problem of optimal loading of vehicles only in the sign of inequality –restriction on the total weight, which does not interfere with the use of the algorithm with state rejection.

As a result, it can be stated that in problems that allow the rejection of states, the new algorithm, as a rule, is more efficient than the traditional R. Bellman algorithm, both in terms of the amount of memory used and the counting time, but the class of such problems is much narrower than for the traditional algorithm.

REFERENCES

[1] R. Bellman. Dynamic Programming, Princeton University Press, Princeton, 1957.

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

[3] E.S. Wentzel. Operations Research: Challenges, principles, metodologiya. KnoRus, Moscow, 2010.

[4] Headmy A. Taha. Introduction to Operations Research, 10th edition: in Russian.- M. Williams Publishing House, 2019.

[5] V.S. Mikhalevich. Sequential optimization algorithms and their application, Cybernetics. No. 19.1965.

[6] G. Cavagnari, A. Marigonda, B. Piccoli. Generalized dynamic programming principle and sparse mean-field control problems. Journal of Mathematical Analysis and Applications, vol.481, Issue 1, 2020.

[7] D.A. Karpov, V.I. Struchenkov. Special Spline Approximation in CAD Systems of Linear Structure Routing, Mathematics and Statistics, vol.7, No 5, 2019.

[8] O.A. Kosorukov O.A., A.V. Mishchenko. Operations

research. Textbook for students universities studying the specialty "Mathematical methods in economy." M.: Exam, 2013.

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

[10] V.I. Struchenkov. Piesewise Linear Approximation of Plane Curves with Restrictions in Computer – Aided Design of Railway Routes, World Journal of Computer Application and Technology, vol. 2, № 1, 2014.

[11] V.I. Struchenkov. Computer Technologies in Linear Structures Routing, Russian Technological Journal, 2017, vol 5, No 1, 28-41 pp.