CSCanada
Canadian Social Science
Vol. 15, No. 8, 2019, pp. 88-95
DOI:10.3968/11267

ISSN 1712-8056[Print]
ISSN 1923-6697[Online]
www.cscanada.net
www.cscanada.org

Dynamic Programming Algorithm for Public Crisis Management Scheme

XIAO Hongfei[a],*

[a] Lecturer, Department of Management, Sichuan Conservatory of Music, Chengdu, China.
*Corresponding author.

Received 9 May 2019; accepted 15 August 2019
Published online 26 August 2019

Abstract
At present, social media is developing rapidly, and the frequency of public crises is increasing, which has a certain impact on social stability. This paper studies the public crisis management measures based on dynamic programming algorithm. According to the knapsack problem of public crisis management, dynamic programming algorithm was designed, and the traditional dynamic programming algorithm came up, then 5 different types of data sets used for three dynamic programming algorithms were run 10 times in the 5 different types of experimental data sets. The test results show that the efficiency of the algorithm proposed in this paper is improved.

Key words: Dynamic programming algorithm; Public crisis; Knapsack problem

Xiao, H. F. (2019). Dynamic Programming Algorithm for Public Crisis Management Scheme. *Canadian Social Science*, *15*(8), 88-95. Available from: http://www.cscanada.net/index.php/css/article/view/11267
DOI: http://dx.doi.org/10.3968/11267

INTRODUCTION
Dynamic programming is an abstract, which is difficult to grasp but a very important method in the basic algorithm design method. (Wong, et al, 2015). On the basis of optimization theory, dynamic programming can efficiently solve many problems which are difficult to solve by search or greedy algorithm (Mehdi, et al, 2015). The application field of the dynamic programming method is very wide, and many important application problems can be solved by dynamic programming method (Wu and Srikanthan, 2006). For example, dynamic programming method can be used for the shortest path, resource allocation, large-scale time delay system of production scheduling, gas production scheduling problem, the longest common subsequence problem, goods merge, in the environment and resources under the constraints of the logistics enterprise, the enterprise site equipment update, the longest increasing subsequence and inventory management, guaranteed service selection, the optimal combination of two binary search tree, knapsack problem, various practical application problems (Hua, Yu, & Lau, 2010). Therefore, it is of great significance to understand the dynamic programming design method to improve the ability and solve practical application problems.

1. STATE OF THE ART
Currently, scholars are rich in research on dynamic programming algorithm. Some scholars use dynamic programming to design a shortest line model in a special network diagram. When the stage and state variables are large, the algorithm is inefficient (Tang and Gupta, 1995). In order to improve the efficiency of dynamic programming in solving such complex multistage decision problems, we propose a way to improve the recursive way in dynamic programming, which improves the unidirectional expansion to a bidirectional expansion mode, and finally the two-way expansion will coincide at a certain stage of the problem, and the algorithm will end. The improved two way expansion dynamic programming method and direct one-way algorithm can solve the problem, but the time efficiency of two-way expansion is better than the one-way recursive method (Wu, et al, 2014). Some scholars in the scheduling problem with dynamic programming method to solve the reservoir, the decision problem is a multi - stage into the two stage

decision-making problem, diminishing marginal benefit in the premise of the reservoir, they analyzed the relationship between reservoir storage capacity optimal drainage quantity and the optimal, and proposed that reservoir operation is monotonous the nature, on the basis of the original improved dynamic programming algorithm. The experimental results showed that the improved algorithm can get the optimal solution of the problem and reduce the calculation time. (Jou Jonathan, et al, 2016). In practical applications, when there are random factors in the multistage decision process, the multistage decision process is called a random decision process (Pombeiro, Machado and Silva, 2015). The dynamic programming method can also deal with the randomness. At this moment, the corresponding dynamic programming method is called random dynamic programming (Delipetrev, Jonoski & Solomatine, 2015). In the stochastic dynamic programming, the state of the next stage is not determined by the state and the decision of the current stage. The state of the next stage will obey a probability distribution. Of course, the probability distribution is still determined by the state and decision of the current stage. In addition, the adaptive dynamic programming and multi-objective constrained dynamic programming, and continuous dynamic programming have also made great progress. (Delipetrev, Jonoski & Solomatine, 2017).

2. METHODOLOGY

2.1 Design of Dynamic Programming Algorithm for Public Crisis Processing

The knapsack problem is a classic combinatorial optimization problem, which has a wide range of applications in many fields, such as cargo loading, investment portfolio, feeding problem and so on. The knapsack problem is widely used in the field of selection based on limited resources. Knapsack problem can be described as: given the weight and value of a group of goods, what items should be selected to maximize the total value of goods within a defined total weight. Knapsack problem can be derived from a series of related problems: 0/1 knapsack problem (only one item), complete knapsack problem (items with infinite parts) and multiple knapsack problem (goods with finite parts).A dynamic programming idea is used to solve the complete knapsack problem (multiple knapsack and complete knapsack problem similar) and optimization.

First, we can describe the complete knapsack problem. For a given n event, the importance of event I is w_i, the value is v_i, each event has infinite parts, and the existing knapsack capacity is w. What events should we do to make the total value of events in the backpack maximum? Describe this problem with mathematical expressions， Finding the n vector $(x_1, x_2, ..., x_n)$ ， $\max imizep = \sum_{i=1}^{n} v_i x_i$ ， And satisfy the constraint conditions： $\sum_{i=1}^{n} v_i x_i \le w, x_i \in \{0,1,2,...w / w_i\}$. The complete knapsack problem is very similar to the 0/1 knapsack problem, but there are infinite pieces per event in the complete knapsack problem. For each event, there are only two decisions in the 0/1 knapsack problem: choose or not. But in the complete knapsack problem, the related strategy of the event is not only two kinds, but there are a variety of strategies, such as selecting 0 pieces, selecting 1 pieces, selecting 2 pieces, selecting $\lfloor w / w_i \rfloor$ parts. Borrowing the idea of solving the 0/1 backpack problem. The sub problem $m(i, j)$ is defined as the maximum value of event i based on knapsack capacity j. Although there are infinite pieces per event, for a specific event, there are still two strategies to put in a backpack or no backpack. According to this, we can write the state transfer equation:

$$m(i, j) = \max\{m(i-1, j-x, w_i) + x \cdot v_i, 0 \le x \cdot w_i \le j\}$$

（1）

The initial condition of the iteration is：

$$m(l, j) = x \cdot v_1, 0 \le x \cdot w_1 \le j \quad （2）$$

This conventional dynamic programming algorithm is called NDP(Normal Dynamic Programming)，The algorithm flow is shown as shown in the diagram. The algorithm (1) shows that：As with the 0/1 knapsack problem, the complete knapsack problem also needs to calculate the $O(nw)$ sub problems, but the time for calculating each sub problem will no longer be a constant time but a $O(w / w_i)$. Therefore, the time complexity of the NDP algorithm is $O(nw (w / w_i))$, and the space complexity is $O(nw)$.

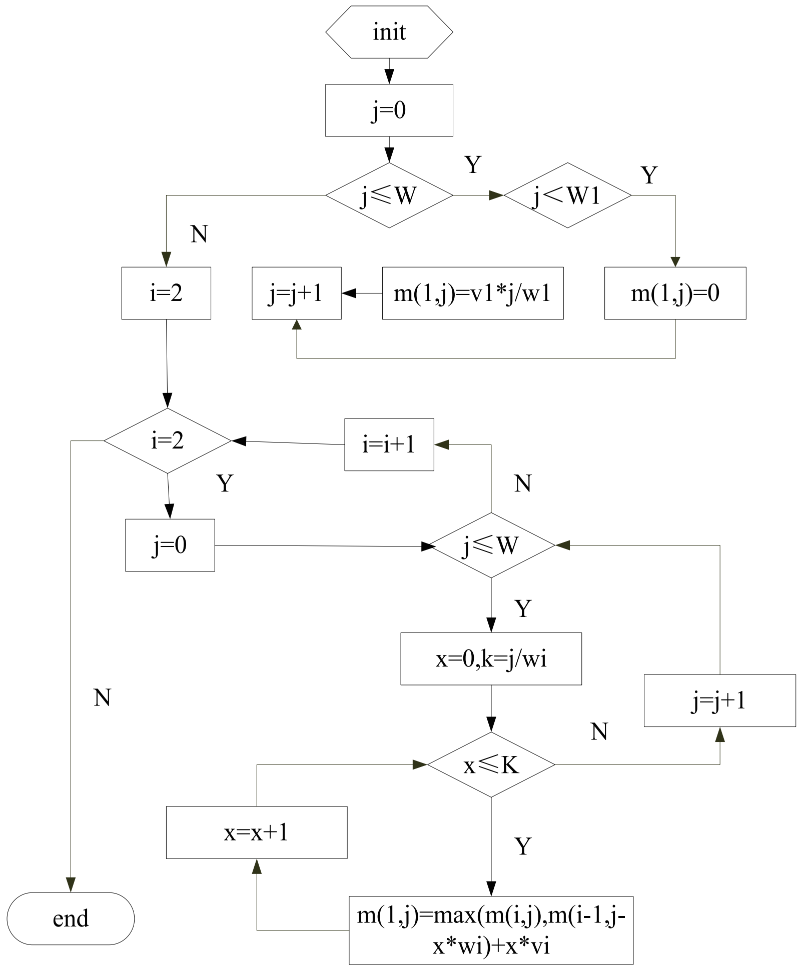

Figure 1
Classic NDP algorithm flow chart

2.2 Improvement of Dynamic Programming Algorithm

For the knapsack problem, the following optimization measures can be considered: for two events, i、j, if $w_i \geq w_j, v_i \leq v_j$ is satisfied, $w_i \geq w_j, v_i \leq v_j$ cannot be considered when putting events into knapsack. This optimization measure is obviously correct: because in any case, the i event will be replaced by a j event with high importance and high value, and the total value of the backpack will not be reduced. This optimized measure can reduce the type of events and speed up the efficiency of the algorithm execution. But in special circumstances (that is, for any two events, I, J, there is no $w_i \geq w_j, v_i \leq v_j$ relationship), No event can be removed at this time, and this measure will not optimize the time efficiency of the algorithm.

Because the structure of complete backpack is similar to that of 0/1 knapsack problem, there is only one event in 0/1 knapsack, but there are many events in complete knapsack, so we can transform the full knapsack into 0/1 knapsack problem to solve. Due to the limitation of backpack capacity, the i event in the complete knapsack problem can only be selected at $\lfloor w/w_i \rfloor$ times. Therefore, the i event can be transformed into the 0/1 knapsack problem, and the $\lfloor w/w_i \rfloor$ value is w_i value v_i, and the state transition equation is established as follows:

$$m(i,j)=\begin{cases}\max\{m(i-1,j)\ m(i-1,j-w_i)+v_i\} & j \geq w_i \\ m(i-1,j) & 0 \leq j \geq w_i\end{cases} \qquad (3)$$

The initial condition of the iteration is:

$$m(1,j)=\begin{cases} v_1, j \geq w_1 \\ 0, 0 \leq j < w_i \end{cases} \qquad (4)$$

But the above transformation method does not improve the time efficiency of the algorithm. A more efficient way of conversion is considered below, and a deduction is proved before discussing the specific transformation method. So we can deduce that any positive integer n can be decomposed into a form of

$n = 1 + 2 + 4 ... 2^{k-1} + n - 2^k + 1$ (where k is the largest integer that satisfies the $n - 2^k + 1 \geq 0$).Any positive integer c, where $c \in [1, n]$ can be represented as: $c = x_0 \times 1 + x_1 \times 2 + x_2 \times 4 + ... x^{k-} 1 + x^k \times (n - 2^k + 1)$, where $c \in \{0,1\}$. This proves that an arbitrary positive integer n can be decomposed into $n = 1 + 2 + 4 ... 2^{k-1} + n - 2^k + 1$ .The above mathematical inference is applied to transform full knapsack into 0/1 knapsack problem. The former way of transformation is to transform I event into $\lfloor w / w_i \rfloor$ event, which is wi value vi. Due to the establishment of the above inference, the transformation of the way is: after the conversion of each event value is $X_i \cdot v_i$, the importance of $X_i \cdot w_i$, where X_i is $1,2,4,...2^{k-1}, w / w_i - 2^k + 1$, this division, the I event is divided into $O(\log \lfloor w / w_i \rfloor)$ parts, the problem can be directly used for 0/1 knapsack problem dynamic programming algorithm, the time complexity of the proposed algorithm for $O(w \sum_{i=1}^{n} \log \lfloor w / w_i \rfloor)$ .This algorithm uses the binary idea to transform the complete knapsack problem into the 0/1 knapsack problem and then the algorithm is recorded as BDP. We give a brief description of the BDP mode. The following is the path problem that can be solved by BDP mode. Through BDP, we can get various paths between different locations.

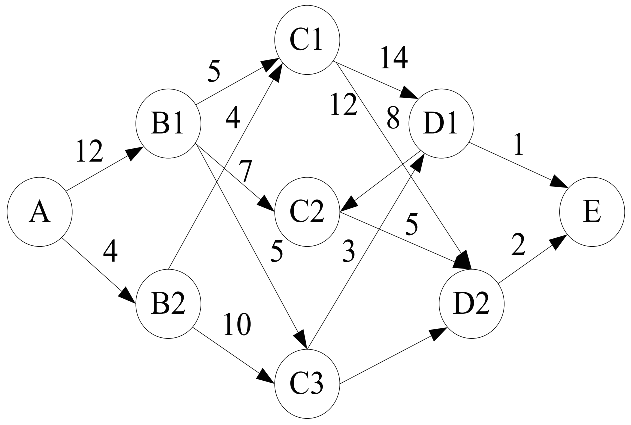

Figure 2
Path problem based on BDP mode

When calculating the sub problem $m(i, j)$, the NDP algorithm refers to the states that have been solved in $x = \lfloor j / w_i \rfloor$. In this paper, “each state transfer involved the state number optimization mentioned: to solve the problem with the dynamic programming algorithm is virtually the process calculation in the problem definition, calculation of the current state is often through solving the state and this state has done decision. Therefore, when calculating the state of each sub problem, the number of states involved in the algorithm will affect the time efficiency of the dynamic programming algorithm, and we can consider the reduction of the number of states involved in each state transfer to optimize the algorithm’s time efficiency. By analyzing the state transfer equation (1), it is found that when the operator problem $m(i, j)$ is considered, the equation refers to every decision of $x = 0,1,2,...,\lfloor j / w_i \rfloor$. Let $x = \lfloor j / w_i \rfloor$ be analyzed. It is found that the sub problem $m(i, j - w_j)$ that is solved when calculating the sub problem $m(i, j)$ is the maximum value of the first i-1 event and the i event of x-1 part based on the knapsack capacity $j - w_i$. The original state transfer equation (1) repeats a large number of states repeatedly when calculating the state of the sub problem, so the state transfer equation can be optimized as follows:

$$m(i, j) = \begin{cases} \max\{m(i-1, j), m(i, j - w_i) + v_i\} & j \geq w_i \\ m(i-1, j) & 0 \leq j \geq w_i \end{cases} \quad (5)$$

Based on the original state of the state transition equation (1) analysis found that each state transition involves a large number of invalid state of state decision dependencies, reducing the number of state each state transition equation in each state, the state transfer number involved is reduced from $O(w / w_i)$ to $O(1)$, the algorithm the time complexity of the optimization for $O(nw)$, said that the algorithm for the ODP (Optimization of Dynamic Programming) algorithm, as shown in Figure 3. Reducing the number of states involved in each state transfer in the algorithm is of great significance in the optimization of dynamic programming algorithms.

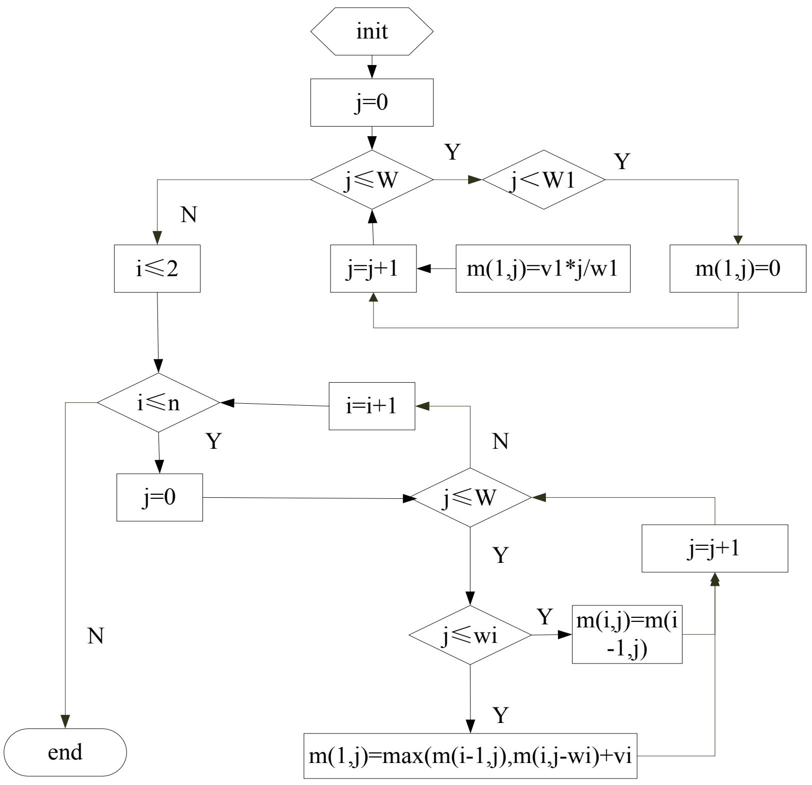

Figure 3
The flow chart of the ODP algorithm is presented in this article

In multiple knapsack problems, each event is not a number of pieces, but a i event has a choice of mi. In multiple knapsack, there are only $x_i = \min\{m_i, w/w_i\}$ decisions for i event: select 0 pieces, select 1 piece, select 2 pieces, and select xi pieces. The sub problem and the state transfer equation of the multiple knapsack problems are very similar to that of the complete knapsack problem. Therefore, the dynamic programming algorithm and optimization measures to solve the complete knapsack problem which can also be applied to solve multiple knapsack problems after proper modification, so this article will not repeat the details of solving multiple knapsack problems.

3. RESULT ANALYSIS AND DISCUSSION

Complete the knapsack problem in order to verify the dynamic programming optimization algorithm is correct and effective, this section will be randomly generated above 5 different types of data sets, so that three kinds of dynamic programming algorithm was run 10 times in the 5 different types of experimental data sets on average. Assuming the capacity of the knapsack W=1000 is to intuitively observe the time efficiency of the algorithm under various scale data sets, the following are illustrated in the form of graphs and tables.

Table 1
Operation Time/ms of the Algorithm When the Weight of the Item is in [150200) and [1200)

Scale	NDP	BDP	ODP	NDP	BDP	ODP
		[150,200)			1,200)	
100	115	70	54	119	69	56
200	233	138	111	233	146	116
300	344	209	170	358	221	169
400	466	280	221	475	294	228
500	574	354	281	5 85	358	282
600	693	416	335	715	446	350
700	810	487	392	833	508	413
800	921	556	448	984	604	474
900	1035	628	506	1063	654	529
1000	1148	698	560	1147	709	563
1100	1254	772	611	1258	778	617
1200	1365	838	661	1366	840	664

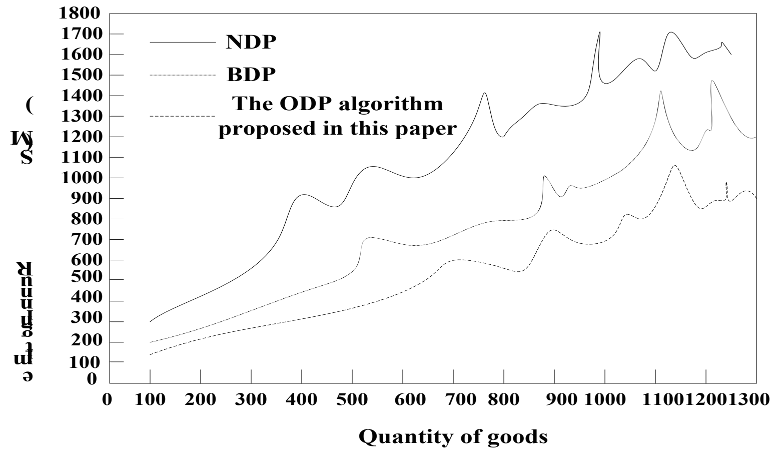

Figure 4
The running time of the algorithm when the size of the problem is different (the weight of the item in the [1200) distribution

From Table 1 and Figure 4, we can analyze from the following two aspects: first, the influence of the importance of events on the running time of NDP algorithm and BDP algorithm when solving the complete knapsack problem. The above experimental results show that when the two algorithms are used to solve the complete knapsack problem with different data classes (the importance distribution of events), when the importance of events increases, the time efficiency of algorithm will be improved accordingly. From Figure 5, we can see that when we solve the complete knapsack problem of the same data size, when the importance distribution of events is from [1,50 to [50100, [100150 to [1_50200, there will be some differences in the time efficiency between NDP and BDP algorithm. For example, when the number of events is n=100, the running time of NDP algorithm is from 1390ms, 241ms, 153ms to 11_Sms, and the running time of BDP algorithm has the same decreasing trend from 188ms, 116ms, 89ms to 70ms, and other data sizes. To further demonstrate this conclusion. we increase the importance distribution range of events from [1,10, [10,20, [20,30 to [190200, and then measure the running time of the algorithm separately, as shown in the table. In the table, the number of events was 1000, when the importance of the event distribution in [[1,10), long running NDP and BDP algorithm, respectively 4_5283ms and 2772ms, and with the distribution range of the importance of events gradually increased, the running time of the two algorithms will be decreasing, when the importance of the event distribution in [[190200), running time of NDP algorithm and BDP algorithm are respectively 119_Sms and 748ms. We also observed the relationship between the running time of the algorithm and the importance distribution of events. The results show that when the importance of events becomes larger, the running time of NDP and BDP algorithms is decreasing when solving the same size full knapsack problem.

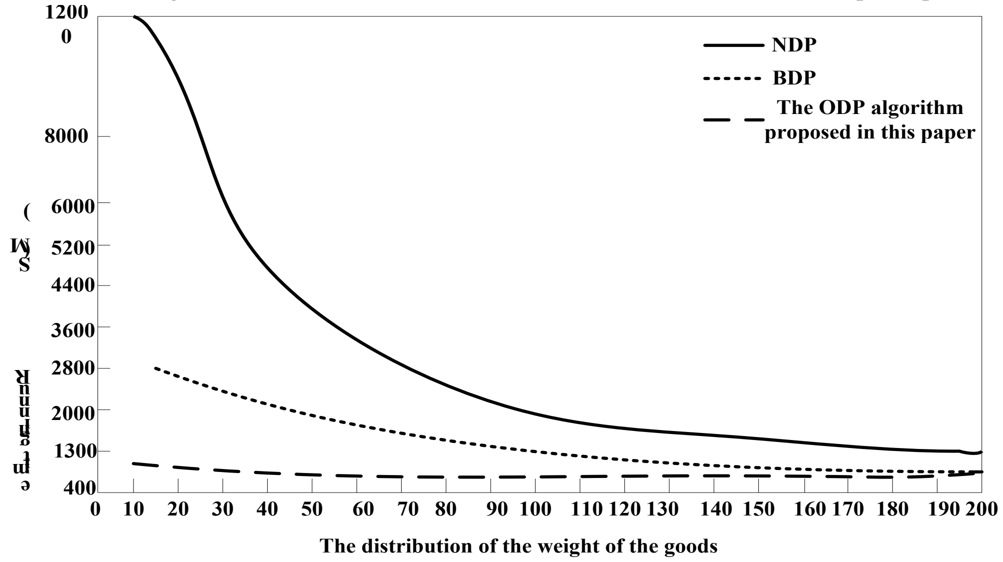

Figure 5
Solving the complete knapsack problem of the same size of data. The effect of the distribution of the weight on the time efficiency of the algorithm

The time complexity of the conventional dynamic programming algorithm, NDP, is directly dependent on the distribution of the importance of the event, as shown in formula (1). The BDP algorithm uses the binary idea to transform the complete knapsack problem into a 0/1 knapsack problem and then solve it again. While the size and importance of the event will affect the capacity of the knapsack that each event can be times into the backpack, which influence the problem into the total number of 0/1 knapsack problem after the event, and the dynamic programming algorithm for 0/1 knapsack problem time complex number degree and is directly related to the event. Therefore, in the BDP algorithm, the efficiency of the algorithm is related to the distribution of the importance of the event. The ODP algorithm proposed in this paper is based on the NDP algorithm to reduce the number of states that each state transfer depends on. The time efficiency of algorithm is not directly related to the importance of events. Therefore, the optimized algorithm ODP shows basically consistent performance in multiple tests with the same data size and different importance distributions.

The second aspect is the difference in time efficiency between different dynamic programming algorithms (NDP, BDP and ODP algorithm proposed in this paper) in solving the complete knapsack problem with the same data size and same data size. From Table 1, we can see that the number of events in the table indicates that the number of events remains unchanged, and the importance distribution of events increases sequentially, which affects the efficiency of various algorithms. We can compare with each table in the transverse, then we can found that regardless of the importance of the event distribution, this paper presents the ODP algorithm in time efficiency than the time and efficiency of NDP and BDP algorithm, the number of cases when the event n=100, the importance of events in [1, _50) randomly distributed, NDP, running time of BDP algorithm and ODP algorithm respectively. 1390ms, 188ms and _59ms; when the importance of the event in [1_50200) randomly distributed, the running time of the three algorithms are 115ms, 70ms and _54ms; when the importance of the event in [[1200) randomly distributed, the running time of the algorithm are 119ms, 69ms and _56ms, these three kinds of situations, compared ODP algorithm and NDP algorithm and BDP algorithm, the time efficiency is high, and the importance of NDP and BDP algorithm's efficiency depends on the incident, but the ODP algorithm in the event importance range, algorithm show stable performance. The vertical row in the table indicates the distribution of the distribution of the importance of the event, and the effect of increasing the number of events on the time efficiency of the algorithm. From the table, we can see that when the number of events increases, the running time of the three algorithms will increase, but in the same event scale, the running time of ODP algorithm is still smaller than the running time of NDP and BDP algorithm. It is shown that the time efficiency of ODP algorithm is higher than that of NDP and BDP algorithm when the number of events and the capacity of the knapsack change simultaneously in the complete knapsack problem.

In general, the BDP algorithm is higher than NDP time efficiency of the algorithm, because the BDP algorithm will be completely transformed into the knapsack problem 0/1 knapsack problem using the binary thinking, thus reducing the number of events after transformation and NDP algorithm can be understood as the problem of direct conversion. The proposed ODP algorithm is based on the analysis of the conventional NDP algorithm, found each state transition involves a large number of invalid state of state effective decision dependencies, reducing the number of state each state transition, which improves the time efficiency.

CONCLUSION

With the rapid development of social media, the speed of events spread faster and the public relations crisis is more prone. Based on this, this paper studies the public crisis management measures based on dynamic programming algorithm by using computer technology. First, a conventional dynamic programming algorithm RDP for solving 0/1 knapsack problem is presented. Then, by restricting the upper and lower bounds of the state in the RDP algorithm, the number of states that needs to be calculated is reduced, and the improved EDP algorithm is obtained. Then, it discusses the complete solving knapsack problem of general dynamic programming algorithm with binary NDP and ideas will be completely transformed into the knapsack problem 0/1 knapsack problem and then solve the BDP algorithm, through the analysis of the conventional NDP algorithm, each state transition in the discovery process involving a large number of invalid state, ODP algorithm analysis of effective state the decision dependencies are optimized, the ODP algorithm reduces the number of state each state transfer, improve the time efficiency of the algorithm. Finally, experiments are given to show the time efficiency of different algorithms in solving the same data size knapsack problem. Experimental results show that the improved algorithm proposed in this paper has higher time efficiency.

REFERENCES

Delipetrev, B., Jonoski, A., & Solomatine, D. P. (2015). A novel nested dynamic programming (nDP) algorithm for multipurpose reservoir optimization. *Journal of Hydroinformatics, 17*(4), 570-583.

Delipetrev, B., Jonoski, A., & Solomatine, D. P. (2017). A novel nested stochastic dynamic programming (nSDP) and nested reinforcement learning (nRL) algorithm for multipurpose reservoir optimization. *Journal of Hydroinformatics*, *19*(1), 47-61.

Hua, Q. S., Yu, Y. X., & Lau, F. C. M. (2010). Dynamic programming based algorithms for set multicover and multiset multicover problems. *Theoretical Computer Science, 411*(26), 2467-2474.

Jou Jonathan D., Jain Swati, J., Georgiev Ivelin S., Donald Bruce R. (2016). BWM*: A novel, provable, ensemble-based dynamic programming algorithm for sparse approximations of computational protein design. *Journal of computational biology: A journal of computational molecular cell biology*, *23*(6), 413-24.

Mehdi, S., Sahar, P., Shahriar, A., & Bijan, R. (2005). Prediction of protein secondary structure based on residue pair types and conformational states using dynamic programming algorithm. *FEBS Letters*, *579*(16), 397-400.

Pombeiro, H., Machado, M. J., & Silva, C. (2015). Dynamic programming algorithm for stochastic logical systems and its application to residual gas fraction control. *Proceedings of the ISCIE International Symposium on Stochastic Systems Theory and its Applications*, 136-141.

Tang, D., & Gupta, G. (1995). *An efficient parallel dynamic programming algorithm*. *Computers and Mathematics with Applications, 30*(8), 65-74.

Wong, W. CW., Wong, S, YS., Jaakkimainen, L., Bondy, S., Tsang, K. KA, Lee, A. (2005). SARS: lessons to learn for GPs when handling a public health crisis. *British Journal of General Practice, 55*(510), 57.

Wu, J. G., & Srikanthan, T. (2006). Low-complex dynamic the programming algorithm for the hardware/software partitioning. *Information Processing Letters*, *98*(2), 41-46.

Wu, Y. j., Wang, L., Zhu, D. X., & Wang, X. D. (2014). An efficient dynamic programming algorithm for the generalized LCS problem with multiple substring exclusive constraints. *Journal of Discrete Algorithms, 26*, 98-105.