Graphical Approach for Combinatorial Problems

A. A. Lazarev*, A. V. Baranov*

*Institute of Control Sciences RAS, lazarev@ipu.ru, av.baranov@physics.msu.ru

Dynamic programming is a general optimization technique developed by Bellman [1]. It can be considered as a recursive optimization procedure which interprets the optimization problem as a multi-step solution process. Bellman's optimality principle can be briefly formulated as follows: Starting from any current step, an optimal policy for the subsequent steps is independent of the policy adopted in the previous steps. In the case of a combinatorial problem, at some step j, $j = 2, \ldots, n$, sets of a particular size j are considered. To determine the optimal criterion value for a particular subset of size j, one has to know the optimal values for all necessary subsets of size $j-1$. If the problem includes n elements, the number of subsets to be considered is equal to $O(2^n)$. Therefore, dynamic programming usually results in an exponential complexity. However, if the problem considered is NP-hard in the ordinary sense, it is possible to derive pseudo-polynomial algorithms [2, 3, 4].

In this paper, we give the basic idea of a graphical modification of dynamic programming algorithm (DPA), which is called Graphical Algorithm (GA). This approach often reduces the number of it states to be considered in each step of the DPA. Moreover, in contrast to classical DPA, it can also treat problems with non-integer data without necessary transformations of the corresponding problem. In addition, for some problems, GA essentially reduces the time complexity.

For the knapsack problem DPA with the same idea like in GA are known (e.g. see [5]). In such DPA not all states $t \in [0, C]$ are considered, but only states, where a value of objective function is changed. Thus, the time complexity of such DPA is bounded by $O(nF_{opt})$, where F_{opt} is the optimal value of objective function. However, these algorithms can be useful only for problems, where $F_{opt} < C$, otherwise we can use the classical DPA. We generalize the idea of such algorithms for the objective function, for which $F_{opt} \gg C$. We have compared GA to algorithm $BalKnap$ [3].

1 Graphical Algorithm

Usually in DPA, we have to compute the value $f_j(t)$ of a particular function for each possible state t at each stage j of a decision process, where $t \in [0, C]$ and $t, C \in Z^+$. If this is done for any stage $j = 1, 2, \ldots, n$, where n is a size of the problem, the time complexity of such a DPA is typically $O(nC)$. However, often it is not necessary to store the result for any integer state since in the interval $[t_l, t_{l+1})$, we have a functional equation $f_j(t) = \varphi(t)$ (e.g. $f_j(t) = k_j \cdot t + b_j$, i.e., $f_j(t)$ a continuous linear function when allowing also real values t).

Assume that we have the following functional equations in a DPA, which correspond to Bellman's recursive equations:

$$f_j(t) = \min_{j=1,2,\ldots,n} \begin{cases} \Phi^1(t) = \alpha_j(t) + f_{j-1}(t - w_j) \\ \Phi^2(t) = \beta_j(t) + f_{j-1}(t - b_j) \end{cases} \tag{1}$$

with the initial conditions

$$\begin{array}{ll} f_0(t) = 0, & \text{for } t \geq 0, \\ f_0(t) = +\infty, & \text{for } t < 0. \end{array} \tag{2}$$

In (1), function $\Phi^1(t)$ characterizes a setting $x_j = 1$ while $\Phi^2(t)$ characterizes a setting $x_j = 0$

representing a yes/no decision, e.g. for an item, a job [2, 7].

To compute function $f_{j+1}(t)$, we compare two temporary functions $\Phi^1(t)$ and $\Phi^2(t)$.

The function $\Phi^1(t)$ is a combination of the terms $\alpha_{j+1}(t)$ and $f_j(t-w_{j+1})$. Function $f_j(t-w_{j+1})$ has the same structure, but all intervals $[t_l, t_{l+1})$ have been replaced by $[t_l - w_{j+1}, t_{l+1} - w_{j+1})$, i.e., we shift the graph of function $f_j(t)$ to the right by the value w_{j+1}. If we can present function $\alpha_{j+1}(t)$ in the form with μ_1 columns, we store function $\Phi^1(t)$ in the form with $m_j + \mu_1$ columns. In an analogous way, we store function $\Phi^2(t)$ in the form with $m_j + \mu_2$ columns.

Then we construct function

$$f_{j+1}(t) = \min\{\Phi^1(t), \Phi^2(t)\}.$$

For example, let the columns of Table $\Phi^1(t)$ contain the intervals

$$[t_0^1, t_1^1), [t_1^1, t_2^1), \ldots, [t^1_{(m_j+\mu_1)-1}, t^1_{(m_j+\mu_1)}]$$

and the columns of Table $\Phi^2(t)$ contain the intervals

$$[t_0^2, t_1^2), [t_1^2, t_2^2), \ldots, [t^2_{(m_j+\mu_2)-1}, t^2_{(m_j+\mu_2)}].$$

To construct function $f_{j+1}(t)$, we compare the two functions $\Phi^1(t)$ and $\Phi^2(t)$ on each interval, which is formed by means of the points

$$\begin{aligned}\{\quad & t_0^1, t_1^1, t_2^1, \ldots, t^1_{(m_j+\mu_1)-1}, t^1_{(m_j+\mu_1)}, \\ & t_0^2, t_1^2, t_2^2, \ldots, t^2_{(m_j+\mu_2)-1}, t^2_{(m_j+\mu_2)}\},\end{aligned}$$

and we determine the intersection points $t_1^3, t_2^3, \ldots, t^3_{\mu_3}$. Thus, in the table of function $f_{j+1}(t)$, we have at most $2m_j + \mu_1 + \mu_2 + \mu_3 \le C$ intervals.

In fact, in each step $j = 1, 2, \ldots, n$, we do not consider all points $t \in [0, C],\ t, C \in Z^+$, but only points from the interval in which the optimal partial solution changes or where the resulting functional equation of the objective function changes. For some objective functions, the number of such points M is small and the new algorithm based on this graphical approach has a time complexity of $O(n \min\{C, M\})$ instead of $O(nC)$ for the original *DPA*.

Moreover, such an approach has some other advantages.

1. The *GA* can solve instances, where p_j, w_j, $j = 1, 2, \ldots, n$, or/and C are not integer.

2. The running time of the *GA* for two instances with the parameters $\{p_j, w_j, C\}$ and $\{p_j \cdot 10^\alpha \pm 1, w_j \cdot 10^\alpha \pm 1, C \cdot 10^\alpha \pm 1\}$ is the same while the running time of the *DPA* will be 10^α times larger in the second case. Thus, using the *GA*, one can usually solve considerably larger instances.

3. Properties of an optimal solution are taken into account. For *KP*, an item with the smallest value $\frac{p_j}{w_j}$ may not influence the running time.

4. As we will show below, for several problems, *GA* has even a polynomial time complexity or we can at least essentially reduce the complexity of the standard *DPA*.

Thus, the use of *GA* can reduce both the time complexity and the running time for *KP*. The application of *GA* to the partition problem is described in detail in [6], where also computational results are presented.

2 Graphical Algorithm for the Knapsack Problem

In this section, we describe the application of this approach to the one-dimensional knapsack problem [6].

One-dimensional knapsack problem (*KP*): One wishes to fill a knapsack of capacity C with items having the largest possible total utility. If any item can be put at most once into the knapsack, we get the binary or $0-1$ knapsack problem. This problem can be written as the following integer linear programming problem:

$$\begin{cases} f(x) = \sum\limits_{j=1}^{n} p_j x_j \to \max \\ \sum\limits_{j=1}^{n} w_j x_j \le C, \\ x_j \in \{0, 1\},\ j = 1, 2, \ldots, n. \end{cases} \tag{3}$$

Here, p_j gives the utility and w_j the required capacity of item j, $j = 1, 2, \dots, n$. The variable x_j characterizes whether item j is put into the knapsack or not.

The DPA based on Bellman's optimality principle is one of the standard algorithms for the KP. It is assumed that all parameters are positive integer: $C, p_j, w_j \in Z^+, j = 1, 2, \dots, n$.

For KP, Bellman's recursive equations are as follows:

$$f_j(t) = \max_{j=1,2,\dots,n} \begin{cases} \Phi^1(t) = p_j + f_{j-1}(t - w_j) \\ \Phi^2(t) = f_{j-1}(t), \end{cases} \tag{4}$$

where

$$\begin{aligned} f_0(t) &= 0, \quad t \geq 0, \\ f_0(t) &= +\infty, \ t < 0. \end{aligned}$$

$\Phi^1(t)$ represents the setting $x_j = 1$ (i.e., item j is put into the knapsack) while $\Phi^2(t)$ represents the setting $x_j = 0$ (i.e., item j is not put into the knapsack). In each step j, $j = 1, 2, \dots, n$, the function values $f_j(t)$ are calculated for each integer point (i.e., "state") $0 \leq t \leq C$. For each point t, a corresponding best (partial) solution $X(t) = (x_1(t), x_2(t), \dots, x_j(t))$ is stored.

Exploring the algorithm can be seen that the j-th step of the algorithm should calculate values of objective function only in range of $\max\{0, C - \sum\limits_{k=j+1}^{n} w_k\}$ to $\min\{C, \sum\limits_{k=1}^{j} w_k\}$

Taking into account this feature in DPA allow decrease the number of calculating points due dynamic range of computing change. Shifted to the left and right boundaries of the interval.

Using the shift of left border in the GA allows to essentially reduce the number of break points and consequently the number of "dummy" operations. Shift of right border in GA is obtained in similar manner.

Let consider calculations for DPA, modified DPA ($BalKnap$ [3]) and GA for follow instance:

$$\begin{cases} 5x_1+7x_2+6x_3+5x_4+4x_5+8x_6+6x_7 \to \max \\ 2x_1+3x_2+5x_3+4x_4+2x_5+3x_6+4x_7 \leq 10, \\ x_j \in \{0,1\}, \, j=1,\dots,7. \end{cases} \tag{5}$$

For DPA it should calculate table with 10×7 elements.

C	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1	0	5	5	5	5	5	5	5	5	5
2	0	5	7	7	12	12	12	12	12	12
3	0	5	7	7	12	12	12	13	13	18
4	0	5	7	7	12	12	13	13	17	18
5	0	5	7	9	12	12	16	16	17	18
6	0	5	8	9	13	15	17	20	20	24
7	0	5	8	9	13	15	17	20	20	24

For modified DPA it should calculate 41 values.

C	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1	0	5								
2	0	5	7	7	12					
3	0	5	7	7	12	12	12	13	13	18
4	0	5	7	7	12	12	13	13	17	18
5			7	9	12	12	16	16	17	18
6						15	17	20	20	24
7										24

And GA calculate only points with provide optimal solution. For this example it's necessary just 14 elements.

C	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1	0	5								
2			7		12					
3								13		18
4							13		17	
5				9			16			
6						15	17	20		24
7										

So on this instance we obtain reduce calculation in 80% for GA in contrast with DPA.

3 Concluding Remarks

The graphical approach can be applied to problems where a pseudo-polynomial algorithm exists and Boolean variables are used in the sense that yes/no decisions have to made (e.g. in the problem under consideration, for KP, an item can be put into the knapsack or not), for example for partition and scheduling problems [6, 7]. However, for the knapsack problem, the graphical algorithm mostly reduces substantially the number of points to be considered but the time complexity of the algorithm remains pseudo-polynomial.

This approach can be used to k-dimensional Knapsack Problem too.

Thus, the graphical approach has not only a practical but also a theoretical importance. Furthermore, graphical algorithm could be efficiently implemented to various parallel architectures.

Acknowledgements

Partially supported by programs of Russian Academy of Sciences and RFBR 11-08-01321. We would like to thank Prof. Dr. Frank Werner and Dr. Evgeny Gafarov for many discussions and helpful suggestions.

References

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

[2] E.R. Gafarov, A.A. Lazarev, F. Werner, *Algorithms for Some Maximization Scheduling Problems on a Single Machine.* Automation and Remote Control, Vol. 71, No. 10, 2070–2084, 2010.

[3] H. Keller, U. Pferschy, D. Pisinger, *Knapsack Problems.* Springer, Hidelberg, 2010.

[4] E.L. Lawler, J.M. Moore, *A Functional Equation and its Application to Resource Allocation and Sequencing Problems.* Management Science, Vol. 16, No. 1, 77–84, 1969.

[5] Ch. Papadimitrou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity.* Dover Publications, INC, Mineola, New York, 1998.

[6] A.A. Lazarev, F. Werner, *A Graphical Realization of the Dynamic Programming Method for Solving NP-Hard Combinatorial Problems*, Computers and Mathematics with Applications, Vol. 58, No. 4, 619–631, 2009.

[7] A.A. Lazarev, F. Werner, *Algorithms for Special Cases of the Single Machine Total Tardiness Problem and an Application to the Even-Odd Partition Problem.* Mathematical and Computer Modelling, Vol. 49, No. 9–10, 2061–2072, 2009.