Acta Univ. Sapientiae Informatica **13,** 2 (2021) 361–372

DOI: 10.2478/ausi-2021-0017

DP-solver: automating dynamic programming

Zoltan KATAI
Sapientia Hungarian University of Transylvania
Department of Mathematics and Informatics
Tirgu Mures, Romania
email: katai_zoltan@ms.sapientia.ro

Attila ELEKES
Sapientia Hungarian University of Transylvania
email: attilaelekes97@gmail.com

Abstract. Dynamic programming (DP) is a widely used optimization method with several applications in various fields of science. The DP problem solving process can be divided in two phases: mathematical part and programming part. There are a number of researchers for whom the mathematical part is available, but they are not familiar with computer programming. In this paper we present a software tool that automates the programming part of DP and allows users to solve problems based only on their mathematical approach. The application builds up the "d-graph model" of the problem to be solved and applies the "d-variant" of the corresponding single source shortest path algorithm. In addition, we report experimental results regarding the efficiency of the tool relative to the Matlab implementation.

1 Introduction

Dynamic programming (DP) is an optimization method used in numerous fields of science. It was proposed by Richard Bellman in his book published

Computing Classification System 1998: F.2.2
Mathematics Subject Classification 2010: 68R15
Key words and phrases: dynamic programming, d-graphs, software tool

in 1957 [1]. Since that, this strategy has become an often used problem solving method in applied mathematics, computer sciences, artificial intelligence, bioinformatics, macroeconomics, etc. The solving process of these kinds of problems can be divided into two major steps. Firstly, a functional equation is established, which describes the problem solving process in a recursive way by implementing the principle of optimality. Secondly, a computer program is implemented, which processes the recursive formula in a bottom-up way. In this paper we will refer to these two phases as mathematical and programming parts of DP.

Due to the diversity of applications of this optimization method, the programming part of DP may cause difficulties for researchers who are not familiar with the programming languages or scripts. In this paper we present a software tool that automates the programming part of DP and allows users to solve problems based only on their mathematical approach.

2 Literature review

As mentioned above, in the last more than half century DP has been applied in several fields of science. More recently, for example, it was applied for solving the k-Color Shortest Path Problem that arises in the field of transmission networks design [3]. The authors of this study compared their DP approach with two previously published methods and they found that the DP algorithm vastly outperformed previous approaches. The authors of paper [18] present a survey of financial applications under a specific semimartingale result of Markov chains and two of the described strategies apply DP approach. Besides, other recent studies apply DP in the field of genomics and bioinformatics [15], for developing modern rainwater harvesting systems [17], for optimizing the synchronization and reducing gear-shifting time in mechanical transmissions [13], and even for solving the School Bus Routing Problem [19].

As DP became more widespread, software tools for solving (more or less automatically) certain DP problem families began to appear. For example, the Stochastic Dynamic Programming application (published in 1995) was a commercial software for solving general stochastic and deterministic optimization problems [14]. This tool (running on PCs using DOS operating system) was able (i) to accommodate user-specified conditions and functions of stage, state and decision; (ii) to minimize or maximize the optimal value function; (iii) to solve finite and infinite time-horizon problems, etc. Another software tool for solving DP problems is DP2PN2Solver [4]. This application uses specialized

Petri nets, named Bellman nets, to solve DP problems. In addition, it should also be mentioned the tool named MDPToolbox, which has been developed since 2004 [2]. MDPToolbox is a set of functions for solving Markov-decision Problems in various platforms like Matlab, Scilab, R and GNU Octave.

De Moor [16] also emphasizes the need for DP solvers. After stating that in contrast with other techniques (such as linear programming, where there exists a single generic program that solves all instances) DP is usually regarded as a design technique where each application is designed as an individual program, he argues that it would be much preferable if dynamic programming could be understood as a software component. The component presented by De Moor is suitable for a large class of applications in which the decision process is a sequential scan of the input sequence. With respect to the programming part he introduces a C++ and Haskell program and concludes that the simplicity offered by lazy functional programming is preferable. Another useful tool is VisuAlgo [24] which is a website for visualizing different algorithms including DP strategies. This tool was developed mostly for educational purposes and also uses a programming language (JavaScript) instead of mathematical functional equations.

Relatively recent studies were realized in this field by Kátai [5, 6, 7, 8, 10, 12]. In [5], he proposed the concept of $\mathfrak{d}$-graphs for modelling DP problems. The software tool presented in [9] was based on this approach. Later, Katai refined the model and introduced the notion of generalized $\mathfrak{d}$-graphs [5]. Katai and Fulop [11] compared Petri nets (used by the above referred DP2PN2Solver) and $\mathfrak{d}$-graphs as tools for intermediate representation of certain DP problems. They emphasize two advantages of $\mathfrak{d}$-graphs: (i) $\mathfrak{d}$-graphs can handle problems with cyclic target function; (ii) while the usage of DP2PN2-Solver requires the knowledge of gDPS language, $\mathfrak{d}$-graphs can be built based on the functional equation. In [12], Katai defines the so-called generalized deterministic Markov decision processes where each decision may result in more than one next state. The author also proposed a combined $\mathfrak{d}$-graph algorithm for finding optimal policies for the previously mentioned Markov processes.

From this enumeration the probably most generally known and commonly used software cannot be missed. The Matlab is an available commercial software, which is able to perform numeric computations and to visualize data structures and models. There are also a lot of available plugins for solving various problems, including DP. The programming language used by Matlab is also relatively simple and has a high abstraction level.

In this paper we present a simplified implementation of the approach Katai proposed [10, 12], and more importantly, we present a software tool that implements this strategy and also visualizes the DP solving process. In addition, we report experimental results regarding the efficiency of the tool we developed (compared with the Matlab solutions).

3 Mathematical part of the DP solving process

The core of DP is the principle of optimality, which states that the optimal solution for the initial problem is based on the optimal solutions of its subproblems. This principle is expressed by a recursive formula (target function). Subproblems are solved in bottom-up order (starting from the level of the trivial ones), and the optimal solution of the current subproblem is computed (based on the recursive formula) from the optimal solutions of previously solved (and stored) simpler subproblems. The solving process is a sequence of optimal decisions (min or max) that results in the optimal solution of the original problem. To illustrate this approach we present the strategy on a sample problem.

Two Person Game: Let us consider a sequence of natural numbers stored in array $a_i, i = 0, \ldots, n-1$ (where n is even). In every step the current player takes a number from the beginning or the end of the sequence. The question is, what is the highest score which can be collected by the beginner? We assume that both players are taking the optimal decision.

Assuming that cell c_{ij} of array c $(i, j = 0, \ldots, n-1)$ stores the optimal value associated to sub-sequence $a_i \ldots a_j$, the functional equation for this problem is the following:

$$c_{ij} = \begin{cases} 0 & j \le i \\ \max\{a_i + \min\{c_{i+1,j-1}, c_{i+2,j}\}, & \\ \qquad a_j + \min\{c_{i+1,j-1}, c_{i,j-2}\}\} & i < j \end{cases}$$

In this problem the sub-problems are represented by the even length sub-sequences of the original sequence. More precisely, cell $c_{i,j}$ stores the maximum score that can be collected by the first player if they play on sub-sequence $a_i \ldots a_j$. The solving process implies the computation of the highest scores which the first player can collect from all 2, 4, ..., n length sub-sequences. Function max reflects that player 1 takes optimal decisions. Function min reflects that player 2 also takes optimal decisions.

4 The idea behind the software

As mentioned above, Kátai [10] proposed generalized d-graphs as intermediate representations to solve DP problems based on the optimal path algorithms in weighted graphs. There are three main single-source shortest path algorithms in weighted digraph:

- If the graph is cycle free, the optimal paths can be found based on the topological order of the vertices (Viterbi algorithm);
- If the graph contains circles and all arcs have positive weights, then we use the Dijkstra algorithm;
- If the graph contains negative weighted arcs, but does not contains "negative cycles" (the sum of the weights of the arcs of the cycle is negative), the problem is solved by the Bellman-Ford algorithm.

Katai [7, 10] emphasizes that, on the one hand, these algorithms implement DP strategies and, on the other hand, the DP problem solving process can be reduced to these strategies. Katai suggests the following approach (for details see [10]): (i) the DP problem to be solved is modelled with a d-graph where p-vertices represent the sub-problems and d-vertices represent the possible choices defined by the target function; (ii) the optimal decision sequence is found by searching for the optimal d-path between the source vertex, representing the original problem, and a virtual sink vertex, representing the trivial sub-problems. In the second step the d-variant of the corresponding optimal path algorithm is applied (d-Viterbi, d-Dijkstra, d-BellmanFord).

5 DP-solver

DP-solver is a software tool for solving DP problems based only on the mathematical functional equation. To automate the programming part of the DP solving process it takes advantage of the relationship between dynamic programming and optimal paths algorithms. The tool is easy to use in the sense that it does not presume to learn any programming language and the input format is very similar to the mathematical syntax. Another functionality is the visualization of the solving process. This feature uses 1D and 2D arrays, where the cells represent the sub-problems. Thus, DP-solver is both a scientific and educational tool.

The application was implemented using the JavaFX language and its core is a Java package, named exp4j [20], which is able to interpret mathematical expressions. From developer point of view, exp4j is an easy to use package which knows the fundamental mathematical operators and functions and it is able to work with variables and general formulas and to detect syntactic and runtime errors. Since it is easy to define more operators and functions, the package could comfortably be adapted to the current software requirements.

As mentioned above, the main functionality of the software is to solve DP problems based on the mathematical equation. Accordingly, the user's role includes the following steps: (i) to introduce the formula and its variables, and to ask the tool (ii) to evaluate the formula and (iii) to visualize the solving process. The application also enables users to save, reload and edit previously created DP models. In visualization mode the software works similarly to a media player.

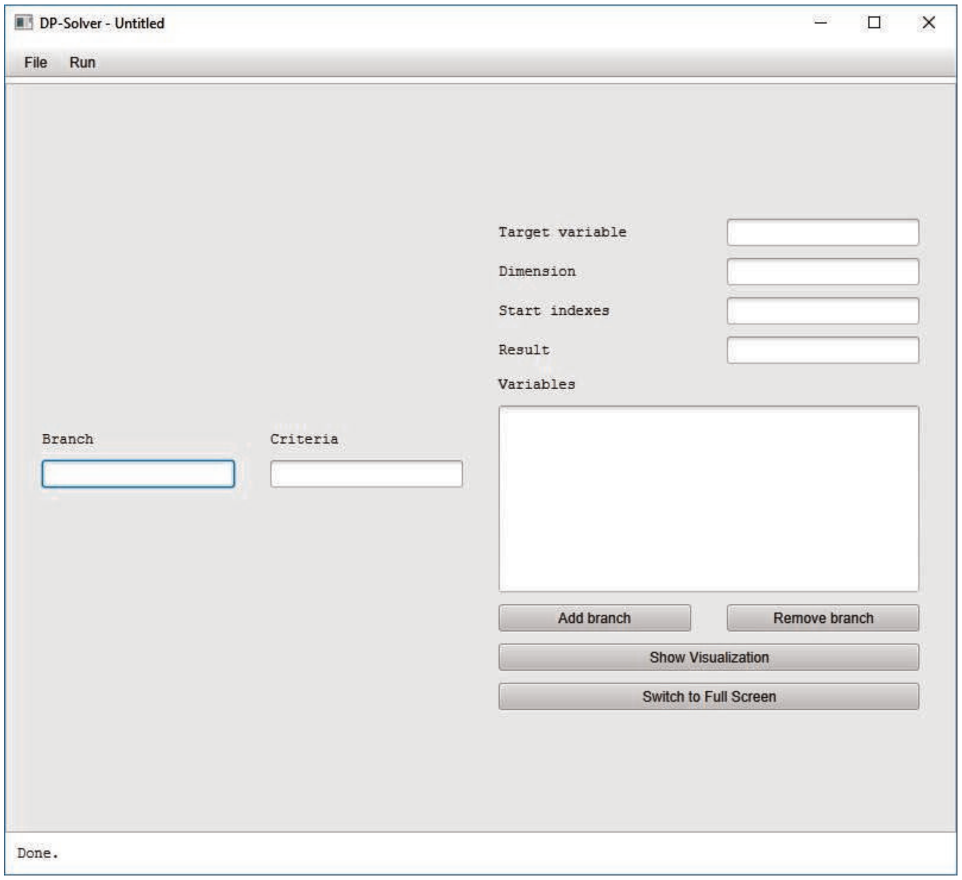


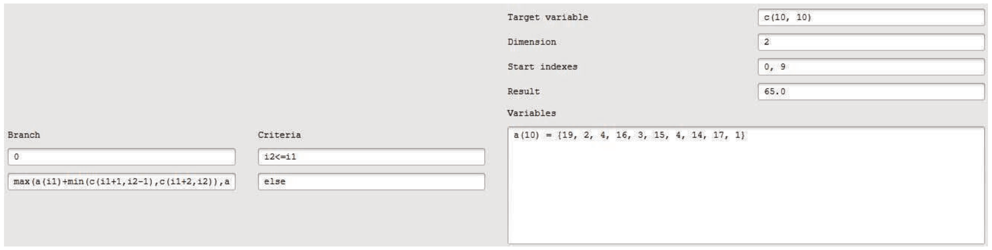
Figure 1: The main window of DP-solver

6 The graphical interface

Figure 1 shows the graphical interface of DP-solver. In the input fields labeled with "Branch" and "Criteria" the user has to introduce the formula, branch-by-branch. Attached to each branch, the corresponding criterion has to be also introduced.

In the "Target variable" field the user has to specify the array which stores the optimal values corresponding to the optimal solutions of the subproblems. The dimension of the problem (dimension of the target variable) and the indices of the cell that represents the optimal value of the original problem ("Start indices") are also required to be specified. In the "Variables" field the input variables (which are included in the formula) have to be defined. Before the evaluation starts the software checks all input fields for syntax errors. The optimal value attached to the optimal solution is displayed in the field named "Result".

The status bar (on the bottom of the window) informs the user about the state of the problem solving process. For example, the user is informed about the currently performed task and error messages also appears here.



Target variable	c(10, 10)
Dimension	2
Start indexes	0, 9
Result	65.0

Variables

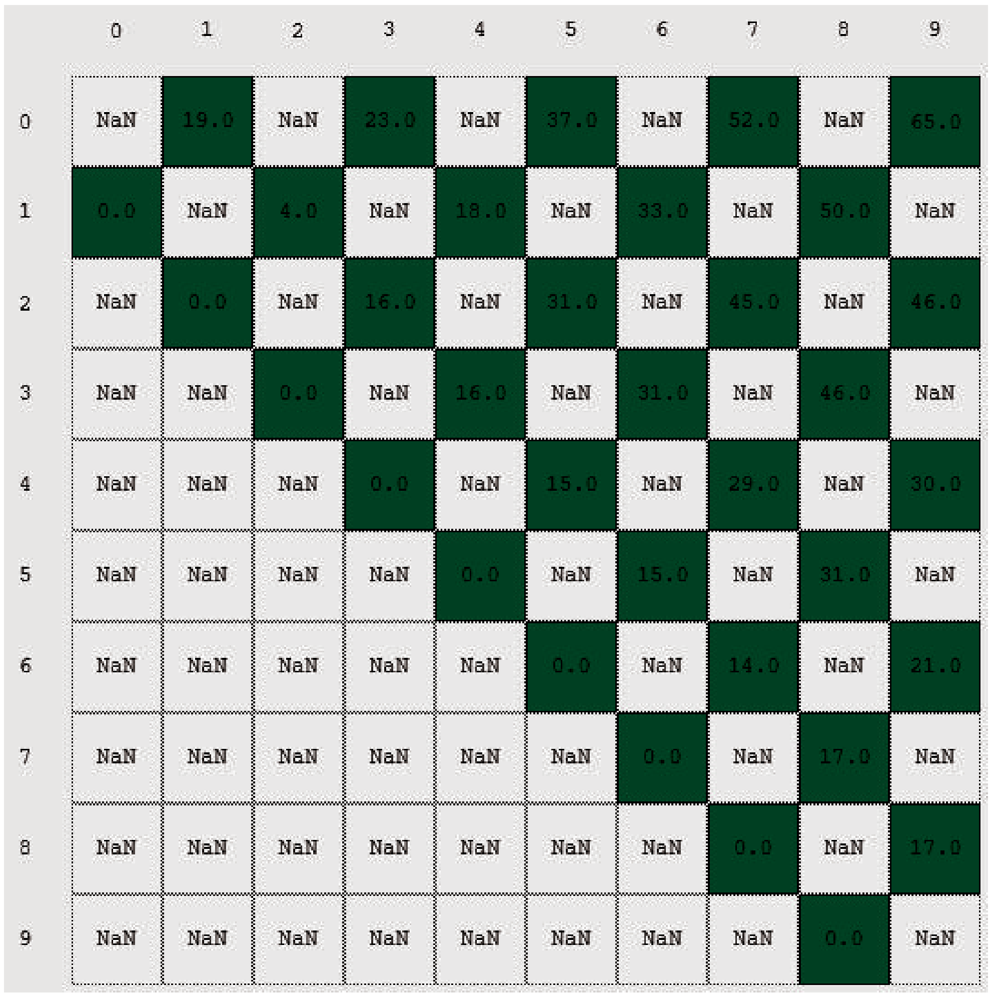
```
a(10) = {19, 2, 4, 16, 3, 15, 4, 14, 17, 1}
```

Branch	Criteria
0	i2<=i1
max(a(i1)+min(c(i1+1,i2-1),c(i1+2,i2)),a	else

Figure 2: Input and output of the Two Person Game problem

Figure 2 illustrates the usage of DP-solver for the above presented Two Person Game problem for a 10-length number sequence. As it could be noticed, the recursive formula has two branches, the target variable is a 10x10 bi-dimensional array and cell c_{09} represents the original problem. If the input sequence stored in array a is $\{19, 2, 4, 16, 3, 15, 4, 14, 17, 1\}$ then the optimal value which is going to appear (after pushing the "Run" button) in the "Result" field is 65.

After the result appears, the solving process can be visualized step by step. Since sub-problems correspond to the even length sub-sequences, only every second diagonal of the target variable is filled (green cells containing non-NaN values) (see Fig. 3). The diagonal below the main diagonal stores the optimal

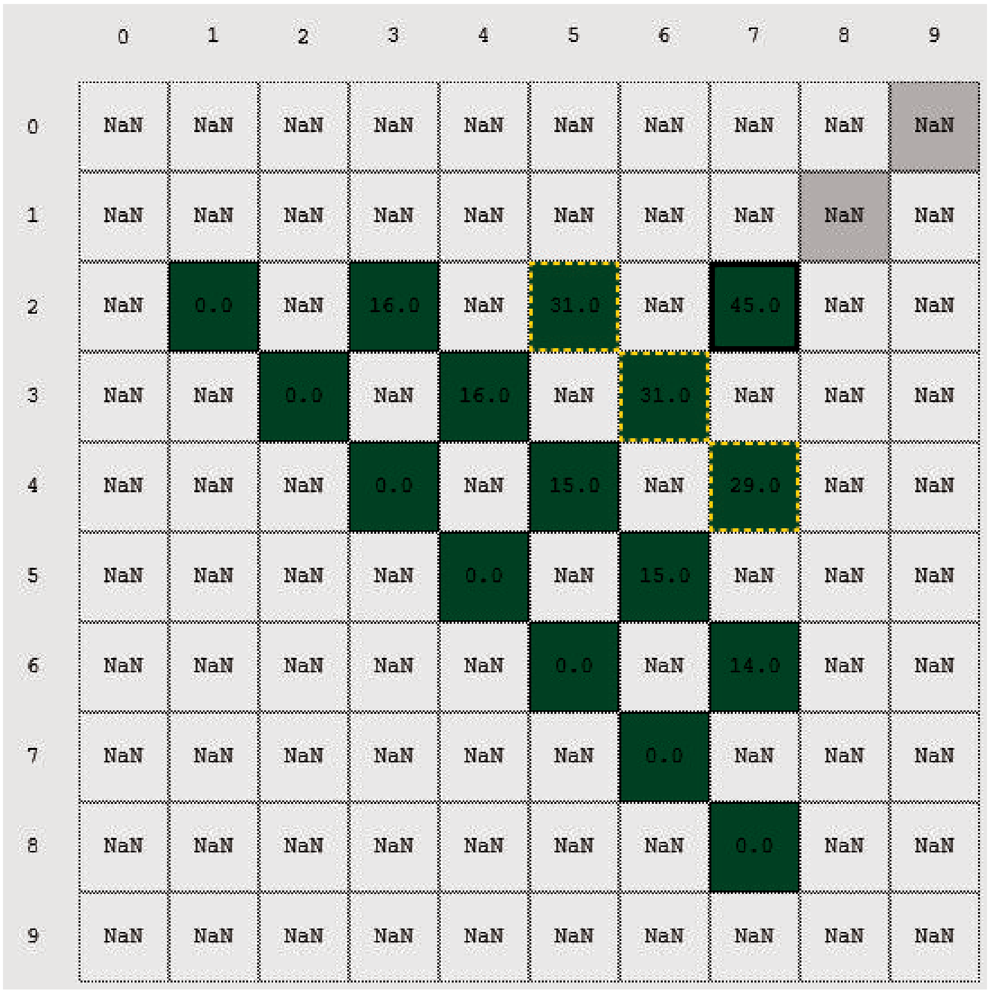


	0	1	2	3	4	5	6	7	8	9
0	NaN	19.0	NaN	23.0	NaN	37.0	NaN	52.0	NaN	65.0
1	0.0	NaN	4.0	NaN	18.0	NaN	33.0	NaN	50.0	NaN
2	NaN	0.0	NaN	16.0	NaN	31.0	NaN	45.0	NaN	46.0
3	NaN	NaN	0.0	NaN	16.0	NaN	31.0	NaN	46.0	NaN
4	NaN	NaN	NaN	0.0	NaN	15.0	NaN	29.0	NaN	30.0
5	NaN	NaN	NaN	NaN	0.0	NaN	15.0	NaN	31.0	NaN
6	NaN	NaN	NaN	NaN	NaN	0.0	NaN	14.0	NaN	21.0
7	NaN	NaN	NaN	NaN	NaN	NaN	0.0	NaN	17.0	NaN
8	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.0	NaN	17.0
9	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.0	NaN

Figure 3: The completed visualization process (Two Person Game problem)

values for the trivial sub-problems (corresponding to the 0-length sequences). The optimal values of "non-trivial cells" are computed from diagonal to diagonal. Lastly, cell c_{09} gets the value 65.

The screenshot from Figure 4 captures the solving process in an intermediate step. By default every cell has a white background color and dotted border. The default value of each cell is NaN until the result of the corresponding sub-problem is computed. Accessed cells get a light gray background color (the solving process reached them, but their values are not computed yet). The



	0	1	2	3	4	5	6	7	8	9
0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	NaN	0.0	NaN	16.0	NaN	31.0	NaN	45.0	NaN	NaN
3	NaN	NaN	0.0	NaN	16.0	NaN	31.0	NaN	NaN	NaN
4	NaN	NaN	NaN	0.0	NaN	15.0	NaN	29.0	NaN	NaN
5	NaN	NaN	NaN	NaN	0.0	NaN	15.0	NaN	NaN	NaN
6	NaN	NaN	NaN	NaN	NaN	0.0	NaN	14.0	NaN	NaN
7	NaN	NaN	NaN	NaN	NaN	NaN	0.0	NaN	NaN	NaN
8	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.0	NaN	NaN
9	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Figure 4: The visualization process in progress (Two Person Game problem)

current cell is marked with black thick solid border and dark gray background color. The cells representing the direct descendants of the current subproblem (the value of the current cell is computed based on these cells' values) are highlighted with a yellow dashed border. When a subproblem is solved, the related cell turns its border to solid and its background color to green. The buttons for controlling the visualization process are in the bottom left corner of the main window.

7 Performance

To find out our software's performance, we compared DP-Solver with Matlab®. We have chosen Matlab®, because it is one of the most commonly used, well known and widely available programming software used for mathematical problem solving. In our tests we used the Matlab® R2017b version. The laptop we used has the following configuration: Intel® Celeron® N2940 CPU (quad core, up to 2.23 GHz), 4 GB DDR3L memory and Samsung 860 EVO SSD.

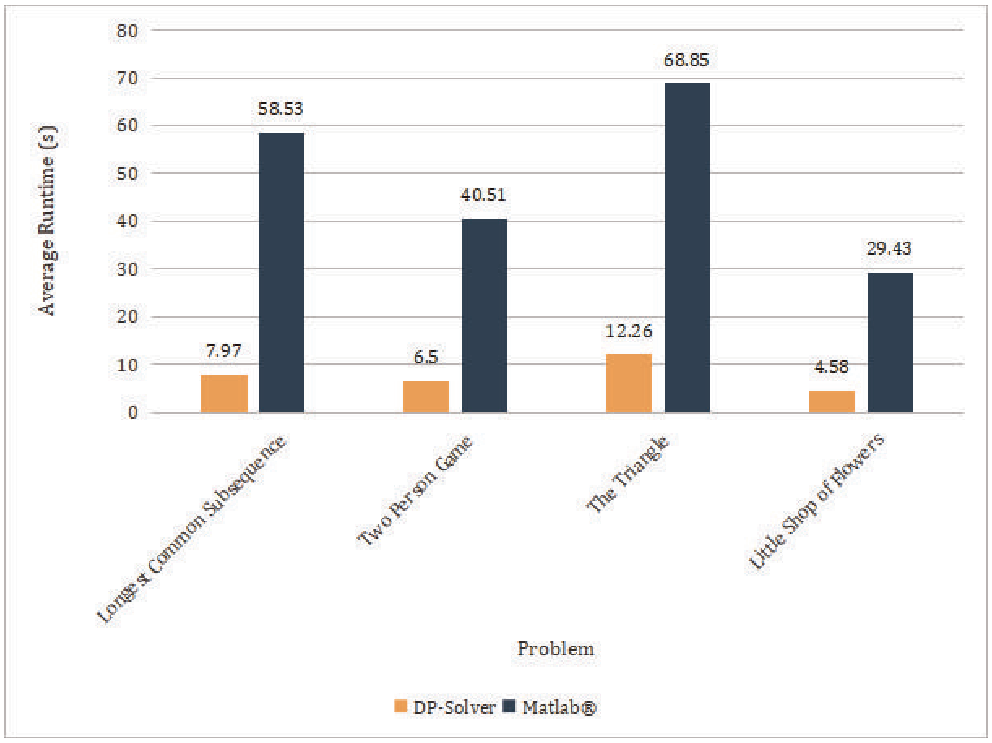


Figure 5: DP-solver vs. Matlab® results

We performed the comparison for the DP solutions of the problems referred below (for details see the corresponding references). The analysis was based on the average run-time after 10 measurements (the brackets include the size of the corresponding one- or bi-dimensional input array(s)):

- Longest Common Subsequence [23] ([1000], [500])

- Two Person Game ([1000])
- The Triangle [21] ([1000]x[1000])
- Little Shop of Flowers [22] ([750]x[1000])

As shown in Figure 5 DP-solver was 6.1 times faster than the Matlab® implementations of the DP solutions.

8 Conclusions

In this paper we presented DP-solver, an easy to use software tool for those who are familiar with the mathematical fundamentals of dynamic programming. The application uses as intermediate representation the d-graph model of the DP problem to be solved. Our experimental results shows that the tool can provide the optimal solution more than six times faster than the corresponding Matlab implementation. The application also includes a visualization module. Consequently, DP-solver is both a scientific and educational software tool.

References

[1] R. Bellman, *Dynamic Programming*, Princeton University Press, New Jersey, 1957. ⇒362

[2] I. Chadès, G. Chapron, M. J. Cros, F. Garcia, R. Sabbadin, MDPtoolbox: a multi-platform toolbox to solve stochastic dynamic programming problems, *Ecography*, **37,** 9 (2014) 916–920. ⇒363

[3] D. Ferone, P. Festa, S. Fugaro, T. Pastore, A dynamic programming algorithm for solving the k-color shortest path problem, *Optimization Letters*, (2020) 1–20. ⇒362

[4] M. Holger, DP2PN2Solver: A flexible dynamic programming solver software tool, *Control and Cybernetics*, **35** (2002) 687–702. ⇒362

[5] Z. Kátai, Dynamic programming and d-graphs, *Studia Universitatis Babes-Bolyai, Informatic*, **51,** 2 (2006) 41–52. ⇒363

[6] Z. Kátai, Dynamic programming strategies on the decision tree hidden behind the optimizing problems, *Informatics in Education - An International Journal*, **6,** 1 (2007) 115–138. ⇒363

[7] Z. Kátai, The single-source shortest paths algorithms and the dynamic programming, *Teaching Mathematics and Computer Science*, **6** (2008) 25–35. ⇒363, 365

[8] Z. Kátai, Dynamic programming as optimal path problem in weighted digraph, *Acta Mathematica Academiae Paedagogicae Nyíregyháziensis*, **24** (2008) 201–208. ⇒363

[9] Z. Kátai, Á. Csíki, Automated dynamic programming, *Acta Universitatis Sapientiae, Informatica*, **1,** 2 (2009) 149–164. ⇒363

[10] Z. Kátai, Modelling dynamic programming problems by generalized d-graphs, *Acta Universitatis Sapientiae, Informatica*, **2,** 2 (2010) 210–230. ⇒363, 365

[11] Z. Kátai, P. I. Fülöp, Modeling dynamic programming problems: Petri nets versus d-graphs, *Proceedings of 8th International Conference on Applied Informatics (ICAI)*, Eger, Hungary, 2010, pp. 217–226. ⇒363

[12] Z. Kátai, Solving Markov decision processes by d-graph algorithms, *Control and Cybernetics*, **41,** 3 (2012) 577–593. ⇒363

[13] Z. Lu, G. Tian, S. Onori, Multistage time-optimal control for synchronization process in electric-driven mechanical transmission with angle alignment considering torque response process, *Journal of Dynamic Systems, Measurement, and Control*, **143,** 4 (2021) 041006. ⇒362

[14] B. C. Lubow, SDP: generalized software for solving stochastic dynamic optimization problems, *Wildlife Society Bulletin*, (1995) 738–742. ⇒362

[15] M. Maiolo, S. Ulzega, M. Gil, M. Anisimova, Accelerating phylogeny-aware alignment with indel evolution using short time Fourier transform, *NAR Genomics and Bioinformatics*, **2,** 4 (2020) lqaa092. ⇒362

[16] O. De Moor, Dynamic programming as a software component, *Proceedings of IEEE Computer Society, Conference on Circuits, Systems, Computers and Communications (CSCC)*, Athens, Greece, 1999. ⇒363

[17] C. Nop, R. M. Fadhil, K. Unami, A multi-state Markov chain model for rainfall to be used in optimal operation of rainwater harvesting systems, *Journal of Cleaner Production*, (2020) 124912. ⇒362

[18] E. Savku, G. W. Weber, A regime-switching model with applications to finance: markovian and non-markovian cases, *Dynamic Economic Problems with Regime Switches*, Springer, Cham, 2021, pp. 287–309. ⇒362

[19] P. Shang, L. Yang, Z. Zeng, L. C. Tong, Solving school bus routing problem with mixed-load allowance for multiple schools, *Computers & Industrial Engineering*, (2020) 106916. ⇒362

[20] ∗ ∗ ∗ exp4j, https://lallafa.objecthunter.net/exp4j ⇒365

[21] ∗ ∗ ∗ International Olympiad in Informatics (IOI), The Triangle, 1994. ⇒371

[22] ∗ ∗ ∗ International Olympiad in Informatics (IOI), Little Shop of Flower, 1999. ⇒371

[23] ∗ ∗ ∗ Longest common subsequence problem ⇒370

[24] ∗ ∗ ∗ VisuAlgo, https://visualgo.net/en ⇒363

Received: December 10, 2021 • Revised: December 26, 2021