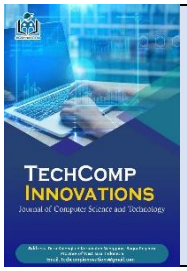

Journal of Computer Science and Technology, Vol 2 (2) 2025: 131-138

TechComp Innovations

https://mabadiiqtishada.org/index.php/TechCompInnovations

ISSN: 3062-8903; e-ISSN: 3062-889X

An Accelerated Dynamic Programming Framework Based on Quadrilateral Inequality and Convex Monotonicity Theory

Kwon Jun Kim[1], Ung Bom So[2], Hyon Chol Kim[3*]

[123] *Kim Chaek University of Technology. Korea, Democratic People's Republic of*

ARTICLE INFO

Received October 15, 2025
Accepted Dec 22, 2025
Published Dec 23, 2025

Keywords:
Dynamic Programming;
Convex Monotonicity;
Quadrilateral Inequality;
Algorithm Acceleration;
Computational Complexity

ABSTRACT

Dynamic programming (DP) is a foundational technique for solving multistage optimization problems; however, its classical formulation often incurs a cubic time complexity, rendering it impractical for large-scale real-world applications. This study proposes an accelerated dynamic programming framework that leverages quadrilateral inequality and convex monotonicity theory to significantly reduce computational overhead. By formally establishing the structural conditions under which DP decision boundaries exhibit monotone behavior, the proposed method eliminates redundant state transitions and maintains a sequence of admissible decisions through an efficient stack-based mechanism. The resulting algorithm achieves a worst-case complexity of $O(n^2)$ and can be further optimized to $O(n)$ when convex monotonicity holds, representing a substantial improvement over traditional $O(n^3)$ techniques. Experimental validation using the classical stone-merging problem demonstrates that the accelerated method executes more than 60 times faster than the standard approach for large inputs, while maintaining identical optimal results. These findings confirm that embedding geometric regularities into DP formulations not only enhances performance but also broadens the applicability of dynamic programming to computation-intensive optimization tasks. This research contributes a mathematically grounded, scalability-oriented solution for accelerating dynamic programming, with implications for logistics, production planning, data segmentation, and other large-scale decision systems.

How to cite: Kim, K. J., So, U. B., & Kim, H. C. (2025). An Accelerated Dynamic Programming Framework Based on Quadrilateral Inequality and Convex Monotonicity Theory. *TechComp Innovations: Journal of Computer Science and Technology*, 2(2), 131–138. https://doi.org/10.70063/techcompinnovations.v2i2.109

This is an open-access article distributed under the Creative Commons 4.0 Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited. ©2023 by author.

* *Corresponding author: HC.Kim@star-co.net.kp*

INTRODUCTION

Dynamic programming generally investigates how to find controls that optimize the objective for controllable objects that can be implemented differently in practical activities. Thus, dynamic programming is a mathematical approach to optimize the automation and management of the production technological process. Dynamic programming is a method that reduces the problem of extrema of several variable functions based on the principle of optimality. Dynamic programming has been widely applied in various fields, including military and economic sectors, as well as in academic fields.

Applications of dynamic programming include the management of freight cars, where decisions about when, where and how many to move have to be made in the presence of numerous sources of uncertainty, including customer demands, transit times and equipment problems. Many researchers have been working on the fault diagnosis of jet engines using dynamic programming in the military field, and various methods and techniques have been proposed for it[14][15]. The idea is to approximate dynamic programming solutions by using a function approximation structure such as neural networks to approximate the cost function[4][5]. Dynamic programming is a method that in general solves optimization problems that involve making a sequence of decisions by determining, for each decision, subproblems that can be solved in like fashion, such that an optimal solution of the original problem can be found from optimal solutions of subproblems [6-13].

Let us assume that the problem is given as follows:

Let us consider a problem where the entire interval is divided into several intervals and the sum of cost of each interval is minimized.

When the cost of the interval from i to j is called $w(i,j)$, this problem reduces to a dynamic programming problem which yields [1][2].

$$R = \sum_{i=1}^{n} f(x_i)$$

$$f(x) = min\{f(i) + w(i,x) | 1 \le i < x\} \quad (1)$$

Let the entire interval be $[1,n]$ and the solution of the problem becomes $f(n)$.

Given this, the previous algorithm performed the corresponding computations by iterating i for each n.

Since the state quantity is two, the execution complexity of this algorithm is $O(n^2)$[3].

Since the overall algorithm complexity is $O(n^2)$, we cannot implement this algorithm when n is very large.

For example, when $n = 10^5$, the total number of operations is 10^{10}, so the operation time will take about 100s.

Due to the too long execution time of the algorithm, we will discuss an accelerated method to improve the complexity of the algorithm to $O(nlogn)$.

RESEARCH METHOD

Quadrilateral inequality and Convex Monotonicity

Before discussing the acceleration method in the dynamic programming, let us discuss the quadrilateral inequality and convex monotonicity.

Definition 1:

For any binary function $w(x,y): N \times N \to R(\forall x \forall y, x \ge y => w(x,y) = 0)$, we define a one-membered function $f(x) = w(x, i+1) - w(x,i): N \to R, i \in N$.

If $f(x)$ is a non-increasing function, i.e., $w(x,i+1)-w(x,i) \geq w(x+1,i+1)-w(x+1,i)$, then $w(x,y)$ is called convex monotonicity.
If $w(x,y)$ satisfies convex monotonicity, when $\forall k>0$, then $f(x)=w(x,i+k)-w(x,i): N \rightarrow R, i \in N$ becomes a non-increasing function.
If $f(x)=w(x,i+k)-w(x,i): N \rightarrow R, i \in N$ is a non-increasing function, then $g(x)=w(i+k,x)-w(i,x): N \rightarrow R, i \in N$ is also a non-increasing function for $\forall k>0$.
Hence, if $w(x,y)$ satisfies convex monotonicity, then satisfies $w(i,j)+w(\acute{\imath},\acute{\jmath}) \leq w(i,\acute{\jmath})+w(\acute{\imath},j)$ for $\forall i,\ \acute{\imath},\ j,\ \acute{\jmath} \in N,\ i \leq \acute{\imath} \leq j \leq \acute{\jmath}$.
Definition 2:
For $\forall i,\ \acute{\imath},\ j,\ \acute{\jmath} \in N,\ i \leq \acute{\imath} \leq j \leq \acute{\jmath}$, if $w(i,j)+w(\acute{\imath},\acute{\jmath}) \leq w(i,\acute{\jmath})+w(\acute{\imath},j))$, then $w(x,y)$ is called to satisfy the quadrilateral inequality.
Similarly, we can derive the convex monotonicity of $w(x,y)$ from the quadrilateral inequality.
Theorem 1: If $w(x,y)$ satisfies convex monotonicity, then it is necessary and sufficient to satisfy the quadrilateral inequality.

Decision monotonicity and Acceleration Method

Recall the above-mentioned isomorphic state transition.

$$f(x)=min\{f(i)+w(i,x)|1 \leq i<x\} \tag{2}$$

Definition 3:
Assume $t(i,x)=f(i)+w(i,x)(1 \leq i<x)$.
For $\exists x$, $f(x)$ is called to satisfy decision monotonicity *if* $t(i,x) \geq t(j,x)(i<j)$, when for $\forall y>x$, $t(i,y) \geq t(j,y)$.
In other words, for $i,j(i<j)$, if at any time the decision i is not superior to the decision j, then the decision i cannot be superior to the decision j.
Let us discuss an accelerated method for finding $f(n)$ in $O(nlogn)$ time if the decision monotonicity is satisfied.
For each decision i from 1 to n, we find $B(i)(i<B(i) \leq n+1)$ in turn.
Here $B(i)=min\ \{x|\forall j(1 \leq j<n), j<i => t(j,x) \geq t(i,x)\}$.
That is, $B(i)$ is the minimum value of x such that decision i is superior to any decision $j(j<i)$.
For any $j(j<i)$, if $B(j) \geq B(i)$, then $f(x)$ satisfies decision monotonicity, so decision j is not needed thereafter.
Thus, the sequence of useful decisions is $B(i_1)<B(i_2)<...<B(i_t)$ at any time, when $i_1,i_2,...,i_t(i_1<i_2<\cdots<i_t)$.
When $f(x)$ is found, the decision i_k is the optimal decision if $k=max\{k|B(i_k) \leq x, k \leq l\}$ is found in the decision sequence.
That is,

$$f(x)=t(i_k,x) \tag{3}$$

Because it is necessary and sufficient that $B(i_k) \leq x$ so that $i_k(1 \leq i_k<x)$ is superior to any random decision $j(j \leq i_k)$.
From the characteristics of the decision sequence, we can then find $\acute{k}$ for $\acute{x}>x(\acute{x}=x+1)$ in the decision sequence, after the kth decision in the decision sequence, in the following.
Thus, the complexity of finding the optimal decision i_k n times becomes $O(n)$.
After finding $f(x)$, x is a useful decision, so we find $B(x)$ and sort out the decision sequence.
When $y=min\{y|t(x,y)<t(i_l,y)\}$, let's see the time to find y as T.
If $y \leq B(i_l)$, then $B(x) \leq B(i_l)$ must be true.
Thus, i_l is a useless decision and must be removed from the decision sequence (simply set $1=$

$1-1$).
This process continues until $y = min\{y|t(x,y) < t(i_l,y)\} > B(i_l)$.
Then $B(x) = y$, and x is added to the end of the decision sequence. We can see that using the stack facilitates this sequence maintenance. Each decision is made to enter the stack at most once or to exit the stack.
Since each decision takes T time when it comes out of the stack, the time complexity of maintaining the sequence is $O(nT)$.
On the other hand, the time complexity of finding the optimal decision from decision monotonicity is $O(n)$.
Thus, the total time complexity is $O(nT)$.
In any case, from the expression of the w function, we can easily find $y = min\{y|t(x,y) < t(i_l,y)\}$, and then $T = O(1)$.
In a more general case, if the w function satisfies convex monotonicity, then $t(x,y) - t(i_k,y)$ is a monotone non-increasing function with respect to y, so we can find y by a two-minute search in $O(logn)$ time.
Thus, if the w function satisfies convex monotonicity, this type of dynamic programming problems can be solved in the worst case $O(nlogn)$ time and in the best case $O(n)$ time.
As can be seen, this method is very efficient.

Quadrilateral Inequality and Decision Monotonicity

To improve the mathematical model of the type of dynamic programming that satisfies the decision monotonicity, the state transition of the corresponding mathematical model must satisfy the quadrilateral inequality.
The relationship between the introduced quadrilateral inequality and the decision monotonicity theory will be solved by developing a mathematical model for the following problem and improving the mathematical model.
There is a row of $n(n \le 1000)$ piles of stones.
Now we want to merge these piles into a single pile of stones, each time we can merge two adjacent piles into a single pile of stones, and the number of newly built stones is the cost of this merger.
When n piles are made of one pile, a mathematical model to find the minimum cost can be established as follows:
[Algorithm Analysis] This problem is a typical application of dynamic programming.
Let n stone piles' numbers be 1, 2, ... , n in turn, and let the number of stones of each pile be $d[1], d[2], \dots, d[n]$. Then, $m[i,j], 1 \le i \le j \le n$ are the minimum cost of merging $d[i], \dots, d[j]$, and the state-transfer equation and edge conditions of the dynamic scheme are as follows :

$$\begin{cases} m[i,j] = 0, i = j \\ m[i,j] = min\{m[i,k-1] + m[k,j] + \sum_{l=i}^{j} d[l]\}, i < j \end{cases} \quad (4)$$

In addition, by convention $s[i,j] = k$, we can find the minimum value by plotting the partition position before the merging of $d[i], \dots, d[j]$ and then find the merging method.
In the above expression, the value of $\sum_{l=i}^{j} d[l]$ can be calculated in $O(1)$ time as $t[i] = \sum_{j=1}^{i} d[j], i = 1 \dots n, t[0] = 0$, first calculated during the preprocessing process $\sum_{l=i}^{j} d[l] = t[j] - t[i-1]$.
In the above algorithm, the total number of states is $O(n^2)$, each state is derived from $O(n)$ states, and since the state transition time is $O(1)$, the total time complexity is $O(n^3)$.
[Improvement of time complexity]

In this problem, $w[i,j]$ satisfies the quadrilateral inequality when $w[i,j] = \sum_{l=i}^{j} d[l]$.
Also, since $d[i] \geq 0, t[i] \geq 0$, we see that $w[i,j]$ satisfies monotonicity with respect to interval membership.
From the definition of $w[i,j]$, the above state transfer can be written as

$$\begin{cases} m[i,j] = 0, i = j \\ m[i,j] = min\{m[i,k-1] + m[k,j] + w[i,j]\}, i < j \end{cases} \quad (5)$$

It can be derived that $m[i,j]$ also satisfies the quadrilateral inequality when $w[i,j]$ satisfies the above two conditions.
That is, for $\forall i,\ \acute{\imath},\ j,\ \acute{\jmath} \in N,\ i \leq \acute{\imath} \leq j \leq \acute{\jmath},\ m[i,j] + m[\acute{\imath},\acute{\jmath}] \leq m[i,\acute{\jmath}] + m[\acute{\imath},j]$.
And this property can be proved by mathematical induction.
In the quadrilateral inequality we have a recursion for length $l = j - i = \acute{\jmath} - \acute{\imath}$.
For $i = \acute{\imath}$ or $j = \acute{\jmath}$, the inequality must be true.
Hence, when $l \leq 1$, the function $m[i,j]$ satisfies the quadrilateral inequality.
We proceed by splitting the proof into two cases.
In case $i < \acute{\imath} = j < \acute{\jmath}$
In this case, the quadrilateral inequality is simplified to the semi-triangular inequality as follows:

$$m[i,j] + m[j,\acute{\jmath}] \leq m[i,\acute{\jmath}] \quad (6)$$

Putting $k = max\{p | m[i,\acute{\jmath}] = m[i,p-1] + m[p,\acute{\jmath}] + w[i,\acute{\jmath}]$, k is either $k \leq j$ or $k > j$.
In the following, we consider only if $k \leq j$.
The proof is similar for $k > j$.
When $k \leq j$

$$\begin{aligned} m[i,j] + m[j,\acute{\jmath}] &\leq w[i,j] + m[i,k-1] + m[k,j] + m[j,\acute{\jmath}] \\ &\leq w[i,\acute{\jmath}] + m[i,k-1] + m[k,j] + m[j,\acute{\jmath}] \\ &\leq w[i,\acute{\jmath}] + m[i,k-1] + m[k,\acute{\jmath}] = m[i,\acute{\jmath}] \end{aligned} \quad (7)$$

In case $i < \acute{\imath} < j < \acute{\jmath}$

$$\begin{aligned} y &= max\{p | m[\acute{\imath},j] = m[\acute{\imath},p-1] + m[p,j] + w[\acute{\imath},j]\} \\ z &= max\{p | m[i,\acute{\jmath}] = m[i,p-1] + m[p,\acute{\jmath}] + w[i,\acute{\jmath}] \end{aligned} \quad (8)$$

Here, we also divide into two cases: the case $z \leq y$ or $z > y$.
In the following, we consider only the case of $z \leq y$.
Since $i < z \leq y \leq j$

$$\begin{aligned} m[i,j] + m[\acute{\imath},\acute{\jmath}] &\leq w[i,j] + m[i,z-1] + m[z,j] + w[\acute{\imath},\acute{\jmath}] + m[\acute{\imath},\acute{\jmath}] + m[\acute{\imath},y-1] + m[y,\acute{\jmath}] \\ &\leq w[i,\acute{\jmath}] + w[\acute{\imath},j] + m[\acute{\imath},y-1] + m[i,z-1] + m[z,j] + m[y,\acute{\jmath}] \\ &\leq w[i,\acute{\jmath}] + w[\acute{\imath},j] + m[\acute{\imath},y-1] + m[i,z-1] + m[y,j] + m[z,\acute{\jmath}] \\ &\leq m[i,\acute{\jmath}] + m[\acute{\imath},j] \end{aligned} \quad (9)$$

Coordinating 2 cased above, we see that $m[i,j]$ satisfies the quadrilateral inequality.
Putting $s[i,j] = max\{k | m[i,j] = m[i,k-1] + m[k,j] + w[i,j]\}$, we can derive the monotonicity of the function $s[i,j]$ from the fact that the function $m[i,j]$ satisfies the quadrilateral inequality.
To prove $s[i,j] \leq s[I,j+1]$, we first set $m_k[i,j] = m[i,k-1] + m[k,j] + w[i,j]$, and then we prove that $\forall i < k < \acute{k} < j, m_{\acute{k}}[i,j] \leq m_k[i,j] => m_{\acute{k}}[i,j+1] \leq m_k[i,j+1]$ from the definition of $s[i,j]$.
In fact, we can prove the inequality $m_k[i,j] - m_{\acute{k}}[i,j] \leq m_k[i,j+1] - m_{\acute{k}}[i,j+1]$ which embeds this expression.
Thus, by expanding the expression from the definition $m_k[i,j] + m_{\acute{k}}[i,j+1] \leq m_k[i,j+1] + m_{\acute{k}}[i,j]$, it follows that $[k,j] + m\left[\acute{k},j+1\right] \leq m\left[\acute{k},j\right] + m[k,j+1]$, and it is a quadrilateral inequality for the case $k < \acute{k} \leq j < j+1$.
The proof of $s[i,j+1] \leq s[i+1,j+1]$ can also be proved in a similar way.

Summarizing the above description, we see that $s[i,j]$ is monotonic when $w[i,j]$ satisfies the quadrilateral inequality.

The improved state transfer equation obtained from this is

$$\begin{cases} m[i,j]=0, i=j \\ m[i,j]=min\{m[i,k-1]+m[k,j]+w[i,j]\}, i<j, s[i,j-1] \\ <k\le s[i+1,j] \end{cases} \quad (10)$$

The time complexity of the algorithm can be improved in a similar way in the problem of finding the maximum cost of merging stone piles.

Let us analyze the time complexity of the algorithm.

For certain states of length $l=j-i$, $m[1,l+1]$ is $s[1,l]$ to $s[2,l+1]$, $m[2,l+2]$ is $s[2,l+1]$ to $s[3,l+2]$.

…

Thus, if the transition times of these states are summed together, then $s[n-l+1,\ n]-s[1,\ l]+n-l+1=O(n)$.

Since $0\le l\le n-1$, the total time complexity is $O(n^2)$.

RESULT AND DISCUSSION

To see the superiority of the previous method of acceleration in solving the above dynamics, let us compare the running time of the algorithm for different n values (Table 1).

Table 1. Simulation analysis result

n	Existing Method(s)	Acceleration method (s)
100	<0.01	<0.01
1000	10	<0.01
10000	>1min	1

As shown in the table above, the execution times are similar for small n values, but the differences in execution times are significant with increasing n values.

CONCLUSION

This study presents an accelerated method for solving dynamic programming problems by leveraging decision monotonicity, convex monotonicity, and quadrilateral inequality. By exploiting these structural properties, the proposed method significantly reduces the computational complexity of classical DP algorithms, making it suitable for large-scale practical applications. Experimental comparisons confirm dramatic improvements in execution time, particularly when the input size becomes large. The research contributes both theoretical foundations and practical algorithmic advances, demonstrating that many DP problems can be solved far more efficiently when their cost structures satisfy specific convexity or geometric properties.

LIMITATIONS AND FUTURE RESEARCH RECOMMENDATIONS

Although the proposed method is highly effective, several limitations should be noted. First, the acceleration technique applies only when the DP transition function satisfies convex monotonicity or quadrilateral inequality; thus, the method is not universally applicable to all DP formulations. The theoretical properties must be verified beforehand, which may require complex analytical work for certain models. Second, experimental validation is limited to simulated environments using the stone merging problem. Real-world systems with noisy

data, dynamic parameters, or nonlinear costs may not strictly satisfy the monotonicity conditions. Third, the study does not explore performance under multi-dimensional or stochastic DP models, where state transition structures may behave differently and could violate the required geometric assumptions.

Future studies should extend the proposed method to broader classes of dynamic programming, including stochastic DP, multi-agent decision models, and high-dimensional state spaces. Investigating automatic detection or learning-based approximation of convex monotonicity may enable greater applicability without explicit mathematical proofs. Additionally, integrating the acceleration method with machine learning techniques—such as neural DP approximators or reinforcement learning—could merge theoretical guarantees with modern data-driven optimization. Lastly, real-world validation in areas such as supply chain optimization, robotics control, and sequence-based cost minimization would further demonstrate the robustness and scalability of the proposed framework.

ACKNOWLEDGEMENT

We would like to thank Pak Ju Song, Wang Chol Jin and Sin Ju Hyok for their contributions to the study. This study was supported by Kim Chaek University of Technology.

REFERENCES

Kalaba, R. E., Natsuyama, H. H., & Ueno, S. (1999). The estimation of parameters in time-dependent transport problems: Dynamic programming and associative memories. *Computers & Mathematics with Applications*, 37(2), 41-45.

Lew, A. (1985). *Computer Science: A Mathematical Introduction*. Prentice-Hall, Inc.

Lew, A. (2006). Canonical greedy algorithms and dynamic programming. *Control and Cybernetics*, 35(3), 621-643.

Lew, A. (2000). N degrees of separation: Influences of dynamic programming on computer science. *Journal of mathematical analysis and applications*, 249(1), 232-242.

Lew, A. (2002, July). A Petri net model for discrete dynamic programming. In *International Workshop on Uncertain Systems and Soft Computing, Beijing*.

Lew, A., & Halverson, R. (1995, April). A FCCM for dataflow (spreadsheet) programs. In *Proceedings IEEE Symposium on FPGAs for Custom Computing Machines* (pp. 2-10). IEEE.

Lew, A & Holger Mauch. Bellman nets: A Petri net model and tool for dynamic programming. In Le Thi Hoai An and Pham Dinh Tao, editors, Proceedings of Modelling, Computation and Optimization in *Information Systems and Management Sciences* (MCO), Metz, France

Lew, A., & Mauch, H. (2003, July). Solving integer dynamic programming using Petri nets. In *Proceedings of the Multiconference on Computational Engineering in Systems Applications* (CESA), Lille, France, July 9–11.

Lew, A., & Mauch, H. (2006). *Dynamic programming: A computational tool* (Vol. 38). Springer.

Powell, W. B. (2007). *Approximate Dynamic Programming: Solving the curses of dimensionality* (Vol. 703). John Wiley & Sons.

Sniedovich, M. (2010). *Dynamic programming: foundations and principles*. CRC press.

Yao, F. F. (1982). Speed-up in dynamic programming. *SIAM Journal on Algebraic Discrete Methods*, 3(4), 532-540.

Yebo L., et al., Research on gas fault fusion diagnosis of aero-engine component, *Acta Aeronaut. Astronaut*

Zhang, H., Liu, D., Luo, Y., & Wang, D. (2012). *Adaptive dynamic programming for control: algorithms and stability*. Springer Science & Business Media.

Zhang, P. (2008). *Aeroengine fault diagnostics based on kalman filter*. Nanjing university of aeronautics and astronautics.