Extended Essay

Word count: 4000

Table of contents

1 Introduction

The topic of dynamic programming has always been of great significance to me, as I am highly interested in the field of programming; I aspire to become a computer scientist one day. Over the past few years, dynamic programming has found its way into multiple fields, including exploitation of natural resources (Stensland and Tjostheim 99), management of finance (Elton and Gruber 473) and, in recent years, modern technology, such as path finding algorithms in GPSs (Spouge 1552). As technology advances, dynamic programming is expected to grow more and more a viable option for effectively optimising programmatic procedures, allowing for faster computation of information.

This paper will focus on determining whether dynamic programming can be used as an effective method of logical problem solving. In their journal article, "The Computational Implications of Variations in State Variable/State Ratios in Dynamic Programming and Total Enumeration", Samuel G. Davis and Edward T. Reutzel describes the nature of dynamic programming as follows:

Dynamic programming computational efficiency rests upon the so-called principle of optimality, where it is possible to decompose combinatorial problems into individual sub-problems (stages). The sub-problems are solved independently and linked together through the use of state variables which reflect optimal decisions for other (preceding) stages. (1003)

Simply put, dynamic programming is a mathematical method in which past answers are incorporated in solving future problems. A fundamental example of dynamic programming includes the recursive function for determining the factorial of a non-negative integer, which can be denoted as follows:

$$f(n) = \begin{cases} n \times f(n-1) & (n \geq 1) \\ 1 & (n = 0) \end{cases}$$

provided n is a non-negative integer.

Thus, to test the potential of dynamic programming, this paper will investigate whether it would be possible to answer the following classic mathematical problem, henceforth research question, through the use of dynamic programming:

"What is the total number of ways in which $1 \times \alpha$ tiles can fit in a $\alpha \times \beta$ board?"

2 Approaching the problem

In order to solve this problem, I will be formulating a mathematical function and writing code in C. As a basic demonstration, the aforementioned function for calculating the factorial of a non-negative integer will be converted into code written in C; henceforth, numbers in bold in the output section of a code will refer to values that have been input into the code:

Code:

```
#include <stdio.h>

int Factorial(int num) {
	if (num == 0)
		return 1;


	return num * Factorial(num - 1);
}

int main(void) {
	int num;

	scanf("%d", &num);
	printf("%d \n", Factorial(num));

	return 0;
}
```

Output:

```
6
120
```

When input a value of 6, the function recalls itself with an input value of 5, again with an input value of 4, and so on, until it reaches an input value of 0 and returns the collective product of the preceding inputs.

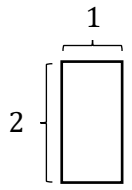
Thus, one may safely assume that the final function derived from the research question may also make use of a recursive function which repeats itself until termination.

3 Tile of dimension 1×2 and board of dimension $2\times\beta$

In order to find a solution and derive a recursive function for the problem, the simplest case of the problem, "What is the total number of ways in which 1×2 tiles can fit in a $2\times\beta$ board?", will first be analysed. Cases of 1×1 tiles and a $1\times\beta$ board will not be analysed as it is obvious there exists only one arrangement for each case, making it trivial to conclude so. In order to represent that β is a natural number, the set, $\mathbb{N}$, will be introduced, where $\mathbb{N}=\{1, 2, 3, 4, \cdots\}$. Henceforth, the set, $\mathbb{N}$, will be used to denote the set of natural numbers.

3.1 Tile of dimension 1×2 and board of dimension 2×1

As $\beta\in\mathbb{N}$, the initial step in approaching the problem would be to examine the total number of ways in which 1×2 tiles can fit in a 2×1 board, as the smallest value in the set, $\mathbb{N}$, is 1. Without a doubt, even those with a rudimentary understanding of geometry would realise that there exists only one way in which 1×2 tiles can fit in a 2×1 board:



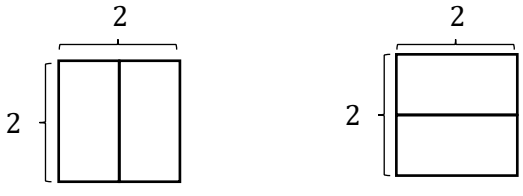
a)

Figure 1: Possible arrangements of 1×2 tiles in a 2×1 board

Although not too interesting at first glance, as the horizontal length of the board, β, increases, the total number of possible positionings begin to present somewhat of an interesting pattern.

3.2 Tile of dimension 1×2 and board of dimension 2×2

In order to determine what this pattern exactly is, the possible posistionings of the next simplest case of the problem will be computed, with tiles of dimensions 1×2 and board of dimensions 2×2:



a) b)

Figure 2: Possible arrangements of 1×2 tiles in a 2×2 board

By placing one of the 1×2 tiles into a horizontal position, it naturally forces the following tile to also be placed horizontally inside the board, allowing for a total of two possible positionings. This structure of horizontal and vertical tiles plays a crucial role in determining the overall number of tile arrangements as in some cases, both vertical and horizontal tiles may constitute an arrangement of tiles within a board.

3.3 Tile of dimension 1×2 and board of dimension 2×3

Namely, the next simplest example, which in this case is with tiles of dimensions 1×2 and board of dimensions 2×3, effectively demonstrates how both vertical and horizontal tiles may coexist within a tile arrangement:

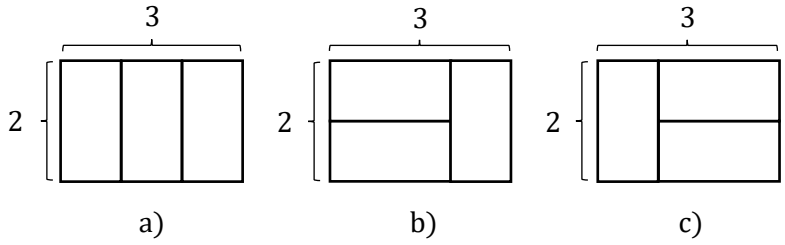


Figure 3: Possible arrangements of 1×2 tiles in a 2×3 board

Applying these patterns of placing a tile vertically or horizontally to decrease the horizontal length of the board allows one to obtain a recursive formula that returns the total number of tile arrangements.

3.4 Tile of dimension 1×2 and board of dimension $2\times n$

As mentioned above, placing the first tile vertically in the board decreases the horizontal length of the total available area for tile placement by 1 unit. Henceforth, shaded regions in a diagram will represent unoccupied areas within a board:

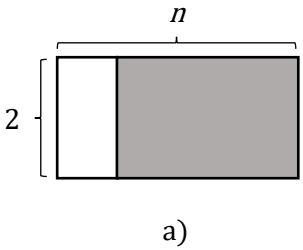
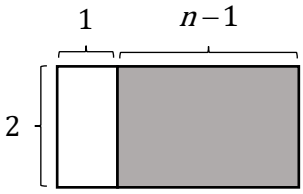


Figure 4: Arrangement of one 1×2 tile in a $2\times n$ board

As the 1×2 tile occupies a certain area of the board, the problem is now reduced to

finding the number of possible arrangements within a board with width, $(n-1)$:

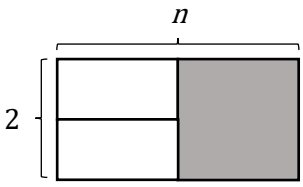


a)

Figure 5: Arrangement of one 1×2 tile in a $2\times n$ board with distinction between tile and remaining space

From this point on, one may repeat the process to reduce the problem until there remains no additional space for further tile placements.

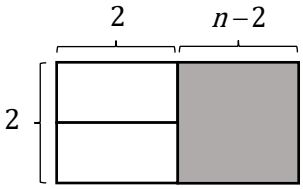
The problem may also be reduced by inserting the first tile horizontally. As the first tile is inserted horizontally into the board, the following tile is forced horizontally either beneath or above the preceding horizontal tile, depending on the position of the first horizontal tile:



b)

Figure 6: Arrangement of two 1×2 tiles in a $2\times n$ board

Similar to the case of vertical tiles, the problem is now reduced to finding the number of possible arrangements within a board of width, $(n-2)$:



b)

Figure 7: Arrangement of two 1×2 tiles in a $2\times n$ board with distinction between tiles and remaining space

One again, the reduced problem may be solved over and over again until there remains no space for additional tiles.

3.5 Derivation of the recursive formula

Analysis of the pattern above suggests that the total number of possible tile arrangements arrangements with 1×2 tiles and a $2\times\beta$ board is equal to the sum of the total number of possible tile arrangements with a $2\times(\beta-1)$ board and a $2\times(\beta-2)$ board. Therefore, the recursive function for calculating the total number of possible tile arrangements with 1×2 tiles and a $2\times\beta$ board can be computed as follows:

$$f(\beta)=f(\beta-1)+f(\beta-2)$$

provided $\beta\in\mathbb{N}$.

However, in certain circumstances, the recursive formula above does not successfully compute unless some of its beginning values are assigned a certain output. For example, when input a value of 3, the recursive function, $f(3)=f(2)+f(1)$, cannot be calculated unless there exist constant outputs for both $f(2)$ and $f(1)$. These output values are otherwise known as initial conditions, which are values required when starting a dynamic system.

In this case, the output values for the initial conditions have already been calculated from Figure 1 and Figure 2, prior to deducing the recursive function; $f(1)=1$ and $f(2)=2$. Thus, the fuction may be newly denoted as follows:

$$f(\beta)=\begin{cases}1 & (\beta=1)\\ 2 & (\beta=2)\\ f(\beta-1)+f(\beta-2) & (\beta\geq3)\end{cases}$$

provided $\beta\in\mathbb{N}$.

The recursive formula above presents some interesting characteristics that become more apparent once listed:

β	1	2	3	4	5	6	7	8	9	10
$f(\beta)$	1	2	3	5	8	13	21	34	55	89

Figure 8: First ten outputs of the recursive formula derived from the case with 1×2 tiles and $2\times\beta$ board

Some avid enthusiasts of mathematics may instantly realise that this pattern almost perfectly resembles the classical Fibonacci sequence, with the only difference being the omission of the initial term, 1:

$$1, 1, 2, 3, \cdots, \quad F_n = F_{n-1} + F_{n-2}, \quad \cdots$$

3.6 Dynamic programme of the recursive formula

Thus, the aforementioned function may now be compiled into code for easier computation of each different case of the simplifed dynamic programming question, "What is the total number of ways in which 1×2 tiles can fit in a $2\times\beta$ board?". The code below includes some extra lines of text for added user-friendliness.

Code:

```
#include <stdio.h>

int totalWays(int beta)
{
	if (beta == 1)
		return 1;
	else if (beta == 2)
		return 2;
	else
		return totalWays(beta - 1) + totalWays(beta - 2);
}

int main(void)
{
	int beta;
	printf("The total number of ways in which 1*2 tiles can fit in a 2*β
board. \n");
	printf("Input variable β: ");
	scanf("%d", &beta);

	printf("Total number of ways: %d \n", totalWays(beta));
	return 0;
}
```

Output:

```
The total number of ways in which 1*2 tiles can fit in a 2*β field.
Input variable β: 8
Total number of ways: 34
```

3.7 Summary

Ultimately, one may deduce that the total number of ways in which 1×2 tiles can fit in a $2\times\beta$ board resembles a recursive pattern in which the output is equal to the collective sum of the outputs involving cases with a $2\times(\beta-1)$ board and a $2\times(\beta-2)$ board. One may also infer from this simplified scenario that the solution to the main question, "What is the total number of ways in which $1\times\alpha$ tiles can fit in a $\alpha\times\beta$ board?", will also most definitely make use of recursion.

4 Tile of dimension $1\times\alpha$ and board of dimension $\alpha\times\beta$

Now, back to the main topic of discussion, calculating the the total number of ways in which $1\times\alpha$ tiles can fit in a $\alpha\times\beta$ board. Since it would virtually be impossible to derive individual sets of functions for each different scenario, the three main conditions in which the total number of tile arrangements depends on will be analysed to compute a collective function. Such three conditions are:

1. $\alpha > \beta$
2. $\alpha = \beta$
3. $\alpha < \beta$

provided $\alpha, \beta \in \mathbb{N}$

4.1 Total number of tile arrangements where $\alpha > \beta$

Proposition:

Provided $\alpha, \beta \in \mathbb{N}$, when the length of the tile, α, is greater than the horizontal length of the board, β, the total number of tile arrangements is equal for all cases: one.

Proof:

A visual rendition of a case involving tiles of dimensions $1\times n$ and board of dimensions $n\times(n-m)$, where $n, m \in \mathbb{N}$ and $n > m$, demonstrates how there exists only one possible tile arrangement for all cases where $\alpha > \beta$:

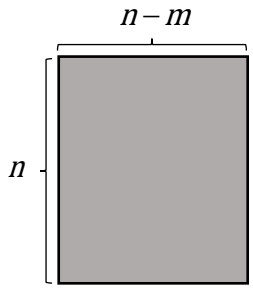


Figure 9: Empty $n\times(n-m)$ board

Since $\alpha = n$, the length of the tile itself should also be n. Considering that the length of the tile, n, is greater than the horizontal length of the board, $n-m$, and that the tiles can only be placed in rectilinear orientations, it can be deduced that all tiles must be and can only be placed vertically in the board:

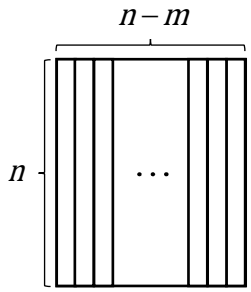


Figure 10: Total permutations of case involving a $n \times (n-m)$ board

Thus, it can be deduced that there is one possible tile arrangement in cases involving $1 \times \alpha$ tiles and a $\alpha \times \beta$ board, where $\alpha > \beta$.

4.2 Function and code for tile arrangements where $\alpha > \beta$

The function for the total number of tile arrangements where $\alpha > \beta$ can, therefore, be readily computed as follows:

$$f(\alpha, \beta) = 1$$

provided $\alpha, \beta \in \mathbb{N}$ and $\alpha > \beta$.

The portion of the code responsible for computing the output of this function may be represented as follows where the function returns the value, 1, if $\alpha > \beta$. Since the Include syntax

```
#include <stdio.h>
```

and the main function

```
int main(void) {
        ....
        return 0;
}
```

remain constant regardless of the recursive function, the two blocks of code will henceforth be omitted when representing derived functions in code format. Therefore, the representation of the code is as follows:

```
int totalWays(int alpha, int beta) {
	if (alpha > beta)
		return 1;
}
```

Thus, when α, the length of the tile and vertical length of the board is greater than β, the horizontal length of the board, the total number of tile arrangements will always be one.

4.3 Total number of tile arrangements where $\alpha = \beta$

Proposition:

Provided $\alpha, \beta \in \mathbb{N}$, when the length of the tile, α, is equal to the horizontal length of the board, β, the total number of tile arrangements is equal for all cases: two.

Proof:

A visual representation of a case with tiles of dimensions $1 \times n$ and board of dimensions $n \times n$, where $n \in \mathbb{N}$, demonstrates how there exists only two possible tile arrangements for all cases, where α is equal to β:

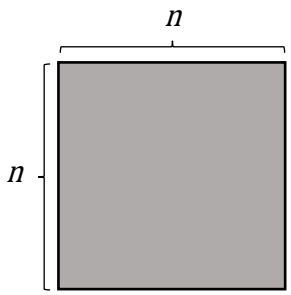


Figure 11: Empty $n \times n$ board

Placing a tile vertically into the board would force the following tile to also be placed vertically, as the horizontal length would decrease by 1 unit, making it impossible for the second tile to be placed horizontally. This pattern also applies to when the first tile is placed horizontally into the board; the following tiles will be forced in horizontally. This exact concept has been explored above with cases involving $1 \times n$ tiles and a $n \times (n-1)$ board:

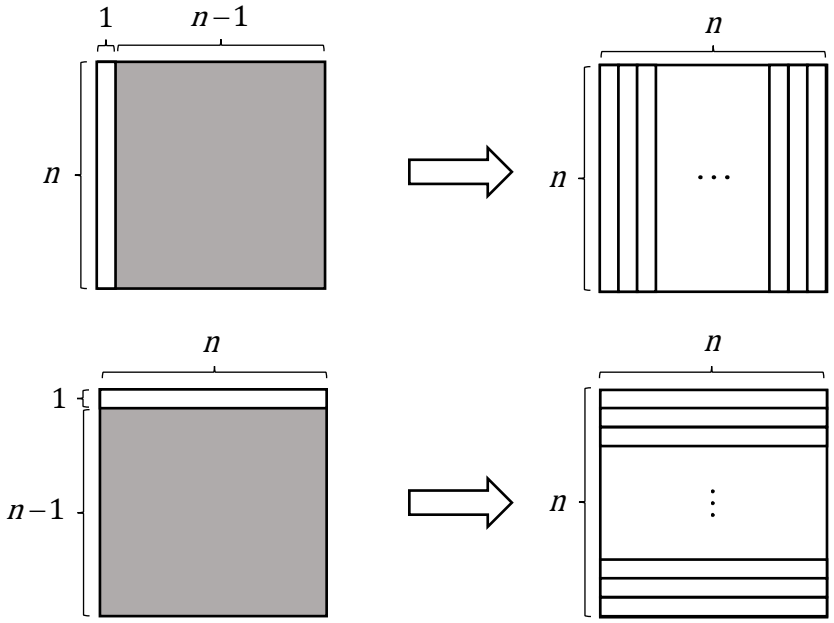


Figure 12: Total permutations of case involving a $n \times n$ board

With cases where $\alpha = \beta$, the tile arrangements can either be initiated by placing a tile vertically or horizontally, leading to a total of two possible permutations; the remaining space would be filled with tiles in the same orientation as the initial tile. Thus, it can be deduced that there exist two possible tile arrangements in cases involving $1 \times \alpha$ tiles and a $\alpha \times \beta$ board, where $\alpha = \beta$.

4.4 Function and code for tile arrangements where $\alpha = \beta$

Thr function for tile arrangements where $\alpha = \beta$ can be readily computed as follows:

$$f(\alpha, \beta) = 2$$

provided $\alpha, \beta \in \mathbb{N}$ and $\alpha = \beta$.

The portion of the code responsible for computing the output of this function may be represented as follows, where the function returns the value, 2, if $\alpha = \beta$:

```
int totalWays(int alpha, int beta) {
	if (alpha == beta)
		return 2;
}
```

Thus, when α, the length of the tile and vertical length of the board is equal to β, the horizontal length of the board, the total number of tile arrangements will always be two.

4.5 Total number of tile arrangements where $\alpha < \beta$

Proposition:

Provided $\alpha, \beta \in \mathbb{N}$, when the length of the tile, α, is less than the horizontal length of the board, β, the total number of tile arrangements will vary for each case in a recursive manner, very much like the Fibonacci sequence.

Proof:

Figure 3 suggests that in cases with 1×2 tiles and a 2×3 board, both horizontal and vertical tiles may coexist within the board. This is due to the nature of the board as it is guaranteed that there will remain space for additional tiles. The number of tile placements varies depending on the length of the tile (the vertical length of the board) and the horizontal length of the board. This idea can be demonstrated through a visual representation of two possible initial permutations in cases involving tiles of dimensions $1 \times \alpha$ and board of dimensions $\alpha \times \beta$, where $\alpha = n$ and $\beta = n + m$, hence, where $\alpha < \beta$:

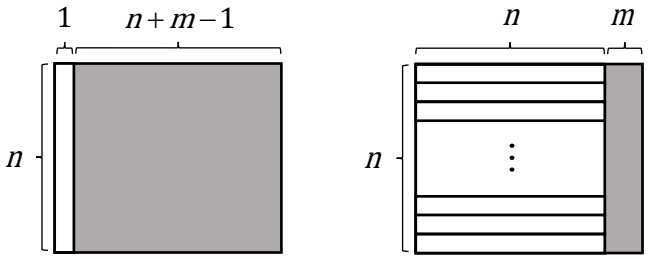


Figure 13: Total initial permutations of case involving a $n \times (n + m)$ board

After the first tile is inserted, additional tiles may either be placed in three different types of arrangements

1. only vertically
2. only vertically or only horizontally
3. both vertically and horizontally

depending on the relationship between the length of the tile, *n*, and the horizontal length of unoccupied space within the board, *m*, with the conditions being

1. $n > m$
2. $n = m$
3. $n < m$

respectively.

This finding suggests that whilst the number of tile arrangements resulting from conditions 1 and 2 outputs the values 1 and 2 respectively, the number of tile arrangements resulting from condition 3 would solely depend on the dimension of the unoccupied area of the board after a tile has been inserted. This concept can be applied to derive the final recursive function for the case of $1\times\alpha$ tiles and a $\alpha\times\beta$ board where $\alpha<\beta$.

4.6 Function and code for tile arrangements where $\alpha<\beta$

Figure 13 suggests that when computing the total number of ways in which $1\times\alpha$ tiles can fit in a $\alpha\times\beta$ board, where $\alpha<\beta$, only two different states need to be considered: $\alpha\times\beta$ board with remaining space of dimensions $\alpha\times(\beta-1)$ and $\alpha\times\beta$ board with remaining space of dimensions $\alpha\times(\beta-\alpha)$. As additional tiles can only be placed in empty areas, the recursive function can be defined by referring to the unoccupied space within the board. Thus, the recursive function for calculating the total number of possible tile arrangements with $1\times\alpha$ tiles and a $\alpha\times\beta$ board, where $\alpha<\beta$, can be computed as follows:

$$f(\alpha,\beta)=f(\beta-1)+f(\beta-\alpha)$$

provided $\alpha,\beta\in\mathbb{N}$ and $\alpha<\beta$.

The portion of the code responsible for computing the output of this function may be represented as follows where the function recalls itself until terminating condtions, if $\alpha<\beta$:

```
int totalWays(int alpha, int beta) {
	if (alpha < beta)
		return totalWays(beta - 1, alpha) + totalWays(beta - alpha,
alpha);
}
```

Thus, when α, the length of the tile and vertical length of the board is less than β, the horizontal length of the board, the total number of tile arrangements will vary depending on the values of α and β.

4.7 Final function and code for the research question

The final function for the dynamic programming question can be deduced as the following:

$$f(\alpha,\beta)=\begin{cases}1 & (\alpha>\beta)\\ 2 & (\alpha=\beta)\\ f(\beta-1)+f(\beta-\alpha) & (\alpha<\beta)\end{cases}$$

provided $\beta\in\mathbb{N}$.

Other than for cases where α, the length of the tile and vertical length of the board is greater than of equal to β, the horizontal length of the tile, the function will behave recursively until $\alpha\geq\beta$ and returns a constant value of either 1 or 2.

Piecing together the final three derived blocks of code

```
int totalWays(int alpha, int beta) {
	if (alpha > beta)
		return 1;
}

int totalWays(int alpha, int beta) {
	if (alpha = beta)
		return 2;
}

int totalWays(int alpha, int beta) {
	if (alpha < beta)
		return totalWays(beta - 1, alpha) + totalWays(beta - alpha,
alpha);
}
```

from three different conditions

1. $n>m$
2. $n=m$
3. $n<m$

respectively, the finished code can be represented as follows. Similar to the code from **3.6**, the code below includes some extra lines of text for added user-friendliness:

Code:

```
#include <stdio.h>

int totalWays(int beta, int alpha)
{
	if (alpha > beta)
		return 1;
	else if (alpha == beta)
		return 2;
	else
		return totalWays(beta - 1, alpha) + totalWays(beta - alpha,
alpha);
}

int main(void)
{
	int alpha, beta;
	printf("The total number of ways in which 1*α tiles can fit in a α*β
board. \n");
	printf("Input variable α: ");
	scanf("%d", &alpha);
	printf("Input variable β: ");
	scanf("%d", &beta);

	printf("Total number of ways: %d \n", totalWays(beta, alpha));
	return 0;
}
```

Output:

```
The total number of ways in which 1*α tiles can fit in a α*β board.
Input variable α: 5
Input variable β: 44
Total number of ways: 119305
```

Comparing with actual values were obtained from The On-Line Encyclopedia of Integer Sequences, , it can be confirmed that the function works as intended:

α	β	Actual value	$f(\alpha, \beta)$
2	24	75025	75025
2	25	121393	121393
2	26	196418	196418
2	27	317811	317811
2	28	514229	514229
2	29	832040	832040
3	30	58425	58425
4	35	43371	43371
5	40	38740	38740
6	45	37975	37975
7	50	39173	39173
8	55	41623	41623

(OEIS)

Figure 12: Total permutations of case involving a $n \times n$ board

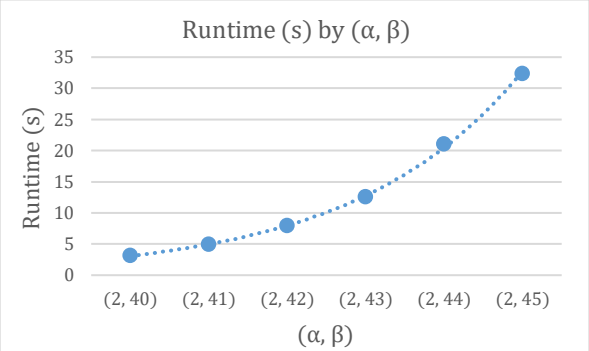
In conclusion, it can be deduced that the dynamic programming question, “What is the total number of ways in which $1 \times \alpha$ tiles can fit in a $\alpha \times \beta$ board?”, can be solved through the use of recursive functions, making it possible to recreate the problem using code for easier, more efficient computation of the inputs, α and β.

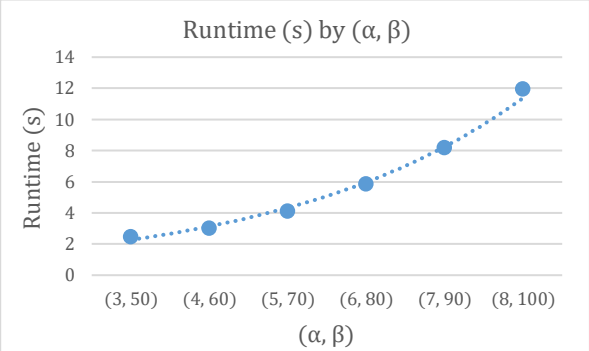
5 Extension of the research question

Although seemingly fast and efficient, dynamic programmes do have one limitation, being that every input, other than the initial conditions, can only be computed once its sub-inputs have been computed. This means that as the input increases in value, the number of computations subsequently increase, leading to inefficient, time-consuming calculations . This problem can in fact be observed from the derived function above.

The addition of the "time.h" library allows the code to print the runtime of each trial after execution:

α	β	Runtime (s)
2	40	3.114000
2	41	4.976000
2	42	7.928000
2	43	12.561000
2	44	21.062000
2	45	32.354000
3	50	2.452000
4	60	2.997000
5	70	4.112000
6	80	5.854000
7	90	8.181000
8	100	11.931000





As the values of α and β increase, the runtime of each trial consequently increases exponentially. In fact, the device I used to measure the runtime of this function could not at all compute any greater values; hence, could not list further values. Therefore, In order to solve this issue of long computation times, I aim to devise an explicit formula for estimating the result for the research question, which is built upon Binet's formula and the Golden Ratio.

5.1 Derivation of the explicit formula for the Fibonacci sequence

As mentioned in **3.5**, the Fibonacci sequence is as follows, where a certain term is the sum of its two preceding terms:

$$1, 1, 2, 3, \ \cdots, \ \ F_n = F_{n-1} + F_{n-2}, \ \ \cdots$$

This nature of the Fibonacci sequence makes it almost impossible to efficiently calculate large inputs through recursion, as computing an output for a certain input requires the outputs of all of its preceding inputs. There exists, however, an explicit formula that allows the computation of the n^{th} term of the Fibonacci sequence: Binet's formula. In my case, I have decided to approach Binet's formula through limits and linear recurrence equations.

As terms in the Fibonacci sequence grow larger, the ratio between two consecutive Fibonacci terms approach a certain value, otherwise known as the Golden Ratio (Heyde, 1079). Henceforth, this value – the golden ratio – will be denoted as "Φ".

$$\lim_{n\to\infty} \frac{F_n}{F_{n-1}} = \Phi$$

Substituting n with $n-1$ returns the following limit:

$$\lim_{n\to\infty} \frac{F_{n-1}}{F_{n-2}} = \Phi$$

As the ratios between F_n and F_{n-1}, F_{n-1} and F_{n-2} both tend towards the same irrational constant, Φ, an equation describing the equality between the ratios of consecutive Fibonacci terms can be defined as follows:

$$\lim_{n\to\infty} \frac{F_n}{F_{n-1}} = \lim_{n\to\infty} \frac{F_{n-1}}{F_{n-2}} = \Phi$$

Consider the recursive formula of the Fibonacci sequence:

$$F_n = F_{n-1} + F_{n-2}$$

Hence, F_n from $\lim_{n\to\infty}\frac{F_n}{F_{n-1}}=\lim_{n\to\infty}\frac{F_{n-1}}{F_{n-2}}=\Phi$ can be substituted with $F_{n-1}+F_{n-2}$ to return:

$$\lim_{n\to\infty}\frac{F_{n-1}+F_{n-2}}{F_{n-1}}=\lim_{n\to\infty}\frac{F_{n-1}}{F_{n-2}}=\Phi$$
$$1+\lim_{n\to\infty}\frac{F_{n-2}}{F_{n-1}}=\lim_{n\to\infty}\frac{F_{n-1}}{F_{n-2}}=\Phi$$

The equation above shows that $\lim_{n\to\infty}\frac{F_{n-1}}{F_{n-2}}=\Phi$, suggesting that $\lim_{n\to\infty}\frac{F_{n-2}}{F_{n-1}}=\frac{1}{\Phi}$, therefore:

$$1+\frac{1}{\Phi}=\Phi$$
$$\Phi+1=\Phi^2$$
$$\Phi^2-\Phi-1=0$$

The quadratic formula, $\frac{-b\pm\sqrt{b^2-4ac}}{2a}$, can be used to determine Φ:

$$\frac{-b\pm\sqrt{b^2-4ac}}{2a}=\frac{-(-1)\pm\sqrt{(-1)^2-\{4\cdot1\cdot(-1)\}}}{2\cdot1}$$
$$=\frac{1\pm\sqrt{5}}{2}$$
$$=\frac{1+\sqrt{5}}{2}\quad\text{or}\quad\frac{1-\sqrt{5}}{2}$$

However, since $F_n\in\mathbb{N}$, it is evident that $F_n>0$, therefore, $\lim_{n\to\infty}\frac{F_n}{F_{n-1}}>0$, meaning that $\Phi>0$; thus, $\Phi=\frac{1+\sqrt{5}}{2}$.

Considering that $F_n=F_{n-1}+F_{n-2}$ is in fact a recurring equation, one may make use of the linear recurrence equation

$$a_n=c_1a_{n-1}+c_2a_{n-2}+\cdots+c_ka_{n-k}$$

where the c_i are non-negative integers. Knowing that the recurring formula, $F_n = F_{n-1} + F_{n-2}$, has two initial conditions, the general solution of the linear recurrence equation

$$a_n = c_1 {x_1}^n + c_2 {x_2}^n$$

may be obtained, where x_i represents the each of the two roots of the quadratic equation above: $\frac{1+\sqrt{5}}{2}$ and $\frac{1-\sqrt{5}}{2}$. Substituting the two initial conditions, $F_1 = 1$ and $F_2 = 1$, the two following equations can be derived from the linear recurrence equation:

$$F_1 = c_1 {x_1}^1 + c_2 {x_2}^1 \qquad F_2 = c_1 {x_1}^2 + c_2 {x_2}^2$$

$$1 = c_1 \left(\frac{1+\sqrt{5}}{2}\right)^1 + c_2 \left(\frac{1-\sqrt{5}}{2}\right)^1 \qquad 1 = c_1 \left(\frac{1+\sqrt{5}}{2}\right)^2 + c_2 \left(\frac{1-\sqrt{5}}{2}\right)^2$$

From these two equations, the follwing system of equations is derived:

$$\begin{cases} 1 = c_1 \left(\frac{1+\sqrt{5}}{2}\right) + c_2 \left(\frac{1-\sqrt{5}}{2}\right) \\ 1 = c_1 \left(\frac{3+\sqrt{5}}{2}\right) + c_2 \left(\frac{3-\sqrt{5}}{2}\right) \end{cases}$$

This system of equations can be solved as follows to derive c_1 and c_2:

$$c_1\left(\frac{1+\sqrt{5}}{2}\right)+c_2\left(\frac{1-\sqrt{5}}{2}\right)=c_1\left(\frac{3+\sqrt{5}}{2}\right)+c_2\left(\frac{3-\sqrt{5}}{2}\right)$$

$$c_1\left(\frac{1+\sqrt{5}-3-\sqrt{5}}{2}\right)=c_2\left(\frac{3-\sqrt{5}-3-\sqrt{5}}{2}\right)$$

$$-c_1=c_2$$

$$c_1\left(\frac{1+\sqrt{5}}{2}\right)-c_1\left(\frac{1-\sqrt{5}}{2}\right)=1$$

$$c_1\left(\frac{2\sqrt{5}}{2}\right)=1$$

$$c_1=\frac{1}{\sqrt{5}}$$

$$c_2=-\frac{1}{\sqrt{5}}$$

Thus, the final explicit formula for computing the nth term of the Fibonacci sequence can be defined as follows:

$$F_n=\frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^n-\frac{1}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^n$$

$$=\frac{1}{\sqrt{5}}\left\{\left(\frac{1+\sqrt{5}}{2}\right)^n-\left(\frac{1-\sqrt{5}}{2}\right)^n\right\}$$

$$=\frac{\Phi^n-(-\Phi)^{-n}}{\sqrt{5}}$$

5.2 Inductive proof of the explicit formula for the Fibonacci sequence

In order to prove that the derived explicit formula above is true for all cases $n\in\mathbb{N}$, proof by induction will be used. Henceforth, let $P(n)$ be the proposition that $F_n=\frac{\Phi^n-(-\Phi)^{-n}}{\sqrt{5}}$ holds true for all cases $n\in\mathbb{N}$, where F_n is the nth Fibonacci term and $\Phi=\frac{1+\sqrt{5}}{2}$.

1. **Check that $P(n)$ is true for when $n=1$ and $n=2$**

$$
\begin{aligned}
F_1 &= \frac{\Phi^1 - (-\Phi)^{-1}}{\sqrt{5}} \\
&= \frac{\left(\frac{1+\sqrt{5}}{2}\right) - \left(\frac{1-\sqrt{5}}{2}\right)}{\sqrt{5}} \\
&= \frac{\left(\frac{2\sqrt{5}}{2}\right)}{\sqrt{5}} \\
&= 1
\end{aligned}
$$

$$
\begin{aligned}
F_2 &= \frac{\Phi^2 - (-\Phi)^{-2}}{\sqrt{5}} \\
&= \frac{\left(\frac{3+\sqrt{5}}{2}\right) - \left(\frac{3-\sqrt{5}}{2}\right)}{\sqrt{5}} \\
&= \frac{\left(\frac{2\sqrt{5}}{2}\right)}{\sqrt{5}} \\
&= 1
\end{aligned}
$$

Therefore, it can be proved that the base cases for $P(n)$ holds true.

2. **Assume that $P(n)$ is true for when $n=k$ and $n=k+1$**

$$F_k = \frac{\Phi^k - (-\Phi)^{-k}}{\sqrt{5}}$$

$$F_{k+1} = \frac{\Phi^{k+1} - (-\Phi)^{-k-1}}{\sqrt{5}}$$

3. **Prove that $P(n)$ is true for when $n = k+2$**

$$F_{k+2} = F_{k+1} + F_k$$

$$\frac{\Phi^{k+2} - (-\Phi)^{-k-2}}{\sqrt{5}} = \frac{\Phi^{k+1} - (-\Phi)^{-k-1}}{\sqrt{5}} + \frac{\Phi^{k} - (-\Phi)^{-k}}{\sqrt{5}}$$

$$= \frac{1}{\sqrt{5}}\left\{\Phi^{k+1} + \Phi^{k} - (-\Phi)^{-k-1} - (-\Phi)^{-k}\right\}$$

$$= \frac{1}{\sqrt{5}}\left[\Phi^{k}(\Phi+1) - (-\Phi)^{-k}\{(-\Phi)^{-1}+1\}\right]$$

From $\Phi^2 - \Phi - 1 = 0$ it can be known that $\Phi + 1 = \Phi^2$ and $(-\Phi)^{-1} = (-\Phi)^{-2} - 1$. Substituting these values into the equation above returns the following:

$$= \frac{1}{\sqrt{5}}\left[\Phi^{k}(\Phi^2) - (-\Phi)^{-k}\{(-\Phi)^{-2} - 1 + 1\}\right]$$

$$= \frac{1}{\sqrt{5}}\left[\Phi^{k}\cdot\Phi^2 - (-\Phi)^{-k}\cdot(-\Phi)^{-2}\right]$$

$$= \frac{\Phi^{k+2} - (-\Phi)^{-k-2}}{\sqrt{5}}$$

LHS = RHS

Thus, by the principle of mathematical induction, $P(n)$ holds true for all cases $n \in \mathbb{N}$.

5.1 Derivation of the explicit formula for the research question

Intuition lead me to exploring the ratios of different sequences which exist in the form of $a_n = a_{n-1} + a_{n-m}$ and discovering that these ratios can be utilised to derive explicit formulas that are quite accurate in estimating the nth term of a sequence. To explain the process in which the formula is derived, the sequence, $N_n = N_{n-1} + N_{n-3}$, otherwise known as Narayana's cows sequence ("A000930 As a Simple Table."), will be analysed.

The ratio of consecutive terms in Narayana's cows sequence also converges to a certain value, which will henceforth be denoted as φ. The limit is as follows:

$$\lim_{n\to\infty}\frac{N_n}{N_{n-1}} = 1.465571231\cdots$$

$$\approx 1.4656$$

The values of ratios will henceforth be truncated to four decimal places, as it will be sufficient to output a fairly accurate value. This limit in particular also happens to be the real positive root of the polynomial, $f(\varphi)=\varphi^3-\varphi^2-1$, as suggested by the following graph:

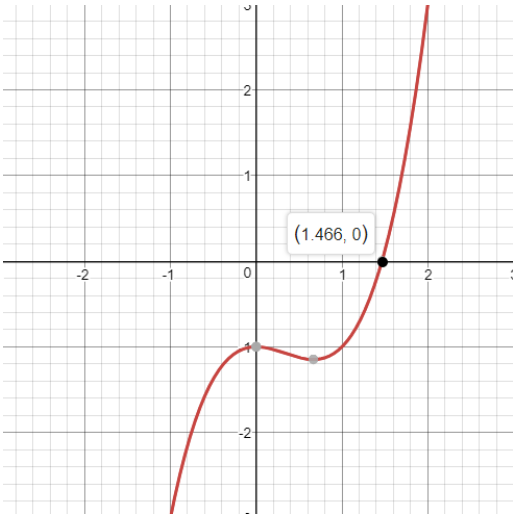


Figure 14: Graph of $f(\varphi)=\varphi^3-\varphi^2-1$

Further investigation of different sequences in the form of $a_n=a_{n-1}+a_{n-m}$ reveals that there exists a correlation between the ratios of increasing consecutive terms and the real positive root of polynomials in the form of $f(x)=x^m-x^{m-1}-1$. This convergent ratio was calculated through the list of sequences on The On-Line Encyclopedia of Integer Sequences, and the real positive root of the $f(x)=x^m-x^{m-1}-1$ polynomial was determined by means of Wolfram Alpha's "Solve for x calculator":

m	$\lim_{n\to\infty}\frac{a_n}{a_{n-1}}$	$x=\left\{x\in\mathbb{R}^+\middle\vert x^m-x^{m-1}-1=0\right\}$
2	1.61803398…	1.61803398…
3	1.46557123…	1.46557123…
4	1.38027756…	1.38027756…
5	1.32471795…	1.32471795…
6	1.28519903…	1.28519903…
7	1.25542287…	1.25542287…
8	1.23205463…	1.23205463…

(OEIS; rfreberg)

Therefore, it can be deduced that the ratio of increasing consecutive terms in the sequence, $a_n = a_{n-1} + a_{n-m}$, is equal to the real positive root of the polynomial, $f(x) = x^m - x^{m-1} - 1$. I intuitively tried substituting Φ from Binet's formula with the ratio of consecutive terms in Narayana's cows sequence, henceforth denoted as φ, and derived a new equation for estimating the n[th] term of the sequence.

Since the $\sqrt{5}$ from $F_n = \frac{\Phi^n - (-\Phi)^{-n}}{\sqrt{5}}$ would change depending on the ratio of consecutive terms, I believed that substituting the ratio and calculating for the denomiator for the n[th] term for when n approaches infinity would allow the derivation of a new denominator, x.

$$\begin{aligned}
\lim_{n\to\infty} N_n &\approx \lim_{n\to\infty} \frac{\varphi^n - (-\varphi)^{-n}}{x} \\
&\approx \lim_{n\to\infty} \frac{1.4656^n - (-1.4656)^{-n}}{x} \\
x &\approx \lim_{n\to\infty} \frac{1.4656^n - (-1.4656)^{-n}}{N_n} \\
&\approx 2.39671370\cdots \\
&\approx 2.3967
\end{aligned}$$

Therefore, for the case of Narayana's cows sequence, the explicit formula for estimating the n[th] term of the sequence can be derived as follows, where $\varphi \approx 1.4656$, a value which approximates to the ratio between two consecutive terms from the sequence:

$$N_n \approx \frac{\varphi^n - (-\varphi)^{-n}}{2.3967}$$

Testing for a random input of 20 returns a fairly accurate result:

$$N_n \approx \frac{\varphi^n - (-\varphi)^{-n}}{2.3967}$$
$$N_{20} \approx \frac{1.4656^n - (-1.4656)^{-n}}{2.3967}$$
$$872 \approx \frac{1.4656^{20} - (-1.4656)^{-20}}{2.3967}$$
$$\approx 872.35921140\cdots$$

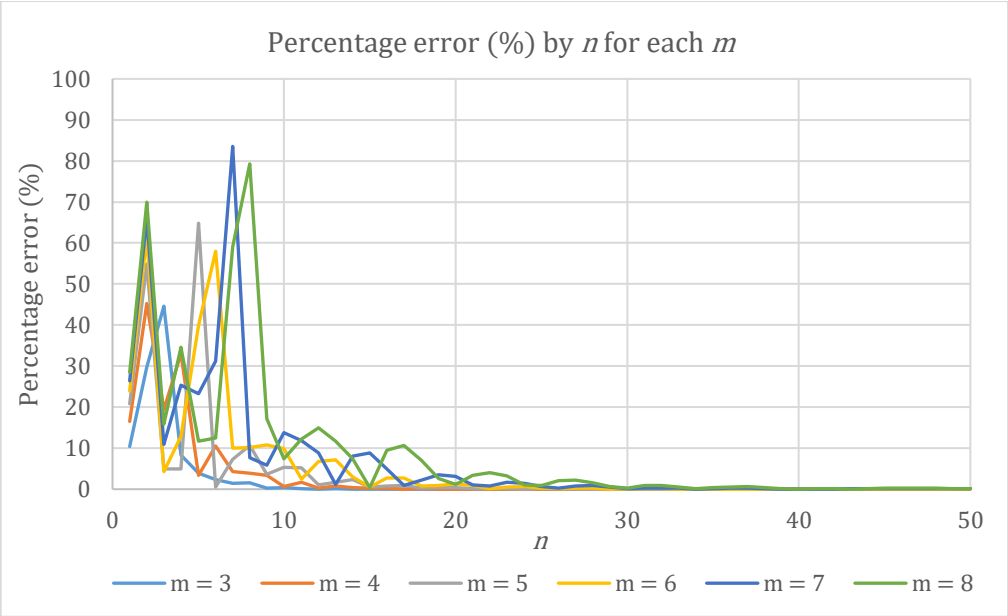
$$\text{Percentage error} = \left|\frac{\text{Value}_{\text{output}} - \text{Value}_{\text{actual}}}{\text{Value}_{\text{actual}}}\right| \times 100$$
$$= \left|\frac{872.35921140\cdots - 872}{872}\right| \times 100$$
$$= 0.00041193\cdots \times 100$$
$$= 0.04119397\cdots$$
$$\approx 0.0412\%$$

Although not perfect, the equation does return quite an accurate result, with a percentage error of 0.0412%. However, this equation is not fully reliable when it comes to smaller inputs, as the outputs tend to deviate significantly from the actual values:

$$N_n \approx \frac{\varphi^n - (-\varphi)^{-n}}{2.3967}$$
$$N_3 \approx \frac{1.4656^3 - (-1.4656)^{-3}}{2.3967}$$
$$1 \approx \frac{1.4656^3 - (-1.4656)^{-3}}{2.3967}$$
$$\approx 1.44604577\cdots$$

$$\text{Percentage error} = \left|\frac{\text{Value}_{\text{output}} - \text{Value}_{\text{actual}}}{\text{Value}_{\text{actual}}}\right| \times 100$$
$$\text{Percentage error} = \left|\frac{1.44604577\cdots - 1}{1}\right| \times 100$$
$$= 0.44604577\cdots \times 100$$
$$= 44.60457722\cdots$$
$$\approx 44.6046\%$$

Analysis of the percentage errors of cases involving different values of m in the recursive equation, $a_n = a_{n-1} + a_{n-m}$, further supports this observation, as the percentage uncertainty of the output of the explicit formula nears 0% as n approaches infinity.



Thus, it can be concluded that the explicit formula for the dynamic programming question can be used as a means of estimating the n^{th} term of a given sequence, but is not accurate enough when it comes to smaller inputs; therefore, can only be used as a band-aid solution to the issue of long computation times.

6 Conclusion

Returning to the original research question, "What is the total number of ways in which $1\times\alpha$ tiles can fit in a $\alpha\times\beta$ board?", the total number of arrangements depends on three different conditions:

1. $\alpha > \beta$
2. $\alpha = \beta$
3. $\alpha < \beta$

provided $\alpha, \beta \in \mathbb{N}$

A conclusive recursive formula can then be derived based on these conditions:

$$f(\alpha,\beta)=\begin{cases} 1 & (\alpha>\beta) \\ 2 & (\alpha=\beta) \\ f(\beta-1)+f(\beta-\alpha) & (\alpha<\beta) \end{cases}$$

Although this function can be used to accurately compute an output for a given case, due to the recursive nature of the function, efficient computation becomes impossible as inputs increase in value. To tackle this issue of inefficient recursive calculations, the explicit formula for calculating the nth term of the Fibonacci sequence, Binet's formula, $F_n=\frac{\Phi^n-(-\Phi)^{-n}}{\sqrt{5}}$, where $\Phi=\frac{1+\sqrt{5}}{2}$, can be examined and modified to derive an explicit formula for closely estimating the total number of arrangements of a given case:

$$A_n=\frac{\theta^n-(-\theta)^{-n}}{x}$$

where $\theta=\lim_{n\to\infty}\frac{A_n}{A_{n-1}}$ and x is a converging variable for when n approaches infinity.

Though I believe that an explicit formula for computing an exact output exists, due to time and knowledge constraints, I am unable to deduce such a formula.

Works cited

"A000045 As a Simple Table." The On-Line Encyclopedia of Integer Sequences, 2018, oeis.org/A000045/list.

"A000930 As a Simple Table." The On-Line Encyclopedia of Integer Sequences, 2018, oeis.org/A000930/list.

"A003269 As a Simple Table." The On-Line Encyclopedia of Integer Sequences, 2018, oeis.org/A003269/list.

"A003520 As a Simple Table." The On-Line Encyclopedia of Integer Sequences, 2018, oeis.org/A003520/list.

"A005708 As a Simple Table." The On-Line Encyclopedia of Integer Sequences, 2018, oeis.org/A005708/list.

"A005709 As a Simple Table." The On-Line Encyclopedia of Integer Sequences, 2018, oeis.org/A005709/list.

"A005710 As a Simple Table." The On-Line Encyclopedia of Integer Sequences, 2018, oeis.org/A005710/list.

Davis, Samuel G., and Edward T. Reutzel. "The Computational Implications of Variations in State Variable/State Ratios in Dynamic Programming and Total Enumeration." *Journal of the Operational Research Society*, vol. 35, no. 11, 1984, pp. 1003–1011., doi:10.1057/jors.1984.196.

Elton, Edwin J., and Martin J. Gruber. "Dynamic Programming Applications in Finance." *The Journal of Finance*, vol. 26, no. 2, May 1971, pp. 473–506., doi:10.2307/2326061.

Heyde, C. C. "On a Probabilistic Analogue of the Fibonacci Sequence." Journal of Applied Probability, vol. 17, no. 04, Dec. 1980, pp. 1079–1082., doi:10.1017/s0021900200097369.

rfreberg. "Solve for X Calculator." Wolfram|Alpha, 2018 Wolfram Alpha LLC—A Wolfram Research Company, 7 Sept. 2011, www.wolframalpha.com/widgets/view.jsp?id=7953c4ea52a4873d32cc72052f3dcb10.

Spouge, John L. "Speeding up Dynamic Programming Algorithms for Finding Optimal Lattice Paths." SIAM Journal on Applied Mathematics, vol. 49, no. 5, Oct. 1989, pp. 1552–1566., doi:10.1137/0149094.

Stensland, Gunnar, and Dag Tjostheim. "Optimal Investments Using Empirical Dynamic Programming with Application to Natural Resources." The Journal of Business, vol. 62, no. 1, Jan. 1989, pp. 99–120., doi:10.1086/296453.