A Dynamic Programming Approach for the Brachistochrone Problem

Anais do Computer on the Beach

André Eduardo Alessi
andrealessi@usp.br
Universidade de São Paulo - Escola de Engenharia de São Carlos - Departamento de Engenharia de Produção
São Carlos, São Paulo, Brazil

Gabriel Vinicius Bacci
gabriel.bacci@usp.br
Universidade de São Paulo - Instituto de Ciências Matemáticas e de Computação
São Carlos, São Paulo, Brasil

Hugo Gielamo Próspero
hugop@usp.br
Universidade de São Paulo - Instituto de Ciências Matemáticas e de Computação
São Carlos, São Paulo, Brasil

José Eduardo Saroba Bieco
jose.bieco@usp.br
Universidade de São Paulo - Instituto de Ciências Matemáticas e de Computação
São Carlos, São Paulo, Brasil

Eduardo Fontoura Costa
efcosta@icmc.usp.br
Universidade de São Paulo - Instituto de Ciências Matemáticas e de Computação
São Carlos, São Paulo, Brasil

Abstract

In this paper, we address the brachistochrone problem using Dynamic Programming and an extension that accounts for forbidden regions or obstacles. The brachistochrone problem seeks the curve of fastest descent under gravity between two points, minimizing travel time. While traditional approaches derive the cycloidal curve as the solution using calculus of variations, this work employs a discrete, grid-based Dynamic Programming formulation to approximate the optimal trajectory. Computational experiments showcase the method's flexibility, particularly in adapting to constraints such as forbidden regions, and its ability to dynamically recalculate paths. Despite some limitations in angular resolution due to discretization, the proposed approach demonstrates robustness and scalability in addressing constrained trajectory optimization problems. Moreover, this project lays the groundwork for extending the proposed methodology to more complex scenarios, such as incorporating randomness into the trajectory, where the adaptability of this approach can be effectively leveraged.

Keywords

Dynamic Programming, Brachistochrone Problem, Brachistochrone with Obstacles.

1 Introduction

The brachistochrone problem, first posed by [1] in 1696, asks: given two points, A and B, in a vertical plane, what curve will a particle, moving solely under the influence of gravity, trace to travel from A to B in the shortest possible time? This problem, graphically illustrated in Figure 1, led to significant advances in physics and mathematics and was initially solved by Bernoulli and other prominent mathematicians of that era, such as Newton, Leibniz, Huygens, and L'Hospital, who identified the cycloidal curve as the solution for the shortest travel time. Since then, the brachistochrone problem has been studied and applied in different theoretical contexts, such as the calculus of variations and the optimal control theory [21].

Building on these foundational studies and practical applications, significant advances have been made in exploring the brachistochrone problem in various fields. As highlighted by [7] and [8], it can be used, for example, in optimizing rocket maneuvers to reduce energy consumption, determining the shortest paths in transport networks, designing ideal tracks for drag racing and athletics competitions, and developing advanced algorithms for computer graphics and engineering simulations. Moreover, the brachistochrone has applications in autonomous navigation and control systems, where minimizing travel time is essential for the efficiency and safety of vehicles and drones. Another practical example is its use in sports competitions, such as skiing, where the optimized trajectory to minimize travel time can offer a competitive advantage.

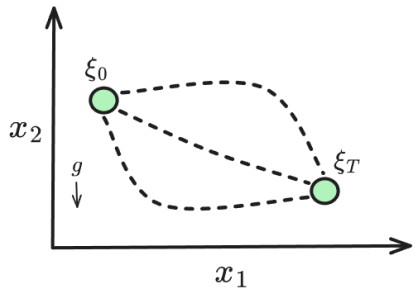

Figure 1: Illustration of the brachistochrone problem.

Several advances in the study of the brachistochrone problem have occurred over the past several decades. In the 1970s, the work of [13] applied principles of energy conservation to model speed and used Dynamic Programming (DP) to optimize the trajectory, employing the successive sweep method to divide the path into intervals and perform optimization. In the 2000s, [14] introduced a

discretization approach that divided the path into linear segments, adjusting the heights of the points to minimize the total travel time. The author used a technique based on DP, called Coarse-to-Fine (CFDP), which increased precision without compromising computational efficiency.

Another way to solve this problem was explored by [5], where authors used the Sequential Quadratic Programming (SQP) method to solve the brachistochrone problem in a scenario with viscous friction, iteratively adjusting the trajectory to account for frictional forces. Later, in [6], the authors combined DP with interior point methods to solve optimal control problems with constraints, applying these concepts to the friction-brachistochrone problem, which simplified the calculation of Lagrange multipliers and improved the efficiency of the solution. [16] addressed the case of a rolling sphere using non-linear optimization techniques, modeling the trajectory to ensure rolling without slipping. More recently, in [11], the authors addressed the problem with a maximum curvature constraint, something of great relevance for practical applications such as vehicle dynamics and robotics. Using Pontryagin's Maximum Principle, they managed to respect the curvature constraints while minimizing travel time. Then, [4] extended the brachistochrone problem by including constraints on the angle of inclination of the trajectory, also applying Pontryagin's Maximum Principle to optimize the path in scenarios with viscous friction. The most recent work [12] adapted the problem to the field of quantum physics, creating the concept of a "quantum brachistochrone," which aims to minimize the transition time between quantum states by optimizing the system's evolution under a time-dependent Hamiltonian.

An interesting study exploring the application of DP to the brachistochrone problem [17] shows an implementation that approximates the ideal trajectory through a discrete grid, highlighting, however, the limitations of DP. In particular, it was observed that more refined grids result in instability and larger errors, moving the solution away from the exact cycloidal curve. The authors claim that this stability problem makes DP less suitable for the brachistochrone compared to continuous methods of optimal control and the calculus of variations, which offer greater accuracy and computational efficiency for continuous trajectories.

Several studies have used discretization techniques and DP. However, despite the advances, no work has been found in the literature so far that focuses entirely on solving the brachistochrone problem through pure DP. Although the works of [5], [13], and [21] have proposed a unified framework to solve the problem, they do not fully explore the potential of DP as the main method.

To address these limitations, this paper proposes a novel solution based entirely on pure DP to solve the brachistochrone problem. One of the main benefits of this approach is to make it easy to add constraints to the problem, such as obstacles included along the curve or the insertion of uncertainty into the problem. Pure DP not only allows such flexibility but also simplifies code implementation. Moreover, the developed code can be easily reused in different scenarios, making this approach particularly efficient. Unlike what was suggested in previous studies, which indicated that pure DP might not be efficient, this work aims to provide a balance between flexibility and computational efficiency, offering a practical solution to the problem of determining the brachistochrone curve.

To provide a comprehensive understanding of the methodology and results, this work is organized as follows. Section 2 presents and details the formulation of the brachistochrone problem, which is the focus of this project, as well as the discretization of the problem using DP, a technique employed to generate approximate solutions. Section 3 describes the computational experiments conducted in this project, analyzing the performance of our implementation in scenarios with varying grid resolutions, testing its behavior under a restricted number of control actions, and examining cases involving prohibited regions. Finally, Section 4 discusses the conclusions drawn and outlines potential directions for future research. The notation used in this work can be checked in Table 1.

x_1, x_2	Vertical and horizontal positions (scalars)
ξ_0, ξ_T	Start and end positions (vectors)
g	Gravity (scalar)
T	Time to travel (scalar)
$x(t)$	Position at instant t, continuous time (vector)
$x^*(t)$	Optimal object trajectory (vector)
x_k	Discrete state (vector)
u_k	Control variable (vector)
$u(t)$	Angle, continuous time (scalar)
$\mathcal{X}$	Continuous time problem domain (polytope in $\mathfrak{R}^2$)
$\mathcal{X}_g$	Gridded space state (countable subset of $\mathcal{X}$)
$x_i(s)$	Interpolated trajectories (vector)
$\mathcal{S}$	Subset of space state (subset of $\mathcal{X}$)
C	Cycloid (subset of $\mathcal{X}$)
$r, h, \hat{h}$	Cycloid parameters (scalar)
N	Time horizon (scalar)
k	Time instant (scalar)
$f(\cdot)$	Dynamic constraint function
$c(x_k, u_k)$	Cost function
$c_N(x_N)$	Terminal cost function
δ	Step value (scalar)
$\mathcal{U}_g$	Restricted control space (polytope in $\mathfrak{R}^2$)
$V_k(x_k)$	Value function
Δt	Time variation (scalar)
Δs	Space variation (scalar)
v	Velocity (scalar)
$\overline{v}$	Average velocity (scalar)
γ	Vertical distance relative to the starting point (scalar)
α	Trajectory angulation (scalar)
n	Number of points (scalar)

Table 1: Notation used in this work

2 Problem formulation and its approximation via discrete DP

The Brachistochrone problem consists of minimizing the time T required for an object to travel between specified start and end points within a uniform gravitational field, assuming energy conservation. In addition, the direction of the object can be freely controlled. In this paper, we constrain the object to move within a box of size

10×10 meters[1], represented by the set $\mathcal{X} = \{0 \leq x_1 \leq 10, 0 \leq x_2 \leq 10\}$), where x_1 and x_2 refer to the horizontal and vertical positions, respectively, and compose the vector of position $x = \begin{bmatrix} x_1 & x_2 \end{bmatrix}'$. The object starts at rest at the position $\xi_0 = \begin{bmatrix} 0 & 10 \end{bmatrix}'$ and ends at $\xi_T = \begin{bmatrix} 10 & 0 \end{bmatrix}'$. These sizes and positions are configurable and can be adjusted as needed. Formally,

$$\begin{aligned} &\min \quad T \\ &\text{s.t. } x(0) = \xi_0,\ x(T) = \xi_T,\ \frac{dx}{dt}(0) = 0, \\ &\frac{d^2x(t)}{dt^2} = \begin{bmatrix} g\sin(u(t))\cos(u(t)) \\ -g\sin^2(u(t)) \end{bmatrix} \end{aligned} \tag{1}$$

where $x(t) \in \mathcal{X}$ is the position of the object at time instant t, and $u(t)$ is the angle with the horizontal line that the object is traveling at time t.

This problem can be tackled using different approaches, such as the variational method, leading to a well-known optima trajectory that describes a cycloid in the state space [18]; we shall denote the optimal solution by $x^\star(t)$, and give details about the cycloid in Appendix A.

In this paper, we are interested in a numerical solution using DP, which is easy to implement in a computer and easy to extend for the case with space constraints, as we shall see later on. Therefore, we consider a classic, approximate discrete DP formulation, following the literature on DP [2, 10]:

$$\min_{u_0,\ldots,u_{N-1}} \sum_{k=0}^{N-1} c(x_k, u_k) + c_N(x_N), \tag{2}$$

$$\text{s.t. } x(0) = \xi_0, \tag{3}$$

$$x_{k+1} = f(x_k, u_k),\ k = 0, 1, \ldots, N-1, \tag{4}$$

where N is the time horizon, $f : \mathcal{X} \to \mathcal{X}$ is a given function describing the dynamic constraints, c is the cost per stage, c_N is the terminal cost, and u_k is the control action.

We use a restricted, gridded space state $\mathcal{X}_\text{g} \subset \mathcal{X}$, a regular grid of n points in each axis, leading to a total of n^2 elements in the form

$$\mathcal{X}_\text{g} = \left\{ \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ \delta \end{bmatrix}, \begin{bmatrix} 0 \\ 2\delta \end{bmatrix}, \ldots, \begin{bmatrix} 10 \\ 10 \end{bmatrix} \right\}, \tag{5}$$

where $\delta = 10/n$ is a step value. As usual, we can expect that the more points n, the better the solution, and the more it approaches the optimal cycloid solution.

As for the decision variable u_k, we decided to express it in the form $u_k = \begin{bmatrix} h & v \end{bmatrix}'$ where h and v are the displacements in the vertical and horizontal axis, respectively, leading to a simple dynamics

$$x_{k+1} = f(x_k, u_k) = x_k + u_k. \tag{6}$$

A restricted control space $\mathcal{U}_\text{g}$ is adopted, e.g.,

$$\mathcal{U}_\text{g} = \left\{ \begin{bmatrix} 0 \\ -\delta \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ \delta \end{bmatrix}, \begin{bmatrix} \delta \\ -\delta \end{bmatrix}, \begin{bmatrix} \delta \\ 0 \end{bmatrix}, \begin{bmatrix} \delta \\ \delta \end{bmatrix} \right\}, \tag{7}$$

which corresponds to actions that make the object move from x_k to the closest neighbors in $\mathcal{X}_\text{g}$ (except moves to the left).

[1]All units in this paper are in the International System of Units (SI); positions are expressed in meters [m].

The DP algorithm for solving the above problem consists of computing the so-called value functions V_k backward in discrete time, starting with $V_N(x_N) = c_n(x_N)$ and then the recursive relation

$$V_k(x_k) = \inf_{u_k} [c_k(x_k, u_k) + V_{k+1}(f_k(x_k, u_k))],\ k = N-1, \ldots, 0, \tag{8}$$

which provides the optimal cost $V_0(x_0)$ for (2). Also, by storing the optimal u_k in each stage of (8), we obtain a feasible control policy.

To make the above formulation a suitable discrete approximation for (1), we set $c_N(\xi_T) = 0$ and $c_N(x) = \infty$, $x \neq \xi_T$, as a way of forcing $x(N) = \xi_T$.

The cost function c is considered as the time taken to move from one state to the other. As it is well-known in Physics [9], the variation of time Δt from one point to another can be expressed as

$$c(x_k, u_k) = \Delta t = \frac{2\|x_{k+1} - x_k\|}{v(x_{k+1}) + v(x_k)},$$

where $v(x_k)$ is the speed of the particle at position x_k, so that $(1/2)(v(x_{k+1}) + v(x_k))$ is the average speed between the states. To calculate the speed, let us denote $x_k = [x_{k_1} \quad x_{k_2}]$; applying conservation of energy, $\frac{1}{2}mv(x_k)^2 = mg\gamma$, where γ is the vertical distance between the current state and the starting point, $\gamma = 10 - x_{k_2}$, which lead to

$$v(x_k) = \sqrt{2g(10 - x_{k_2})}.$$

Combining the above equations,

$$c(x_k, u_k) = \frac{2\|x_{k+1} - x_k\|}{\sqrt{2g(10 - x_{(k+1)_2})} + \sqrt{2g(10 - x_{k_2})}}, \tag{9}$$

Now, let us explore in more detail the relevance of a proper choice for $\mathcal{U}_\text{g}$. One common rule of thumb is to choose $\mathcal{U}_\text{g}$ as the smallest set that allows for reaching the entire state space (within the time horizon) starting from the initial condition ξ_0; this would lead to the above choice for $\mathcal{U}_\text{g}$, which is of small cardinality and does allow to reach any point in $\mathcal{X}_\text{g}$ (from the initial condition ξ_0). However, **this choice for $\mathcal{U}_\text{g}$ has a major drawback**: it only allows for a few angulations of the trajectory. To better explain this point, let us define, for a given gridded trajectory $x(k), 0 \leq k \leq N$, the linearly interpolated trajectory $x_\text{i}(s), 0 \leq s \leq N$ satisfying $x_\text{i}(k) = x(k)$ and $x_\text{i}(s) = (s-k)x(k+1) + (1-s+k)x(k)$, $k \leq s \leq k+1$; see an illustration in Figure 2. Also, define the angulation of a trajectory with the horizontal line as $\alpha(x(s)) = \tan^{-1} \frac{\partial x_1(s)}{\partial x_2(s)}$. With the above $\mathcal{U}_\text{g}$, $\alpha(x_\text{i}(s))$ only takes values in the set $\{-\pi, -\pi/2, 0, \pi/2, \pi\}$, as illustrated in Figure 2. It is worth mentioning that the lack of diversity of angulations of the interpolated solution, as explained above, does not mean that we are not able to approximate the optimal, cycloid solution in some senses, for example, in the "visual sense" illustrated in Figure 3 and its detail in Figure 4, where we adopted $n = 1000$, corresponding to $\delta = 0.01$ [m]. However, the illustrated trajectories are not similar in terms of the most relevant aspect: time to travel between points $\xi_0 = [0;\ 10]$ and $\xi_T = [10;\ 0]$, which is approximately 1.842952 seconds in the optimal solution ($x^\star(t)$) versus approximately 2.004831 seconds in the approximated solution $x_\text{i}(s)$. This issue persists with different n (different grids for the space state), as explained in Appendix B, where the heuristic we have used to find the approximate x_i, shown in Figure 3, is also explained. In order to mitigate the issue of the lack of angulation

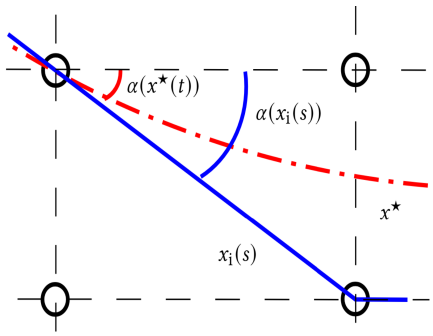

Figure 2: A portion of the state space $\mathcal{X}$ and its subset $\mathcal{X}_{\mathrm{g}}$ based on n points (white circles). The dot-dash red line represents the optimal solution $x^\star$ and the blue line is the interpolated solution x_{i}. The angulations of these trajectories are also illustrated at the same point in the grid.

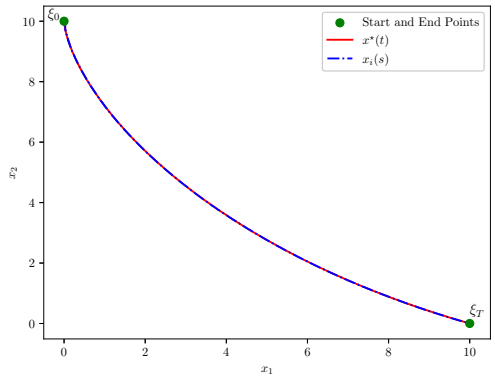

Figure 3: Complete state space $\mathcal{X}$ containing the initial and terminal points $[0;\ 10]$ and $[10;\ 0]$. The optimal solution $x^\star(t)$ (red, continuous line) and an approximated, interpolated solution $x_{\mathrm{i}}(s)$ (blue, dash-dot) with $\mathcal{U}_{\mathrm{g}}$ as given in (5). The trajectories are almost completely overlapping, making the red curve difficult to see.

explained above, we choose a larger set of control actions. In this paper, we have considered

$$\mathcal{U}_{\mathrm{g}} = \left\{ \begin{bmatrix} 0 \\ -8\delta \end{bmatrix}, \begin{bmatrix} 0 \\ -7\delta \end{bmatrix}, \ldots, \begin{bmatrix} 0 \\ 8\delta \end{bmatrix}, \begin{bmatrix} \delta \\ -8\delta \end{bmatrix}, \ldots, \begin{bmatrix} 8\delta \\ 8\delta \end{bmatrix} \right\}, \qquad (10)$$

Next, we will describe the computational experiments done to evaluate our approach and report its results.

3 Computational experiments

The experiments were performed on a machine equipped with 32 GB of RAM (3200 MHz CL 18) and an AMD Ryzen 5 5600x CPU running at stock settings (6 cores, 12 threads, 4.6 GHz frequency).

The developed DP approach was analyzed using the gridded state space $\mathcal{X}_{\mathrm{g}}$ with different numbers of points n; for simplicity,

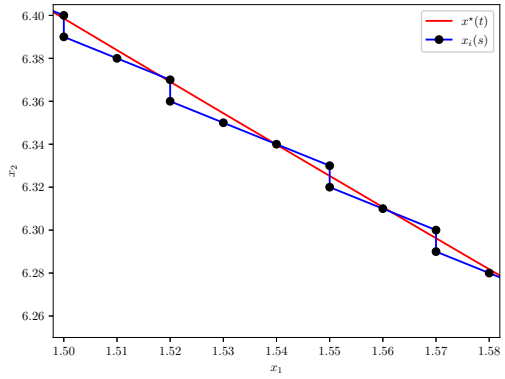

Figure 4: Detail of Figure 3 with a zoom of a portion of the trajectories, now making evident that $x^\star(t)$ (red) and $x_{\mathrm{i}}(s)$ (blue) are different, and also confirming that the angulation of the $x_{\mathrm{i}}(s)$ belong to the set $\{-\pi, -\pi/2, 0, \pi/2, \pi\}$.

we shall refer to these sets as "$n \times n$ grids", in this section. The grid for the control actions is as in (10), and the time horizon is set as $N = \frac{n}{2}$. Regarding the computational complexity, the number of times (8) is evaluated is $n \times n \times N = n^3/2$, which is also proportional to the execution time and memory size required.

The time for the object to travel from ξ_0 to ξ_T and the execution times are shown in Table 2. The results indicate that the solution quality improves significantly when increasing the grid resolution from 20×20 to 100×100. However, further increases beyond 200×200 do not yield significant improvements in the precision of the trajectory time, as illustrated in Figure 6. This suggests that excessively high resolutions introduce additional computational costs without relevant gains in solution quality, rendering larger resolutions unnecessary for this problem.

Curve	Time (seconds)	Exec time (seconds)
DP (10×10)	1.855791	< 1
DP (20×20)	1.847736	< 1
DP (50×50)	1.844821	< 1
DP (100×100)	1.844038	2
DP (200×200)	1.843666	15.8 ± 1
DP (500×500)	1.843542	265.6 ± 8.9
DP (1000×1000)	1.843528	2136.6 ± 45.6
Cycloid	1.842952	

Table 2: Results with control space $\mathcal{X}_{\mathbf{u}}$ as in (10) and different grids $n \times n$. The code was executed 5 times for each grid in order to calculate the execution time. The cycloid solution is presented for comparison.

The interpolated trajectories $x_{\mathrm{i}}(s)$ obtained by DP are illustrated in Figure 5 and 6. In Figure 5, it is evident that as n increases (i.e., the grid becomes finer), the trajectories get closer to the cycloid, as expected. In Figure 6, we observe excellent agreement between the trajectory obtained with the 200×200 grid and the cycloid solution,

highlighting the method's accuracy. Finer grids, such as 500×500 and 1000×1000, were omitted because their curves practically coincide with the cycloid.

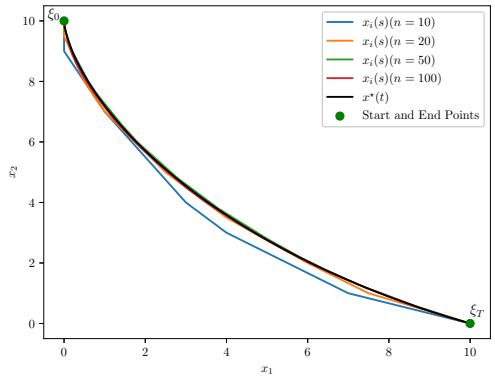

Figure 5: Curves generated by DP with different grid configurations (10 X 10, 20 X 20, 50 X 50, and 100 X 100) and the cycloid solution $x^\star(t)$.

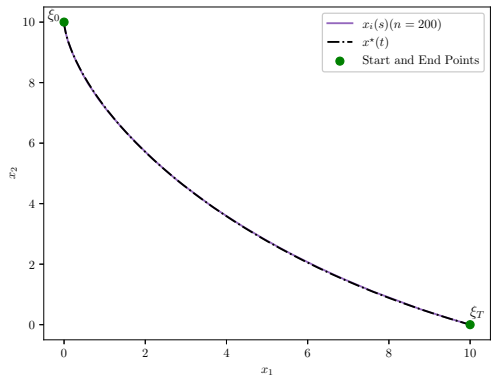

Figure 6: Comparison of the curve generated by DP with 200 X 200 grid (purple) and the cycloid solution (black). Curves obtained with higher resolution grids (500 X 500 and 1000 X 1000) are omitted, as they are indistinguishable from the cycloid.

Remark 1. *Although we have already explained in Section II that a low-resolution grid for the control space $\mathcal{U}_g$ as in (7) does not properly approximate the necessary angulations for the trajectory, we decided to perform experiments with it just to confirm this statement. Results are given in Table 3, showing that the obtained times do not approach the optimal solution (approximately 1.8429) even when we use high-resolution grids for the state space, as in the case with $n = 500$. Figure 7 illustrates results with $n = 200$, quite discrepant from the cycloid.*

It is perhaps worth mentioning that PD ensures that these results are optimal in a certain sense: they do minimize the travel time when actions are constrained to the low-dimensional $\mathcal{U}_g$ in (7) (that is, when the particle / object has a few possible angulations) and the object has to satisfy the gridded state space. They do not approach the cycloid because, in its context, the object is free (subject to no constraints).

Curve	*Time (seconds)*	*Exec time (seconds)*
DP (20 × 20)	*1.897302*	*< 1*
DP (50 × 50)	*1.896857*	*< 1*
DP (100 × 100)	*1.896788*	*< 1*
DP (200 × 200)	*1.896788*	*3*
DP (500 × 500)	*1.896785*	*48.2 ± 1.3*
Cycloid	*1.842952*	

Table 3: Results with control space $\mathcal{U}_g$ as in (7) and different grids $n \times x$. The code was executed 5 times for each grid in order to calculate the execution time. The cycloid solution is included for comparison.

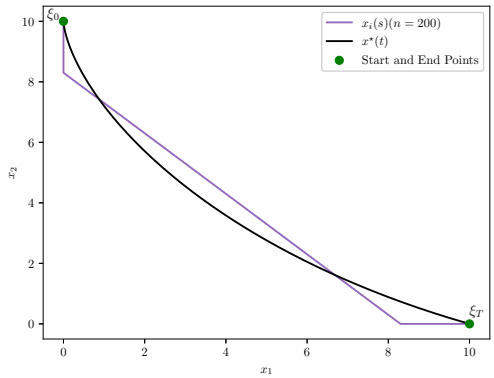

Figure 7: Curve generated by DP with 200 X 200 grid and $\mathcal{U}_g$ as in (7) (control actions reduced to the nearest neighbor), compared to the cycloid solution. This configuration demonstrates how a proper choice for $\mathcal{U}_g$ is key for approaching $x^\star$.

3.1 Incorporating barriers in the state space

The DP approach demonstrates exceptional flexibility when applied to environments with spatial constraints, such as forbidden regions of the state space. These are critical for simulating real-world scenarios, e.g., avoiding obstacles in robotics [3, 15] or restricted air spaces in aerospace trajectory planning [20]. The computational experiment conducted below highlights DP's ability to navigate these challenges effectively while maintaining the goal of minimizing travel time.

Here we consider $\mathcal{X}_g$ with $n = 500$ points. Initially, we computed the trajectory without considering any obstacles, resulting in a path that closely approximated the ideal cycloidal curve with 1.84354

seconds. The forbidden regions were then introduced by marking specific grid cells as "forbidden", simulating spatial constraints. The DP algorithm remains essentially unaltered (we only change $\mathcal{X}_g$), and generated a new solution with travel time 1.89359, which is quite similar to the optimal travel time obtained before. This stepwise introduction of obstacles underscores the robustness of DP in adapting to environments where constraints often emerge unpredictably. Figure 8 illustrates this process, comparing the original trajectory, computed without constraints, to the recalculated path that avoids a forbidden region. The new optimal trajectory smoothly detours around the restricted zone. This shows the DP's capacity to integrate both local and global cost optimizations effectively, even when faced with abrupt changes in the environment.

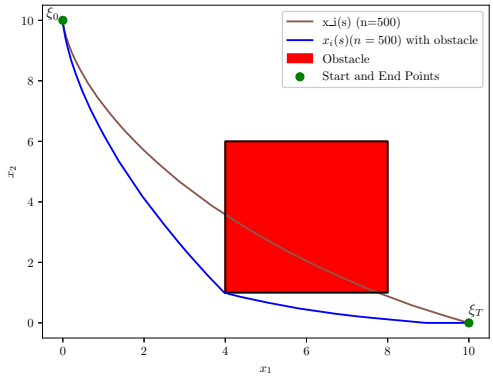

Figure 8: Adaptation of the brachistochrone trajectory to avoid forbidden regions.

Despite the recalculated trajectory being able to successfully avoid the forbidden region, a closer examination reveals an important consideration related to interpolation and trajectory representation. As shown in Figure 9, zooming in the vicinity of the obstacle reveals that the straight-line interpolation between consecutive trajectory points might appear to cross into the restricted area. This visual artifact, however, is misleading. The actual trajectory points, computed via DP, remain strictly outside the forbidden region, adhering to all constraints. Considering the approximate, gridded DP approach in this paper, we implemented a verification step to ensure that the line segment connecting two trajectory points does not intersect the forbidden region, thereby enhancing the realism and robustness of the trajectory representation.

The recalculated trajectories produced by DP demonstrate efficient detours that preserve the overall quality of the solution. For example, when a forbidden region was placed directly along the original trajectory, the DP algorithm recalculated an alternative path that bypassed the obstacle with minimal additional travel time.

This adaptability has significant practical implications. In robotics, mobile units operating in dynamic environments, such as automated warehouses, must avoid static obstacles like shelving units and dynamic ones like other robots [3, 15]. Similarly, in aerospace trajectory planning, avoiding zones of space debris or gravitational

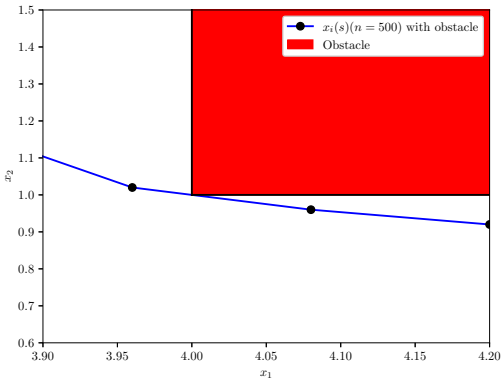

Figure 9: Zoomed view of the trajectory near the forbidden region.

anomalies is essential while optimizing fuel efficiency[19, 20]. Unlike continuous optimization methods, which often require recalculating the entire solution, DP offers the advantage of recalibrating specific trajectory segments efficiently as constraints evolve.

These results highlight DP's versatility in generating feasible and efficient paths, even in environments with dynamic and complex constraints [19]. The experiments conducted in this study illustrate the potential of DP to solve not only theoretical problems, such as the brachistochrone, but also practical challenges in constrained and dynamic scenarios.

4 Concluding remarks

In this paper, we have presented a successful application of the DP approach to the Brachistochrone problem, including some cases with obstacles in the state space. As expected, when there are no barriers, the solution approaches the well-known cycloid solution when the number of points in the gridded state space $\mathcal{X}_g$ increases, as expected.

The cardinality of the control action space $\mathcal{U}_g$ plays a central role in the success of the DP approach, and surprisingly, this approach has never been properly addressed in the literature, to the best of our knowledge. As we have found out in the experiments and explained in Section 2, it is not enough to pick a $\mathcal{U}_g$ that allows reaching every point of $\mathcal{X}_g$ (as in (7)); a larger set as in (10) has to be used to give a suitable "diversity of angulations" of the trajectory, to approach the optimal Brachistochrone solution.

The extension of the brachistochrone problem to the case with barriers, which was studied in Section 3.1, showed that DP performs even better with restricted state space (in terms of execution time). Future work will analyze cases with intermittent and stochastic barriers, as well as stochastic perturbations in the trajectory and control, which can be found in real-world applications. Stochastic scenarios like these require the evaluation of the expectation of the value function; depending on the case and distribution of the random variables involved, the additional complexity may vary from a simple change in the formula of the cost per stage $c(\cdot, \cdot)$ to the incorporation of one additional loop in the DP algorithm to

estimate the expected value of the value function, in which case the user may control the accuracy of the estimate for establishing a trade-off between optimality and computational complexity.

ACKNOWLEDGMENTS

This work was supported in part by Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) under Grants 316534/2021-8, in part by Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP) under Grants FAPESP-CEPID 2013/07375-0 and 2024/05642-5, and in part by Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) under Grants 88887.984492/2024-00, 88887.003126/2024-00 and 88887.083738/2024-00.

References

[1] J. Bernoulli. 1696. Problema novum ad cujus solutionem mathematici invitantur. A new problem to whose solution mathematicians are invited. *Acta Eruditorum* 18 (1696), 269.

[2] Dimitri P. Bertsekas. 1995. *Dynamic Programming and Optimal Control.* Athena Scientific.

[3] Jarad Cannon, Kevin Rose, and Wheeler Ruml. 2021. Real-Time Motion Planning with Dynamic Obstacles. In *Symposium on Combinatorial Search.* https://api.semanticscholar.org/CorpusID:315285

[4] O.Yu. Cherkasov and N.V. Smirnova. 2022. On the Brachistochrone problem with state constraints on the slope angle. *International Journal of Non-Linear Mechanics* 60, 3 (2022), 123–135.

[5] C. R. Dohrmann and R. D. Robinett. 1997. Efficient Sequential Quadratic Programming Implementations for Equality-Constrained Discrete-Time Optimal Control. *Journal of Optimization Theory and Applications* 95, 2 (1997), 323–346.

[6] C. R. Dohrmann and R. D. Robinett. 1999. Dynamic Programming Method for Constrained Discrete-Time Optimal Control. *Journal of Optimization Theory and Applications* 101, 2 (1999), 259–283.

[7] J.E. Drummond and G.L. Downes. 1971. The brachistochrone with acceleration: A running track. *Journal of Optimization Theory and Applications* 7, 6 (1971), 444 – 449. https://doi.org/10.1007/BF00931980

[8] Rabiatul Adawiah Fadzar and Md Yushalify Misro. 2023. Brachistochrone Curve Representation via Transition Curve. *Mathematics and Statistics* 11, 1 (2023), 213 – 222. https://doi.org/10.13189/ms.2023.110125

[9] D. Halliday, R. Resnick, and J. Walker. 2017. *Fundamentals of Physics, Volume 1.* Wiley. https://books.google.com.br/books?id=4iqVDwAAQBAJ

[10] P. R. Kumar and Pravin Varaiya. 1986. *Stochastic Systems: Estimation, Identification, and Adaptive Control.* Prentice-Hall.

[11] S.S. Lemak and M.D. Belousova. 2021. The brachistochrone problem with constraints on the curvature of the trajectory. *IFAC-PapersOnLine* 54, 13 (2021), 437–442.

[12] P.G. Morrison. 2023. Quantum Brachistochrone for the Majorana Particle. *arXiv:2305.19285* (2023).

[13] I. H. Mufti. 1970. The Successive Sweep Method and the Second Variation. *Lecture Notes in Operations Research and Mathematical Systems* 27, 1 (1970), 23–26.

[14] C. Raphael. 2001. Coarse-to-Fine Dynamic Programming. *Pattern Recognition* 34, 8 (2001), 1651–1662.

[15] R. Saravanan, S. Ramabalan, and C. Balamurugan. 2009. Evolutionary multi-criteria trajectory modeling of industrial robots in the presence of obstacles. *Engineering Applications of Artificial Intelligence* 22, 2 (2009), 329–342. https://doi.org/10.1016/j.engappai.2008.06.002

[16] O. Taylor and A. Rodriguez. 2018. Optimal shape and motion planning for dynamic planar manipulation. *Autonomous Robots* 43, 3 (2018), 327–344.

[17] Weimich. 2023. Solving the Brachistochrone Problem using Dynamic Programming and the GNU Octave Tool. https://www.dsp-weimich.com/digital-signal-processing/solving-the-brachistochrone-problem-using-dynamic-programming-and-the-gnu-octave-tool/ Accessed: 2024-10-30.

[18] Eric W. Weisstein. 2024. Brachistochrone Problem. https://mathworld.wolfram.com/BrachistochroneProblem.html From MathWorld–A Wolfram Web Resource. Accessed: 27 November 2024.

[19] Xiaobo Zheng, Feiran Xia, Defu Lin, Tianyu Jin, Wenshan Su, and Shaoming He. 2024. Constrained Parameterized Differential Dynamic Programming for Waypoint-Trajectory Optimization. *Aerospace* 11, 6 (2024). https://doi.org/10.3390/aerospace11060420

[20] Tianqi Zhu, Jianliang Mao, Linyan Han, Chuanlin Zhang, and Jun Yang. 2024. Real-Time Dynamic Obstacle Avoidance for Robot Manipulators Based on Cascaded Nonlinear MPC With Artificial Potential Field. *IEEE Transactions on Industrial Electronics* 71, 7 (2024), 7424–7434. https://doi.org/10.1109/TIE.2023.3306405

[21] S. Šalinic , A. Obradovic , Z. Mitrovic , and S. Rusov. 2011. Brachistochrone with limited reaction of constraint in an arbitrary force field. *Nonlinear Dynamics* 69, 3 (2011), 211–222.

A The optimal solution $x^\star(t)$

Let us denote by $\mathcal{S} \in \mathcal{X}$ the set traced by the particle when traveling in shortest time from ξ_0 to ξ_T; this set can be characterized as follows. A cycloid is given by $\mathcal{C} = \{[\xi_{0_1} + r(h - sen(h)), \xi_{0_2} - r(1 - cos(h))]', r \geq 0, 0 \leq h \leq 2\pi\}$; we select $r, \hat{h}$ to satisfy the end point, $\xi_{T_1} = \xi_{0_1} + r(\hat{h} - sen(\hat{h}))$, $\xi_{T_2} = \xi_{0_2} - r(1 - cos(\hat{h}))$ (which can be done, e.g., using Newton's method as explained later in this Appendix); finally, the optimal trajectory $x^\star(t)$ belongs to the set $\mathcal{S} = \{[\xi_{0_1} + r(h - sen(h)), \xi_{0_2} - r(1 - cos(h))]', r \geq 0, 0 \leq h \leq \hat{h}\}$. Note that this gives the region $\mathcal{S}$ formed by the optimal trajectory parameterized by h, not by t, for instance, $x_1^\star(t) \neq \xi_{0_1} + r(h - sen(h))$ in general. For finding $x^\star(t)$, we either discretize the interval $0 \leq h \leq \hat{h}$ or discretize x_1 axis and obtain the corresponding x_2.

To obtain the values of r and $\hat{h}$, we built a non-linear system of equations using ξ_0 and ξ_T:

$$\begin{cases} r(\hat{h} - sen(\hat{h})) + \xi_{0_1} - \xi_{T_1} = 0 \\ -r(1 - cos(\hat{h})) + \xi_{0_2} - \xi_{T_2} = 0 \end{cases}$$

Such system of equations can be easily solved using Newton's method for a system of non-linear equations, where $f(r, \hat{h}) = r(\hat{h} - sen(\hat{h})) + \xi_{0_1} - \xi_{T_1}$ and $g(r, \hat{h}) = -r(1 - cos(\hat{h})) + \xi_{0_2} - \xi_{T_2} = 0$, and we use the Euclidean norm of $[r_{k+1} - r_k, \hat{h}_{k+1} - \hat{h}_k]$ as a stopping criterion.

$$\begin{bmatrix} \frac{\partial f}{\partial r} & \frac{\partial f}{\partial \hat{h}} \\ \frac{\partial g}{\partial r} & \frac{\partial g}{\partial \hat{h}} \end{bmatrix} \begin{bmatrix} r_{k+1} - r_k \\ \hat{h}_{k+1} - \hat{h}_k \end{bmatrix} = - \begin{bmatrix} f(r_k, \hat{h}_k) \\ g(r_k, \hat{h}_k) \end{bmatrix}$$

After finding the value of r we can draw the curve, discretizing the x_1 axis, then finding its corresponding value of h applying the fixed point method on $f(h) = r(h - sen(h)) + \xi_{0_1} - \xi_{T_1}$ and applying it on $g(h) = -r(1 - cos(h)) + \xi_{0_2} - \xi_{T_2} = 0$, creating a list of coordinates which approximate the curve. We have implemented this method in the Python programming language to get the parameters for the cycloid, draw the curve, and calculate the time it would take for a particle to go through it using the same method used in DP, therefore finding $x^\star(t)$.

B Direct neighborhood heuristic

We present here a heuristic method for finding a "visual approximation" for the cycloid solution $x^\star$, which was employed in Section 2 (in particular, for obtaining Figures 3 and 4). After presenting the heuristic, we give some results in Table 4 and a brief discussion.

The algorithm is as follows Start at ξ_0 and compute the distances from the curve $\mathcal{S}$ (defined in Appendix A) to the following points: 1) the point to the right of ξ_0, 2) the point below ξ_0, and 3) the point on the right-below diagonal to ξ_0. Then, move to the point closest to the curve $\mathcal{S}$ among these three candidates. Note that these points are elements of the grid $\mathcal{X}_g$.

We used grids of 10×10, 100×100, 1000×1000 and 2000×2000 to illustrate how refining the grid does not help to approximate the cycloid solution $x^\star$. Results are summarized in Table 4 and show that more points not only fail to approximate $x^\star$ but also degrade

Curve	Time (seconds)
Newton's method cycloid	1.842952
Heuristic method ($1m$ interval)	1.919449
Heuristic method ($0.1m$ interval)	1.957759
Heuristic method ($0.01m$ interval)	2.004831
Heuristic method ($0.005m$ interval)	2.049348

Table 4: Results of the implemented heuristic method

travel time. This behavior arises from the limitations of the defined control action $\mathcal{U}_{\mathrm{g}}$ as explained in Section 2.

The heuristic given above was implemented in Python and is available in our GitHub repository (https://github.com/Programacao-Dinamica-SME5979-2024-2/Brachistochrone-curve). Results are illustrated in Table 4, which makes clear that a solution that "visually approaches" $x^\star$ does not necessarily provide small travel time, in fact, larger n makes the travel time worse.