Revista Iberoamericana de Automática e Informática Industrial

Revista Iberoamericana de Automática e Informática Industrial 19 (2022) 37-47

www.revista-riai.org

Método de error de Bellman con ponderación de volumen para mallado adaptativo en programación dinámica aproximada

Leopoldo Armesto[a,*], Antonio Sala[b]

[a]*Instituto de Diseño y Fabricación, Universitat Politècnica de València, Cno. Vera, s/n, 46022, Valencia, España.*
[b]*Instituto U. de Automática e Informática Industrial, Universitat Politècnica de València, Cno. Vera, s/n, 46022, Valencia, España.*

To cite this article: Armesto, L., Sala, A. 2022. Volume-weighted Bellman error method for adaptive meshing in approximate dynamic programming. Revista Iberoamericana de Automática e Informática Industrial 19, 37-47. https://doi.org/10.4995/riai.2021.15698

Resumen

El control óptimo y aprendizaje por refuerzo llevan asociada una "función de valor" que debe ser adecuadamente aproximada. Estos problemas de aproximar funciones de valor tienen, usualmente, diferentes requerimientos de precisión en diferentes regiones del espacio de estados. Un mallado uniforme tiene problemas porque desperdicia recursos en regiones en las que la función de valor es suave, mientras que no tiene la suficiente resolución en zonas con grandes cambios en dicha función. El presente trabajo propone una metodología de programación dinámica aproximada con mallado adaptativo, para poder adaptarse a dichos requerimientos cambiantes sin incrementar en exceso el número de parámetros del aproximador. La propuesta se basa en mallados simpliciales y en el error en la ecuación de Bellman con un criterios para añadir y quitar puntos del mallado: se modificarán propuestas de la literatura incluyendo el volumen de los símplices afectados en los criterios y se detallarán las manipulaciones de la triangulación necesarias.

Palabras clave: Control inteligente, programación dinámica aproximada, control óptimo, aprendizaje.

Volume-weighted Bellman error method for adaptive meshing in approximate dynamic programming

Abstract

Optimal control and reinforcement learning have an associate "value function" which must be suitably approximated. Value function approximation problems usually have different precision requirements in different regions of the state space. An uniform gridding wastes resources in regions in which the value function is smooth, and, on the other hand, has not enough resolution in zones with abrupt changes. The present work proposes an adaptive meshing methodology in order to adapt to these changing requirements without incrementing too much the number of parameters of the approximator. The proposal is based on simplicial meshes and Bellman error, with a criteria to add and remove points from the mesh: modifications to proposals in earlier literature including the volume of the affected simplices are proposed, alongside with methods to manipulate the mesh triangulation.

Keywords: Intelligent control, approximate dynamic programming, optimal control, neural learning.

1. Introducción

El diseño de controladores basados en optimización es de gran relevancia actual en múltiples facetas teóricas (Liberzon, 2011; Ariño et al., 2010) y prácticas (Armesto et al., 2015; Duarte-Mermoud and Milla, 2018; Rubio et al., 2018).

Si los modelos son lineales, el control óptimo lineal cuadrático (conocido como LQR, del inglés Linear Quadratic Regulator), es una metodología bien estudiada en muchos textos (Athans and Falb, 2013; Albertos and Sala, 2006). En el caso no-lineal, en general no existen soluciones con una expresión analítica explícita del control óptimo. Una alternativa sería ha-

*Autor para correspondencia: larmesto@idf.upv.es

cer control predictivo (Camacho and Bordons, 2010) no lineal, resolviendo problemas de optimización en línea (Allgower and Zheng, 2012; Ziogou et al., 2013) o iterativa (Ariño et al., 2014; Armesto et al., 2015; Li and Todorov, 2007), si bien el principal problema que plantean es que el tiempo de resolución en línea.

Es interesante mencionar que, dentro del aprendizaje de controladores óptimos, existen técnicas que ajustan directamente los parámetros θ de una política de control $\pi(x, \theta)$ a base de repeticiones exhaustivas de experimentos; estas estrategias son conocidas como *búsqueda de políticas*, y en el trabajo (Deisenroth et al., 2013) se hace una revisión en profundidad de dichas metodologías. Estas opciones de exploración son de interés si el tamaño de x es alto y el tamaño de θ es bajo. Este manuscrito persigue acercarse a soluciones óptimas a horizonte infinito mediante programación dinámica (Bertsekas and Tsitsiklis, 1996), que permitan obtener controladores aplicables a todo el espacio de estados, y se dejan, intencionalmente, fuera de los objetivos a plantear las técnicas de control predictivo basadas en simulaciones a horizonte finito, así como la búsqueda aleatoria dirigida de políticas.

En particular, el área de control inteligente (Santos, 2011) de sistemas no lineales es, desde sus inicios, muy activa en este tipo de problemas, dando lugar a las disciplinas que se conocen como programación dinámica aproximada (Busoniu et al., 2010; Lewis and Vrabie, 2009; Lewis and Liu, 2013) o aprendizaje por refuerzo (Sutton and Barto, 1998). Ambas concepciones están muy relacionadas. En efecto, ambas técnicas buscan expresiones explícitas de funciones de valor $V(x)$ o de acción-valor $Q(x, u)$ relacionadas con la ecuación de Bellman (Bertsekas and Tsitsiklis, 1996; Sutton and Barto, 1998) y controladores asociados que sean una *aproximación* razonablemente cercana al controlador óptimo (que solo en algunos casos de espacio de estados discreto puede ser calculado de forma exacta). Pese a las similitudes teóricas en el manejo de $V(x)$ o $Q(x, u)$, el concepto de aprendizaje por refuerzo en literatura suele entenderse como más orientado hacia la mejora progresiva de un control a base de repeticiones de la tarea adquiriendo datos por experimentación en línea, mientras que el concepto de programación dinámica aborda problemas de aprendizaje de controladores óptimos a partir de un modelo con el que simular todas las transiciones posibles. Es este segundo enfoque en el que se centrará este trabajo.

Para obtener las funciones de valor anteriormente mencionadas en programación dinámica, los algoritmos más extendidos son los denominados "iteración de política" (*policy iteration* en lengua inglesa, PI) o "iteración de valor" (*value iteration* en la literatura en lengua inglesa, VI), que progresivamente van mejorando los estimados de las funciones y controladores intervinientes, véase, por ejemplo Lewis and Liu (2013); Busoniu et al. (2010), para más detalles.

En espacios de estado contínuos, se necesita usar aproximadores funcionales (splines, redes neuronales, ...), con algún vector de parámetros ajustables θ, para las funciones de valor $V(x, \theta)$, $Q(x, u, \theta)$ o controladores $u = \pi(x, \theta)$. El teorema de aproximación funcional universal afirma que muchos aproximadores pueden aproximar a un error arbitrariamente pequeño una función continua en una zona de modelado compacta añadiendo un suficiente número de regresores, polinomios, neuronas, etc. (Hornik et al., 1989).

El problema fundamental de la programación dinámica aproximada es que las garantías de optimalidad se pierden al usar aproximadores $V(x, \theta)$ y, asimismo, los algoritmos iterativos clásicos PI o VI podrían no converger (Fairbank and Alonso, 2012), por pérdida de contractividad (Busoniu et al., 2010); ello podría requerir cambios en ellos, utilizando, por ejemplo, iteraciones descendiendo por gradiente para minimización del residuo de Bellman (Antos et al., 2008; Díaz et al., 2020). Como alternativas a las metodologías iterativas, cambiando igualdades por desigualdades en la ecuación de Bellman, reformulaciones del problema como un problema de programación lineal pueden obtener, sin iteraciones, la solución óptima en casos de espacios de estado y control discretos, o una aproximación a la misma (De Farias and Van Roy, 2003). El coste computacional de una solución de programación lineal es mayor que el de una iteración PI/VI, pero solo es necesario ejecutarlo una vez, lo cual, añadido a que se eliminan los problemas de convergencia, supone una ventajosa opción en algunos casos (Díaz et al., 2019).

En algunos casos, algunos interpoladores borrosos para espacios de estados continuos también verifican condiciones de convergencia (Buşoniu et al., 2010; Busoniu et al., 2010; Bertsekas, 2018) para PI/VI. Eso ocurre también en aproximadores afines por tramos con estructura simplicial Grüne (1997), y este tipo de aproximaciones serán el objeto de estudio en este trabajo. En estos aproximadores, una triangulación del espacio de estados permitirá aproximar la función de valor en puntos arbitrarios como una interpolación lineal de los valores estimados en los vértices.

El objetivo de este trabajo será presentar una metodología para refinar mallados simpliciales irregulares para programación dinámica aproximada, y compararla con propuestas como Grüne (1997); Munos and Moore (2002) en un caso de estudio. Se propondrá un mallado simplicial basado en triangulaciones Delaunay, se indicarán las operaciones para su refinamiento o eliminación de puntos y se evaluarán criterios de adición/eliminación de puntos basados en la función de valor y la ecuación de Bellmann para aprovechar de forma eficiente los recursos limitados (número de vértices, esto es, de puntos de interpolación).

La estructura de este trabajo es la siguiente: la Sección 2 presentará definiciones preliminares y planteará el problema a resolver, la Sección 3 desarrollará la metodología propuesta de adaptaciones de mallados en programación dinámica y la Sección 4 presentará un ejemplo de control de un sistema montaña-carrito y un análisis comparativo con otras propuestas en literatura sobre el mismo ejemplo. El trabajo se cerrará con una breve sección de conclusiones.

2. Definiciones preliminares

Sea

$$x^+ = f(x, u) \tag{1}$$

el modelo de un sistema dinámico discreto no lineal, siendo $x \in \mathbb{X} \subseteq \mathbb{R}^n$ el estado del sistema, x^+ representa el estado sucesor como consecuencia de aplicar la entrada $u \in \mathbb{U}$, siendo $\mathbb{U}$ un conjunto finito de posibles acciones de control y $f : \mathbb{X} \times \mathbb{U} \to \mathbb{R}^n$ la dinámica del sistema. Consideraremos

$\mathbb{T} := \mathbb{R}^n \sim \mathbb{X}$ como conjunto *terminal*, es decir, todo aquello del espacio $\mathbb{R}^n$ que no sea $\mathbb{X}$.

Definiremos una política $u = \pi(x) : \mathbb{X} \to \mathbb{U}$ como una función que representa el controlador en bucle cerrado del sistema, que se transformará con dicha política en $x^+ = f(x, \pi(x))$.

Dado un estado inicial x_0, el coste de aplicar una política $\pi(x)$ partiendo de dicho estado, se denomina función de valor, $V^\pi : \mathbb{R}^n \to \mathbb{R}$. Dicha función se define como:

$$V^\pi(x_0) := V_\mathbb{T}(x_{T_f}) + \sum_{k=0}^{T_f-1} \gamma^k L(x_k, \pi(x_k)) \tag{2}$$

siendo x_k el estado en el instante de tiempo k, de la trayectoria $\tau^\pi(x_0) = \{x_0, x_1, ..., x_{T_f}\}$ como consecuencia de aplicar la política $\pi(x)$, y siendo T_f el instante en el que la trayectoria alcanza el conjunto terminal $\mathbb{T}$, o infinito si siempre permanece en $\mathbb{X}$. A la función escalar $L(x, u) : \mathbb{X} \times \mathbb{U} \to \mathbb{R}^+$, se le denomina "coste inmediato", $0 < \gamma < 1$ es el factor de descuento y a $V_\mathbb{T} : \mathbb{T} \mapsto R^+$ se le denomina "coste terminal".

El coste terminal tiene dos interpretaciones:

a) Un coste reducido (premio) o elevado (castigo) al alcanzar un conjunto de estados objetivo o prohibido en el segundo caso, en tiempo finito.
b) La función de valor de trayectorias que continúan evolucionando en el tiempo con un cierto controlador terminal.

La metodología propuesta funcionará independientemente de la interpretación que se haya supuesto al plantear el problema de control óptimo.

En cuanto al factor de descuento, $\gamma < 1$ garantiza convergencia en gran número de resultados teóricos de estimación de función de valor, pero en aplicaciones prácticas de control óptimo $\gamma = 1$ suele ser lo que realmente se busca. Si γ es menor pero muy cercano a 1, las leyes de control óptimo resultantes serán muy parecidas si $\gamma^{T_f} \approx 1$.

El control óptimo trata de buscar una política óptima $\pi^*(x)$ que minimice $V^\pi(x)$ de entre todas las políticas factibles. Para ello, se puede aplicar el bien conocido principio de optimalidad de Bellman que establece los fundamentos de la Programación Dinámica y que estipula que:

$$V^*(x) = \min_{u \in \mathbb{U}} \; L(x, u) + \gamma V^*(f(x, u)) \tag{3}$$

Nótese que si $f(x, u) \in \mathbb{T}$ entonces V^* coincide con el coste terminal $V_\mathbb{T}$.

El lado derecho de la igualdad se suele representar como el operador de Bellman $\mathcal{T}$:

$$[\mathcal{T}(\mathrm{V})](x) := \min_{u \in \mathbb{U}} L(x, u) + \gamma V(f(x, u)) \tag{4}$$

de modo que la solución de la ecuación de Bellman es un punto fijo de $\mathcal{T}$, $V^* = \mathcal{T}[V^*]$.

Por tanto, la política óptima será aquella que minimice el lado derecho de la ecuación (3):

$$\pi^*(x) = \arg\min_{u \in \mathbb{U}} \; L(x, u) + \gamma V^*(f(x, u)). \tag{5}$$

Por lo general, podremos calcular la función de valor de una política $\pi(x)$ arbitraria resolviendo la ecuación de Bellman:

$$V^\pi(x) = L(x, \pi(x)) + \gamma V^\pi(f(x, \pi(x))) \tag{6}$$

Nótese que, con el fin de evaluar el operador de Bellman y la política óptima, es necesario conocer el modelo del sistema (1). Además, téngase en consideración, que el factor de descuento ha sido definido estrictamente menor a 1 para garantizar un coste finito $V^\pi(x)$ si $\mathbb{X}$ es compacto y tanto $L(x, u)$ como $V_\mathbb{T}(x)$ están acotados.

2.1. Programación Dinámica Aproximada

La Programación Dinámica Aproximada es necesaria porque, en general, no es posible describir de forma exacta la función de valor $V^\pi(x)$ para cualquier valor de x arbitrario con un conjunto finito de parámetros θ, en forma $V^\pi(x) \equiv V^\pi(x, \theta)$, excepto en unos pocos casos especiales, como el control óptimo LQR.

En este caso, debemos usar aproximadores funcionales parametrizados que nos permitan describir el error de la diferencia temporal (temporal difference error, TDE, en lengua inglesa) como:

$$\epsilon(x, \theta) := V^\pi(x, \theta) - L(x, \pi(x)) - \gamma V^\pi(f(x, \pi(x)), \theta) \tag{7}$$

Por tanto, la evaluación aproximada de una política $\pi(x)$ consiste en obtener el conjunto de parámetros θ^* que minimizan el error cuadrático de la diferencia temporal como:

$$\theta^* := \arg\min_\theta \|\epsilon(x, \theta)\|^2 \tag{8}$$

siendo

$$\|\epsilon(x, \theta)\|^2 := \int_\mathbb{X} \epsilon(x, \theta)^2 \, dx \tag{9}$$

La evaluación de dicha política se conoce, en la terminología de la lengua inglesa y dentro de este ámbito como *Policy Evaluation*.

De igual forma, dada una función de valor $V^\pi(x, \theta)$, se puede obtener una política que mejore el valor de la función de valor a un paso, es decir, que obtiene la política óptima a usar en un instante suponiendo que posteriormente se usará la política $\pi(x)$ para el resto de instantes, siendo, en este caso, los parámetros dados:

$$\pi^+(x, \theta) := \arg\min_{u \in \mathbb{U}} L(x, u) + \gamma V^\pi(f(x, u), \theta) \tag{10}$$

Este paso es conocido en la literatura inglesa como *Policy Improvement*, ya que se puede demostrar que $V^{\pi^+}(x, \theta) \leq V^\pi(x, \theta)$ si $V^\pi(x, \theta)$ fuera coincidente con la función de valor "exacta" $V^\pi(x)$, lo que implica una mejor política.

En este sentido, las propuestas más populares en la literatura para resolver el problema de programación dinámica aproximada son:

1. **iteración de la política** o PI en la que se ejecutan de forma iterativa los pasos de evaluación (6) y mejora (10), véase, por ejemplo, (Lewis and Vrabie, 2009);
2. **iteración aproximada de la función de valor** o (fitted value iteration, en lengua inglesa) en la que a partir de un estimado inicial de la función de valor $V(x, \theta^{[0]})$ se obtiene una solución para (8) de forma iterativa, esto es:

$$\theta^{[k+1]} := \arg\min_{\hat{\theta}} \left\| V(x, \hat{\theta}) - \min_{u \in \mathbb{U}} \left(L(x, u) + \gamma V(f(x, u), \theta^{[k]}) \right) \right\|^2 \tag{11}$$

hasta que converja, a $\theta^* := \theta^{[\infty]}$.

3. **Optimización global**, donde un optimizador genérico pudiese obtener el óptimo (8), esto es:

$$\theta^* := \arg\min_{\hat{\theta}} \left\| V(x,\hat{\theta}) - \min_{u\in\mathbb{U}} \left(L(x,u) + \gamma V(f(x,u),\hat{\theta}) \right) \right\|^2 \tag{12}$$

   No obstante, esta optimización global es costosa, debido al mínimo de u anidado y, solo en el caso de determinadas expresiones parametrizando la política que se busque, la función de valor esto puede dar lugar a resultados computacionalmente tratables, como por ejemplo los controladores Takagi-Sugeno y aproximaciones cuadráticas de Díaz et al. (2020).
4. **Cotas inferiores o superiores**, reemplazando la igualdad en (3) o (6) por una desigualdad permite obtener cotas inferiores (optimistas) o superiores (conservadoras) de la función de valor, usando programación lineal (De Farias and Van Roy, 2003; Díaz et al., 2019).

La aproximación a la política óptima, una vez calculada la función de valor aproximada, se reduce a:

$$\pi(x,\theta^*) := \arg\min_{u\in\mathbb{U}} \left(L(x,u) + \gamma V(f(x,u),\theta^*) \right), \forall x \in \mathbb{X}, \tag{13}$$

Como se ha comentado en (3), se asumirá que si $f(x,u) \in \mathbb{T}$ entonces debe usarse el coste terminal $V_{\mathbb{T}}$ en lugar de expresiones parametrizadas $V(x,\theta)$ en todas las expresiones desde (7) hasta (13) y en las expresiones análogas que aparecen en los desarrollos que siguen.

Caso finito, soluciones exactas. En el caso de sistemas con una representación finita del espacio de estados, los algoritmos PI y VI convergen a la política óptima con una implementación mediante tablas en las que los mismos parámetros son en realidad el valor que toma la función de valor (Sutton and Barto, 1998).

2.2. Representación paramétrica de funciones de valor

En este trabajo, la función de valor será aproximada por un conjunto de parámetros $\theta = [\theta_1\ \theta_2\ \ldots\ \theta_m]^T \in \mathbb{R}^m$ combinados de forma lineal con un conjunto de regresores:

$$V(x,\theta) := \sum_{i=1}^{m} \phi_i(x)\theta_i = \Phi^T(x)\theta, \tag{14}$$

siendo $\phi(x) : \mathbb{X} \to \mathbb{R}$ y $\Phi(x) = [\phi_1(x)\ \phi_2(x)\ \ldots\ \phi_m(x)]^T \in \mathbb{R}^m$.

Una clase particular de regresores, denominados "borrosos" o "interpolativos" Buşoniu et al. (2010); Díaz et al. (2020) expresa la función de valor $V(x,\theta)$ como una interpolación (suma borrosa) de los valores de la función en un conjunto de puntos $\mathcal{X} := \{x_1, x_2, \ldots, x_M\}$ prototipo, esto es $\theta_i \equiv V(x_i)$. Si para un x que no esté en $\mathcal{X}$ se puede expresar $V(x,\theta) = \sum_{i=1}^{M} \mu_i(x)\theta_i = \sum_{i=1}^{M} \mu_i(x)V(x_i)$ con $0 \le \mu_i(x) \le 1$ y $\sum_{i=1}^{k} \mu_i(x) \le 1$, entonces puede demostrarse que el operador de Bellman (4) es contractivo (Buşoniu et al., 2010; Bertsekas, 2018). Con esta contractividad, la iteración aproximada de la función de valor (11) converge.

Por tanto, para calcular la función de valor con esta clase de regresores, la minimización (11) se reduce a una mera asignación:

$$\theta_i^{[k+1]} := \min_{u\in\mathbb{U}} \left(L(x_i,u) + \gamma V(f(x_i,u),\theta^{[k]}) \right), \forall x_i \in \mathcal{X}, \tag{15}$$

de tal modo que cuando la iteración de valor converge θ_i puede entenderse como el valor estimado de $V(x_i)$, como se ha comentado arriba.

Ejemplos de este tipo de regresores podrían ser: regresores aditivos binarios (Bertsekas and Tsitsiklis, 1996); codificación de "azulejos" (o *tile coding* por su terminología en inglés) (Sherstov and Stone, 2005), aunque en este caso un parámetro ajustable afecta localmente a unos pocos puntos del conjunto de datos; particiones borrosas (Buşoniu et al., 2010); particiones simpliciales (Grüne, 1997), o arbóreas (Munos and Moore, 2002).

En regresores interpolativos, la distribución que tienen los puntos de $\mathcal{X}$ es de vital importancia para poder aprovechar al máximo los recursos disponibles en problemas complejos. En este sentido, las distribuciones regulares suelen caracterizarse por ser poco eficientes, particularmente en aquellos lugares en los que el error de Bellman (7) sea mayor y cuya aproximación puede depender de la densidad del mallado regular.

Las propuestas en Buşoniu et al. (2010) plantean reducir complejidad generando $\mathcal{X}$ como el producto cartesiano de particiones unidimensionales con espaciado logarítmico, pero ello presupone que la zona cercana al origen es la que más granularidad requiere y, por otro lado, estas representaciones tienen como principal inconveniente el hecho de que la división en una región de una de las dimensiones del espacio de estados afecta a todas las "celdas" en el resto de dimensiones.

Las particiones simpliciales y arbóreas son capaces de dividir una región concreta del estado de forma localizada, por este motivo, este tipo de representación será la usada como parte de nuestra metodología.

2.3. Planteamiento del problema

Como se ha comentado anteriormente, la distribución de los regresores (o puntos característicos en regresores interpolativos) es crucial y las distribuciones regulares tienen como principal desventaja su excesivo número de parámetros a ajustar, lo que puede reducir su aplicabilidad en espacios de estado de mayor dimensionalidad. Las propuestas de particiones simpliciales y arbóreas que se han citado son un paso en la buena dirección, pero el aspecto clave entonces es determinar buenos criterios para aumentar o disminuir la granularidad del aproximador funcional en una región concreta.

Por tanto, el objetivo de este trabajo es proponer una metodología con un mallado adaptativo como veremos a continuación, a partir de la cual se pueda obtener una representación eficiente de la función de valor con el menor número de parámetros posible en función de los recursos disponibles, combinando criterios espaciales y de aproximación de funciones arbitrarias con otros específicamente concebidos para programación dinámica aproximada. En este sentido, compararemos con otras propuestas de la literatura.

3. Metodología de Programación Dinámica Aproximada Adaptativa

A continuación, se presenta una metodología que permite resolver el problema de programación dinámica aproximada.

La metodología propondrá combinar varios ingredientes:

a) Mallado simplicial irregular.

b) Criterio para añadir puntos al mallado basado en métodos para refinar dicho mallado (o deshacer refinados anteriores); métodos para generar puntos candidatos relevantes no contenidos en el mallado, para evaluar criterios sobre ellos; y un criterio para añadir nuevos vértices al mallado.

c) Criterio de simplificación (eliminación de vértices) del mallado.

Pasemos a analizar cada uno de estos elementos en secciones separadas.

3.1. *Mallados simpliciales irregulares*

En este trabajo se utilizará una representación con un mallado irregular que permite concentrar parámetros en aquellas regiones del estado que más se necesiten.

En concreto, proponemos usar un mallado simplicial dado por un conjunto de M puntos característicos $\mathcal{X}$. Se asumirá que el espacio de estados no terminales $\mathbb{X}$ es un poliedro y que los vértices de dicho poliedro forman parte de la lista de puntos $\mathcal{X}$.

Se operará con una partición simplicial de $\mathbb{X}$, que denominaremos "mallado", en el que un símplice $\mathcal{S}_i$ está definido por sus vértices $\mathcal{V}_i$:

$$\mathcal{V}_i := \{x_{C_i(1)}, x_{C_i(2)}, \ldots, x_{C_i(n+1)}\}, \tag{16}$$

siendo C_i un conjunto de índices (lista de conectividad del mallado) que indica qué vértices pertenecen a un símplice u otro a partir del conjunto de puntos totales $\mathcal{X}$ usados para construir dicho mallado. Entonces $\mathcal{S}_i = Co(\mathcal{V}_i)$, denotando "$Co$" la envoltura convexa de un conjunto finito de puntos. Los símplices forman una partición de modo que $\mathcal{S}_i$ y $\mathcal{S}_j$ tienen interiores disjuntos para $i \neq j$ y la unión de todos los símplices da lugar a $\mathbb{X}$.

Dado un punto de $\mathcal{X}$, este punto podrá ser vértice de varios símplices del mallado y también puede haber puntos compartidos en caras o aristas de varios símplices. El conjunto $\mathcal{L}(x)$ denotará la "región local" influenciada por el punto x, siendo:

$$\mathcal{L}(x) = \bigcup_{x \in \mathcal{S}_j} \mathcal{S}_j \tag{17}$$

En este caso, la propuesta es aproximar la función de valor como:

$$V(x, \theta) := \sum_{j=1}^{n+1} \mu_{i,j}(x)\theta_{C_i(j)}, i : x \in \mathcal{S}_i \tag{18}$$

donde i es el índice del único símplice al que pertenece x (si no está en una de las caras de las fronteras comunes), o del primero en la lista de símplices en ser encontrado en el caso de puntos en la frontera de los símplices.

En la expresión anterior, $\mu_{i,j}(x)$ son los valores de las coordenadas baricéntricas de un punto x, definidas como cada elemento de un vector $\mu_i(x)$ obtenido como:

$$\mu_i(x) = \underbrace{\begin{bmatrix} x_{C_i(1)} & x_{C_i(2)} & \ldots & x_{C_i(n+1)} \\ 1 & 1 & \ldots & 1 \end{bmatrix}^{-1}}_{D_i} \begin{bmatrix} x \\ 1 \end{bmatrix} \tag{19}$$

con $\mu_i(x) = [\mu_{i,1}(x), \mu_{i,2}(x), \ldots, \mu_{i,n+1}(x)]^T$. Las coordenadas baricéntricas cumplen, sí y solo sí $x \in \mathcal{S}_i$:

$$\sum_{j=1}^{n+1} \mu_{i,j}(x) = 1 \tag{20}$$

$$0 \leq \mu_{i,j}(x) \leq 1, \tag{21}$$

Por tanto, la operación de búsqueda del símplice a la que pertenece un punto x se determinará en función de si las coordenadas baricéntricas de x cumplen con las condiciones (20)-(21).

Dado que las coordenadas baricéntricas suman 1, cumplen las condiciones de contractividad discutidas en la sección 2.2 y, por tanto, la iteración de valor convergerá a un punto fijo que aproxima la función de valor óptima, independientemente de las condiciones iniciales de los parámetros ajustables.

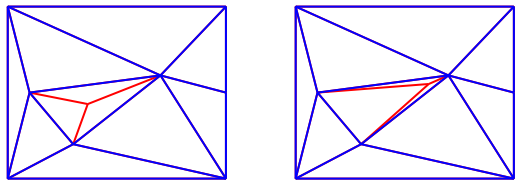
En concreto, una vez (15) ha convergido, el punto fijo verificará

$$\theta_i = \min_{u_j \in \mathbb{U}} \left(L(x_i, u_j) + \gamma V(f(x_i, u_j), \theta) \right), \forall x_i \in \mathcal{X}, \tag{22}$$

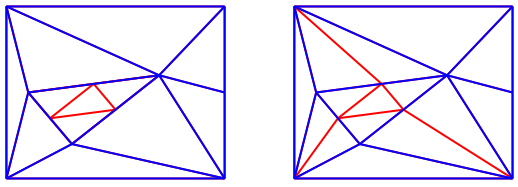
con lo que el error de diferencia temporal TDE (7) será cero en los puntos vértice del mallado. Obviamente, como el espacio de estados $\mathbb{X}$ es continuo, dicho error no será cero en el resto de puntos de $\mathbb{X}$.

3.2. *Criterio para añadir puntos al mallado*

El particionado de un símplice consiste en la subdivisión de un símplice en símplices más pequeños disjuntos del símplice original. La Figura 1 contempla diferentes posibilidades a la hora de dividir un símplice.



(a) Baricentro interior (b) Punto arbitrario interior



(c) Puntos medios (d) Puntos medios con continuidad

Figura 1: Métodos de particionado de un triángulo.

Aunque el criterio a discutir para decidir subdividir un símplice podrá depender de puntos arbitrarios en el interior del mismo, se ha decidido en este trabajo realizar la subdivisión por los puntos medios de las aristas del símplice, ya que esto permite obtener tamaños de símplices más equilibrados (Grüne, 1997), como se observa en la Figura 1(c).

Por cada división de un símplice, opcionalmente se pueden buscar los símplices con aristas adyacentes de forma que estos

también sean divididos convenientemente, según se representa en la Figura 1(d). Esto permite garantizar la continuidad de la interpolación de la función de valor resultante, dado que, por ejemplo en la Figura 1(c) los puntos de las aristas del triángulo dividido pertenecen a dos símplices (cerrados) y la interpolación sería diferente en el triángulo partido y en el adyacente no dividido, mientras que, en la Figura 1(d), eso no ocurre.

Tal y como ya se ha indicado, el TDE para cualquier punto dentro de la región $\mathbb{X}$ quedará definido por (7). Es decir, que el valor del TDE en los vértices de un símplice será igual a cero, mientras que para cualquier punto dentro de él podrá ser diferente a cero. Por tanto, es razonable establecer como criterio para ordenar los símplices de cara a su posterior división el máximo |TDE| en sus puntos, ya que afectarán al estimado de la función de valor de todas las trayectorias que pasen por ellos, al propagarse estos errores hacia atrás en el tiempo, en base a la ecuación de Bellman.

En Grüne (1997), se propone como criterio la subdivisión de aquellos símplices que dispongan de un mayor |TDE|, evaluados en los puntos intermedios de las aristas y su baricentro. En efecto, el baricentro es, heurísticamente, el punto que posiblemente tenga mayor |TDE| por estar más alejado de los vértices, cuyo valor de |TDE| es cero; un razonamiento similar justificaría la inclusión de los puntos medios de las aristas.

Sin embargo, en símplices lo suficientemente grandes, por ejemplo en iteraciones iniciales, es decir, cuando se disponen de pocos puntos para describir la aproximación de la función de valor, el valor del |TDE| en el baricentro o en puntos medios puede ser poco informativo, ya que la función de valor "ideal" a aproximar puede tener grandes variaciones dentro del símplice. Es decir, que la interpolación lineal entre sus vértices no va a poder ajustar bien dicha función de valor ideal; puede incluso tener discontinuidades, como se verá en los ejemplos.

Como, según lo discutido, el |TDE| puede ser grande en puntos que no necesariamente correspondan con el baricentro o los puntos intermedios, se propone que la evaluación del |TDE| deba hacerse en todo el símplice. De forma práctica, se pueden evaluar puntos distribuidos de forma aleatoria dentro de un símplice. Se generarán puntos aleatorios $\tilde{x}_i^{rnd}$ dentro del símplice i mediante:

$$\tilde{x}_i^{rnd} = \frac{\sum_{j=1}^{n+1} r_j \cdot x_{C_i(j)}}{\sum_{j=1}^{n+1} r_j}, \tag{23}$$

con $r_j \sim \mathcal{U}(0,1)$ una distribución uniforme de números aleatorios en el intervalo $[0,1]$. Esta forma de generar los puntos aleatorios ha sido intencionalmente escogida porque los puntos cercanos a los vértices tienen menor densidad de probabilidad, lo cual coincide con la idea de que el |TDE| cercano a los vértices será, quizás, más reducido. La distribución de probabilidad resultante se representa en la Figura 2.

Dado que una discontinuidad arbitraria en la función de valor nunca va a poder ser aproximada sin error mediante los interpoladores basados en mallado simplicial, el |TDE| no se reducirá al subdividir los símplices dentro de los cuales esté la discontinuidad. Esto ocasiona que, en problemas más complejos, el algoritmo propuesto por Grüne (1997) se bloquee, dividiendo sin fin zonas cercanas a las discontinuidades; este tipo de bloqueos ya fue reportado, por ejemplo, en Munos and Moore (2002).

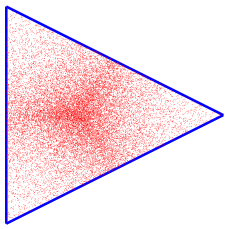


Figura 2: Distribución de puntos al azar en un símplice.

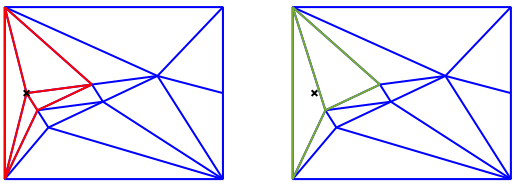
Es por ello, que proponemos usar, como variante del criterio, el producto $|\text{TDE}| \cdot A$, siendo A el área (o hipervolúmen en más dimensiones) del símplice. Se trata por tanto de un criterio en el que se consideran ideas "espaciales" (relacionadas con la "densidad de los puntos") y por otro lado, el error en diferencias temporales para integrarse correctamente en problemas de programación dinámica.

Para un símplice definido por sus vértices $\mathcal{V}_i$ (16), su volumen se calcula a partir de :

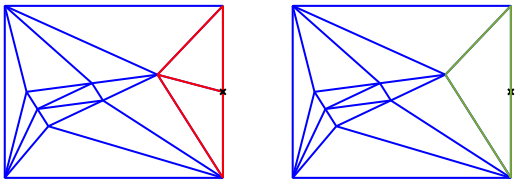
$$A_i = \frac{1}{(n+1)!} |D_i| \tag{24}$$

siendo D_i la matriz, sin invertir, definida en (19).

3.3. *Criterio para eliminar puntos*



(a) Región local no convexa (b) Retriangulación local



(c) Región local de punto en frontera (d) Retriangulación local

Figura 3: Métodos de desparticionado de un triángulo.

Como la función de valor estimada cambia conforme evoluciona el mallado, proponemos también un criterio para eliminar puntos que no modifiquen sustancialmente la función de valor aproximada en un determinado momento.

Para ello, deberemos asociar a cada punto $x_i \in \mathcal{X}$ del conjunto de puntos que definen el mallado, un valor numérico que indicará el "error que se cometería si se eliminara x_i". Vamos a detallar el cálculo de dicho índice de error.

Primero, consideraremos cada punto $x_i \in \mathcal{X}$ y calcularemos su región local $\mathcal{L}(x_i)$, según (17). Una vez obtenido $\mathcal{L}(x_i)$ para cada punto, se calcula una nueva triangulación de $\mathcal{L}(x_i)$, utilizando todos los vértices de los símplices $\mathcal{S}_j$ que la componen según la definición (17) antes mencionada, excepto el propio

x_i; denominaremos a dicho mallado local como $\mathcal{P}(x_i)$. Las figuras 3(a) y 3(c) representan en rojo las regiones $\mathcal{L}(x_i)$, donde el punto x_i se ha marcado con una X; las figuras 3(b) y 3(d) representan en verde las nuevas triangulaciones $\mathcal{P}(x_i)$ de la misma región, observándose que, como es de esperar, los puntos x_i ya no son vértices de un simplex del nuevo mallado.

Nótese que la región local podría no ser convexa, como en la Figura 3(a), de modo que deben especificarse adecuadamente las aristas o caras frontera para que la triangulación diferencie el interior del exterior de $\mathcal{L}(x_i)$, véase, por ejemplo, la documentación de `delaunay` en Matlab (Inc, 2021); el resultado se representa en la Figura 3(b). Las caras frontera son aquéllas que no contienen al punto x_i candidato a ser eliminado.

Nótese asimismo que, cuando se elimina un punto que está en la frontera del propio $\mathcal{L}(x_i)$, como en la Figura 3(c), calcular las caras frontera como las que no contienen a x_i dejaría el simplex "abierto". Se necesitan manipulaciones adicionales para conectar la cara faltante, detalles omitidos por brevedad.

Una vez el nuevo mallado $\mathcal{P}(x_i)$ ha sido construido, la estimación de la función de valor en x_i, $V^{reduc}(x_i)$, se calcularía utilizando las coordenadas baricéntricas de x_i en los nuevos símplices (19) para expresar la función de valor en x_i como una interpolación de los valores asignados a los vértices del símplice adecuado de $\mathcal{P}(x_i)$. Con ello, se evalúa el error de aproximación, $\mathcal{E}(x_i)$, como la diferencia (en valor absoluto) entre $V(x_i)$ proporcionado por la iteración de valor en el mallado original y $V^{reduc}(x_i)$. Calculando $\mathcal{E}(x_i)$ para todos los puntos del mallado $\mathcal{X}$ (excepto, obviamente aquéllos puntos vértice de la envoltura convexa del espacio de estados $\mathbb{X}$, que no pueden ser eliminados), permitirá ordenarlos para proceder a eliminar un pequeño número de ellos, según se discute a continuación.

3.4. Resumen de la metodología: algoritmo

A continuación se presentará un algoritmo que resume la propuesta metodológica, con pasos de inicialización de un mallado, de refinado del mismo (subdividiendo símplices según el criterio descrito en la Sección 3.2) y de simplificación del mismo (eliminando puntos poco relevantes y retriangulando, según discutido en la Sección 3.3).

Antes de detallar el algoritmo, conviene discutir algunas observaciones sobre el mallado inicial. Primero, como cada dimensión del estado tendrá un rango de valores diferentes, es aconsejable la normalización entre [0 1]. De ese modo, todos los componentes del espacio de estados tendrán igual importancia cuando se vayan a hacer cálculos con métricas como distancias o áreas, que dependen del escalado; también dependen del escalado las triangulaciones Delaunay para generar el mallado inicial, con lo que se propone que se realicen en este espacio normalizado. Otra cuestión a considerar es que, dado que inicialmente se desconoce la zona donde deben concentrarse los puntos, la metodología debería comenzar con un mallado inicial regular (poco denso), esto es, sería aconsejable, quizás, que los vértices iniciales estuvieran distribuidos uniformemente en $\mathbb{X}$. Esto permite obtener una primera estimación de la función de valor, y también una política inicial. A partir de esta primera estimación, el algoritmo evoluciona según el criterio indicado.

El algoritmo se detalla a la derecha. Como se comenta al final del mismo, el número de etapas de refinamiento y de simplificación puede elegirse a voluntad, según las prestaciones obtenidas, por ejemplo, mediante la simulación sobre un determinado conjunto de condiciones iniciales y evaluando el desempeño de dicha política. También podría comenzarse con un mallado inicial regular denso y ejecutar primero la fase de simplificación, según se considere conveniente (o probar varias opciones, a ver cuál funciona mejor en cada problema particular).

Algoritmo 1 Programación Dinámica Aproximada Adaptativa

Función PDAA($\mathcal{X}$,$\mathbb{U}$, q,e,Q,E)
entrada: $\mathcal{X}$: conjunto inicial de M datos; $\mathbb{U} := \{u_1, \ldots, u_m\}$: conjunto (finito) de acciones de control; q: número de símplices a partir por iteración; e: número de puntos a eliminar del mallado por iteración; Q número máximo de puntos admisibles, E número total de puntos a eliminar.

1: Obtener los sucesores de los puntos del mallado $\mathcal{F}_i := \{f(x_1, u_1), f(x_1, u_2), \ldots, f(x_1, u_m)\}\ \forall x_i \in \mathcal{X}$.
2: Obtener un mallado inicial de $\mathcal{X}$ (p.e.: Delaunay), que contendrá un número S de símplices con vértices y lista de conectividad asociados, según (16).

[Refinado del mallado]
3: **mientras** $M < Q$ **hacer**
4: Ejecutar iteración de valor hasta que (15) converja.
5: Calcular la política óptima aproximada (13), $\pi(x_i, \theta^*)$ para todos los x_i de $\mathcal{X}$.
6: Para cada símplice $\mathcal{S}_i$, calcular $|\mathrm{TDE}_i|$ como el máximo de los valores absolutos de los errores (7), usando la política del paso anterior. Los puntos a evaluar son: el baricentro, los puntos medios de las aristas y un cierto número de puntos aleatorios generados dentro de cada símplice.
7: Calcular el volumen A_i (24) de cada símplice $\mathcal{S}_i$.
8: Ordenar los símplices según el criterio $|\mathrm{TDE}_i| \cdot A_i$ de mayor a menor y seleccionar los q primeros símplices para ser subdivididos.
9: Dividir los símplices (eligiendo, a conveniencia, alguno de los métodos de la Figura 1). Añadir los nuevos vértices generados a $\mathcal{X}$. Calcular sus sucesores (para futuras ejecuciones del paso 4) y actualizar el número de puntos M.
10: **fin mientras**

[Simplificación del mallado]
11: **mientras** $M > E$ **hacer**
12: Ejecutar iteración de valor hasta que (15) converja.
13: Para cada punto $x_i \in \mathcal{X}$, calcular el conjunto de regiones locales $\mathcal{L}(x_i)$ y calcular el error de aproximación, $\mathcal{E}(x_i)$, en caso de quitar x_i.
14: Ordenar los puntos según el criterio $\mathcal{E}_i$ de menor a mayor y eliminar los q primeros puntos. Retriangular la región local, tras eliminar los puntos, según se indica en la Sección 3.3 y actualizar el número de puntos M.
15: **fin mientras**
16: Repetir pasos 3-10 (refinado del mallado) y 11-15 (simplificación del mallado), según convenga.

3.5. Discusión y análisis comparativo

Aunque en la introducción y preliminares se han citado múltiples opciones de programación dinámica aproximada, incluyendo trabajos previos en el tema de los autores, la discusión y el análisis comparativo se hará solo con las técnicas de mallados simpliciales/arbóreos adaptativos de Grüne (1997) y Mu-

nos and Moore (2002). Las contribuciones son las seminales, que originaron las ideas básicas de este enfoque, pero al menos en la búsqueda realizada por los autores, no se han encontrado propuestas posteriores con nuevas ideas prometedoras con las que comparar; los resultados recientes toman direcciones más relacionadas con las redes neuronales profundas o similar. Tampoco compararemos con mallados no adaptativos, que a veces resuelven con éxito problemas sencillos con, por ejemplo, espaciado logarítmico entre nodos (Busoniu et al., 2010) pero que requieren un conocimiento *a priori* sobre las regiones del espacio de estado donde se necesita mallar con mayor densidad.

El principal inconveniente de Grüne (1997) es que, al basarse únicamente en el |TDE|, en las discontinuidades de la función de valor óptima (o en zonas de rápida variación de la misma), el |TDE| no decrece cuando se añaden puntos: los aproximadores funcionales universales no pueden aproximar uniformemente discontinuidades, esto es, el máximo del error de aproximación no decrece a menos que los regresores se alineen "perfectamente" con la discontinuidad.

Sin embargo, los aproximadores funcionales sí que son capaces de reducir la "integral" del error de aproximación sobre el espacio de estados al refinarse y aumentar el número de parámetros ajustables. Esta idea de la "integral" del error es lo que, aproximadamente, en nuestros mallados simpliciales, inspira el criterio de "error × volumen", esto es |TDE| · A en este trabajo, en vez del puro error de Grüne (1997).

Nótese que Munos and Moore (2002) también se percataron del problema que Grüne (1997) tenía y planteó muchas propuestas alternativas. La que los autores identificaron como la mejor fue la que hemos comparado en la sección de ejemplos de este trabajo. Dicha política multiplicaba un criterio de "influencia de las discontinuidades en la acción de control" por otro criterio de "desviación típica acumulada" (estimado de la incertidumbre en la aproximación de la función de valor), que nombraron como Std·Inf. A priori, sí que funciona mejor que Grüne (1997), en sus simulaciones y en nuestros ejemplos, pero hemos apreciado que se centra excesivamente en dividir las zonas con discontinuidades de la acción de control incluso aunque la función de valor en ellas sea "suave". En el ejemplo que proponemos posteriormente, se observa que su mallado resulta menos eficiente que nuestra propuesta, quedándose también estancado.

Nótese, por último, que en las discontinuidades, los puntos tenderán a acumularse de forma densa, hasta que la resolución sea lo suficientemente adecuada, lo que hace que se pueda prescindir de muchos de los puntos que han sido añadidos en la iteraciones previas. Es por ello que es necesario implementar un mecanismo de simplificación del mallado que permita aprovechar mejor los recursos disponibles.

4. Ejemplo

Sea $x = [p\ v]^T$ el estado del conocido problema Montaña-Carrito (Bertsekas and Tsitsiklis, 1996; Munos and Moore, 2002) (como se aprecia en la Figura (5)), siendo p la posición del carrito y v su velocidad, con la siguiente dinámica $\dot{p} = v$ y $\dot{v} = -g\sin(ap + b) + u$, siendo g la magnitud de la gravedad, $a = \frac{3\pi}{80}\text{m}^{-1}$ y $b = \frac{\pi}{80}$ definen una curva senoidal de la montaña y u es la acción de control (aceleración) expresada en m/s^2. El sistema será discretizado (Euler) a período $T = 0{,}01$ s.

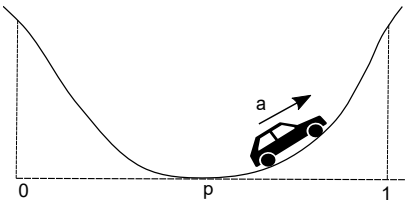


Figura 5: Figura conceptual del problema Montaña-Carrito.

El estado está acotado al rango $x \in \mathbb{X} = [-10, 10] \times [-15, 15]$m×m/s, mientras que la acción de control se asume discreta $u \in \mathbb{U} = \{-4, -\frac{8}{3}, -\frac{4}{3}, 0, \frac{4}{3}, \frac{8}{3}, 4\}$m/s^2. La dificultad del problema de control radica en el hecho de que la aceleración no es suficiente para vencer la gravedad en todos los puntos de la "montaña" y por tanto debe buscar políticas donde se aproveche la "inercia" incluso impulsándose en sentido contrario en los instantes iniciales.

El objetivo es encontrar la política $\pi^*(x)$ que minimiza (5), con coste inmediato $L(x, u) = 6$ y factor de descuento $\gamma = 0{,}999$. La penalización o coste terminal es:

$$V_{\mathbb{T}}(x) = \begin{cases} 4000 & \text{si } |v| > 15 \\ 4000 & \text{si } p < -10 \\ 0 & \text{si } p > 10 \end{cases} \tag{25}$$

de modo que si se abandona el rectángulo en el espacio de estados $\mathbb{X}$ por velocidad excesiva o por posición fuera de límite negativo, hay un coste de 4000, y si se abandona el espacio de estados con velocidad dentro de límites por el lado derecho entonces no hay coste (ese sería, por tanto, el objetivo de control: alcanzar el extremo derecho en el menor tiempo posible). Como se ha discutido anteriormente, el espacio de estados, el modelo y las funciones de coste han sido normalizadas para operar con la región de estados $[0, 1] \times [0, 1]$.

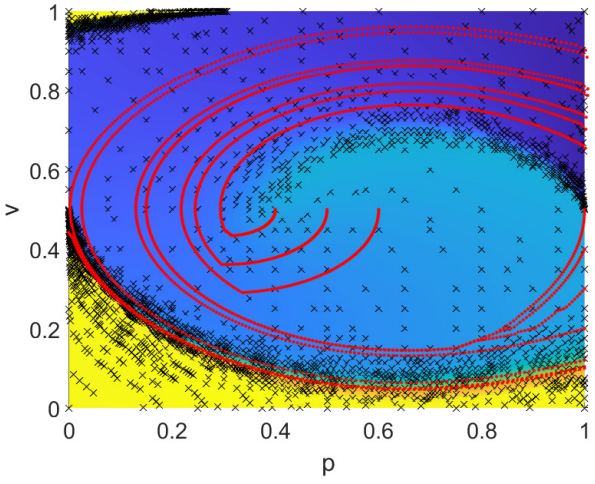
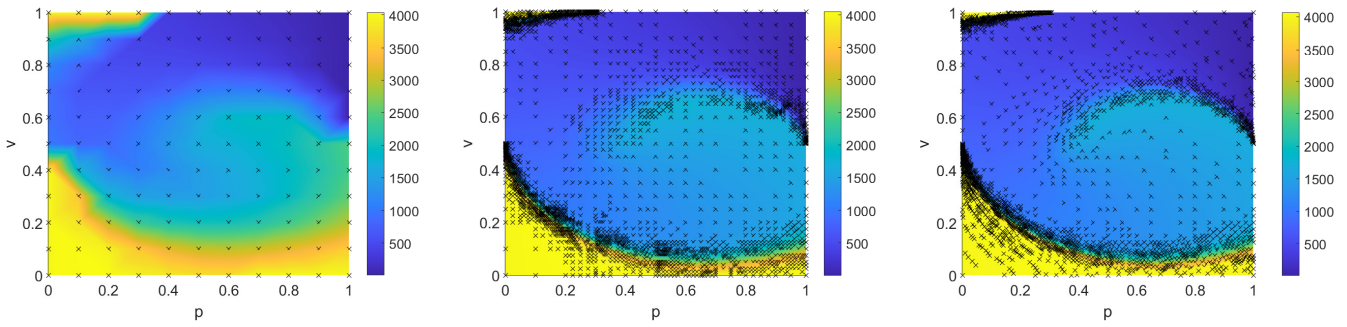


Figura 6: Diagrama de fase de las trayectorias optimas desde diferentes estados iniciales.

Pasemos a discutir los resultados del algoritmo propuesto. Nótese que para el refinamiento del mallado, en el paso 9 del al-



(a) Mallado inicial con 121 puntos (b) Con 2601 puntos añadidos (c) Con limpieza hasta 1000 puntos y añadido hasta 2601 puntos

Figura 4: Función de valor y puntos vértice del mallado usados para la aproximación.

goritmo, se ha usado una subdivisión de símplices por los puntos medios de las aristas con continuidad, esto es, la representada en la Figura 1(d).

La Figura 4(a) representa el mallado inicial del algoritmo (regular 11×11, $M = 121$). Seleccionando $Q = 2601$, correspondiente al número de puntos de un mallado 51×51 regular con el que luego se comparará, el refinado del mallado situa los puntos como se presenta en la Figura 4(b). Tras ejecutar una simplificación del mallado hasta $E = 1000$ puntos y un nuevo refinamiento hasta 2601 puntos, se obtiene la distribución de puntos mostrada en la Figura 4(c). Se observa que, conforme a lo esperado intuitivamente, los puntos se reparten en las zonas más importantes, cercanas a las regiones con cambios bruscos de la función de valor.

En la figura 6, se puede apreciar las trayectorias en el plano de fase del bucle cerrado con la política óptima aproximada con la distribución de puntos final, de la Figura 4(c). En concreto, la figura muestra las trayectorias que comienzan en los puntos normalizados $[0{,}4, 0{,}5]^T$, $[0{,}5, 0{,}5]^T$, $[0{,}6, 0{,}5]^T$, $[1, 0{,}5]^T$, $[1, 0{,}4]^T$, $[1, 0{,}3]^T$, $[1, 0{,}2]^T$, $[1, 0{,}1]^T$, $[1, 0{,}12]^T$ y $[1, 0{,}104]^T$. Estos puntos han sido seleccionados de tal forma que el carrito comience en la zona inferior de la montaña y también en la zona próxima a la salida, pero con velocidad negativa. Se observa que la política aprendida necesita imprimir aceleraciones negativas para desplazar hacia la izquierda hasta que dispone de la energía necesaria para, acelerando hacia la derecha, salir por el lado derecho y entrar a la región terminal de coste cero. Las regiones "amarillas" de la figura (abajo y parte superior izquierda), con coste superior a 4000 indican el subconjunto del espacio de estados para el que no existe una solución óptima que pueda alcanzar la zona objetivo (lado derecho), por comenzar con una velocidad excesiva que la limitada acción de control no puede reducir antes de que abandonen $\mathbb{X}$ por una zona severamente penalizada.

La Figura 7 muestra un ejemplo de la evolución temporal de las trayectorias para el caso en el que el carrito comienza desde el punto $[0{,}5, 0{,}5]^T$ del estado. También muestra la acción de control aplicada que ha generado un coste acumulado de 1505,06, comparado al coste obtenido del controlador de la primera fase en la que solo se añaden puntos, con un coste de 1558,64.

Con respecto a la elección del factor de descuento comparado con $\gamma = 1$ que sería la solución de "tiempo mínimo" en abandonar $\mathbb{X}$, en nuestro caso con $\gamma = 0{,}999$, el coste de una trayectoria que no abandonara $\mathbb{X}$, daría un coste de $V_{MAX} := \frac{\text{máx } L}{1-\gamma} = 6000$ (suma prograsión geométrica). El coste estimado siempre está por debajo de 4100 y la zona azul de la Figura 4(b) está por debajo de 2000 (se alcanza el objetivo lado derecho). Esto significa que sí se intenta escapar rápidamente de $\mathbb{X}$ y que las soluciones obtenidas son las razonablemente esperadas (Figura 6), por lo que la elección de γ parece adecuada.

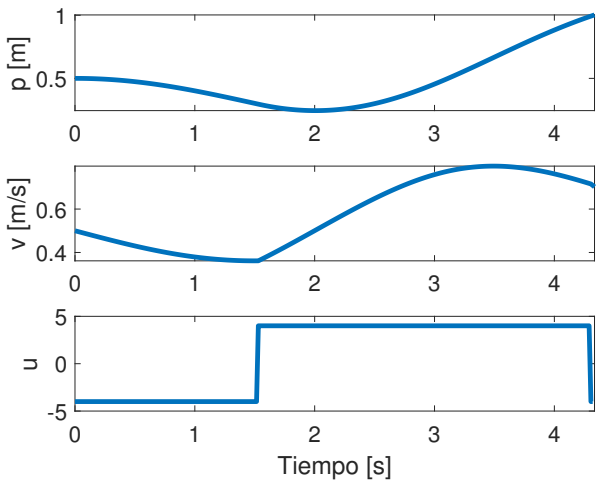


Figura 7: Trayectorias temporales del estado y la acción de control partiendo del punto $x = 0{,}5$ y $v = 0{,}5$.

4.1. *Análisis comparativo*

Con el propósito de comparar los resultados obtenidos con distintos mallados, se evaluará el desempeño de un controlador mediante la evaluación del coste acumulado (2) de todas las trayectorias **simuladas** desde condiciones iniciales situadas en puntos según un espaciado regular de 51x51 puntos del espacio $\mathbb{X}$. Estos puntos representan, por tanto, un conjunto de datos de validación y son fijos para todos los métodos, sin necesidad de coincidir con los puntos característicos (vértices) de las triangulaciones simpliciales adaptativas del aproximador de la función de valor subyacente. El desempeño acumulado final es la suma de los costes reales (2) de todas esas 2601 simulaciones partiendo desde cada uno de los estados iniciales. Cuandas más trayectorias consigan llevarse al objetivo (lado derecho) rápidamente, mejor será dicho desempeño acumulado; estrategias

subóptimas podrían hacer que algunas trayectorias fracasen por salir por lado izquierdo o superar límite de velocidad, según explicitado en (25).

Como solución de referencia, se ha sintonizado con iteración de valor un controlador basado en un mallado regular (Delaunay) con ese mismo espaciado de 51x51, esto es, en este caso los 2601 puntos del mallado son coincidentes con las condiciones iniciales simuladas. El desempeño de otros algoritmos de mallado se ha comparado con dicha solución y lo que se representa en la Figura 8 como la "*diferencia de desempeño*" es la diferencia entre el desempeño acumulado final (2601 simulaciones) y dicho desempeño acumulado de la solución de referencia. Un valor de cero en el eje de ordenadas indica, por tanto, unas prestaciones iguales al mallado regular de 51×51 puntos. Obviamente, el objetivo es conseguir prestaciones similares o mejores (por debajo de cero en la referida figura) con un menor número de puntos del mallado.

Los métodos que se han comparado en la Figura 8 son:

1. Mallado con espaciado regular (la solución de referencia es el mallado de 2601 puntos, con lo cual el valor en la figura es cero al llegar al máximo de puntos), representado en color azul.
2. Método propuesto por Grüne (1997), en verde.
3. Método Std·Inf propuesto por Munos and Moore (2002); aunque originalmente está propuesto con celdas cúbicas, se ha adaptado a las triangulaciones utilizadas aquí, para que los criterios de adición sean equiparables, en color negro.
4. El criterio propuesto en este trabajo, error en diferencia temporal por área, $|\text{TDE}| \cdot \text{A}$, solo añadiendo puntos al mallado (ejecutando solo el refinado del mallado en el algoritmo), en color rojo.
5. El algoritmo completo, ejecutando el cuarto método con eliminación posterior de puntos ($E = 1000$) y una segunda ejecución del algoritmo volviendo a añadir puntos hasta $Q = 2601$, dibujando dicha segunda fase de refinamiento en color magenta.

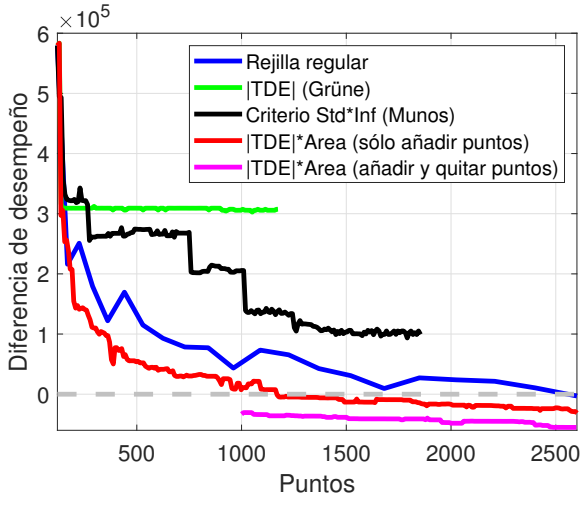


Figura 8: Comparativa entre métodos con un mallado inicial de 11x11. La linea gris a rayas denota el nivel de referencia de prestaciones (mallado regular 51×51) con el que se compara, cota cero del eje de ordenadas.

Tal y como puede apreciarse en la Figura 8 el desempeño de nuestro método es capaz de encontrar soluciones que mejoran a la solución de referencia con un número significativamente menor de puntos, batiendo al resto de opciones. En concreto, la solución de referencia es mejorada con alrededor de 1200 puntos (por este motivo la diferencia baja de cero): obtenemos un resultado equiparable al caso del grid regular con 51x51 puntos, pero solo con un 45 % de los vértices y que se ha podido mejorar todavía más al añadir vértices adicionales al mallado.

El método original propuesto por Grüne se queda claramente estancado por culpa de la discontinuidad, ya que el |TDE| máximo siempre aparece en las proximidades de las discontinuidades. El método propuesto por Munos, consigue mejorar al método propuesto por Grüne, pero no es capaz de mejorar la implementación trivial del mallado regular y también se queda estancado por encima del desempeño de referencia. Ambos métodos, provocan errores de triangulación por acumular demasiados puntos cercanos, sobrepasando los límites de precisión numérica admisibles, y por este motivo no son capaces de alcanzar los 2601 puntos.

5. Conclusiones

En este artículo, se ha propuesto un método para resolver problemas de Programación Dinámica Aproximada basado en la idea de la obtención de mallados simpliciales adaptativos, en los que se ha definido, por un lado, un criterio espacio-temporal para añadir puntos al mallado basándose en el error en diferencias temporales y la "densidad" de los puntos (volúmen del símplice al que pertenecen). Por otro lado, se ha definido un criterio para eliminar puntos que está basado en el error de aproximación que se produce tras la hipotética eliminación de un punto del mallado.

En el algoritmo propuesto en este artículo, se han combinado ambos criterios de forma que en una primera pasada se consigue realizar un refinado del mallado a partir de un mallado inicial poco denso, para posteriormente realizar una simplificación del mismo. Obviamente, estos dos pasos pueden repetirse de forma arbitraria y podría haberse planteado un mallado inicial regular denso y ejecutar primero la fase de simplificación.

Se ha realizado un análisis comparativo con otras opciones en literatura, mostrando que el método propuesto es capaz de obtener una aproximación de la función de valor eficiente, con mejores prestaciones para el mismo número de vértices que otros métodos comparados.

Agradecimientos

Este artículo ha sido financiado por la Agencia Española de Investigación mediante el proyecto del Plan Nacional PID2020-116585GB-I00.

Referencias

Albertos, P., Sala, A., 2006. Multivariable control systems: an engineering approach. Springer, London, U.K.

Allgower, F., Zheng, A., 2012. Nonlinear model predictive control.

Antos, A., Szepesvári, C., Munos, R., 2008. Learning near optimal policies with bellman-residual minimization based fitted policy iteration and a single sample path. Machine Learning 71 (1), 89–129.

Ariño, C., Pérez, E., Querol, A., Sala, A., 2014. Model predictive control for discrete fuzzy systems via iterative quadratic programming. In: Fuzzy Systems (FUZZ-IEEE), 2014 IEEE International Conference on. IEEE, pp. 2288–2293.

Ariño, C., Pérez, E., Sala, A., 2010. Guaranteed cost control analysis and iterative design for constrained takagi–sugeno systems. Engineering Applications of Artificial Intelligence 23 (8), 1420–1427.

Armesto, L., Girbés, V., Sala, A., Zima, M., Šmídl, V., 2015. Duality-based nonlinear quadratic control: Application to mobile robot trajectory-following. IEEE Transactions on Control Systems Technology 23 (4), 1494–1504.

Athans, M., Falb, P. L., 2013. Optimal control: an introduction to the theory and its applications. Courier Corporation.

Bertsekas, D. P., 2018. Abstract dynamic programming. Athena Scientific.

Bertsekas, D. P., Tsitsiklis, J. N., 1996. Neuro-Dynamic Programming. Athena Scientific, Belmont, MA, USA.

Busoniu, L., Babuska, R., De Schutter, B., Ernst, D., 2010. Reinforcement learning and dynamic programming using function approximators. CRC press, Boca Raton, FL, USA.

Buşoniu, L., Ernst, D., De Schutter, B., Babuška, R., 2010. Approximate dynamic programming with a fuzzy parameterization. Automatica 46 (5), 804–814.

Camacho, E. F., Bordons, C., 2010. Control predictivo: Pasado, presente y futuro. Revista Iberoamericana de Automática e Informática Industrial 1 (3), 5–28.

De Farias, D. P., Van Roy, B., 2003. The linear programming approach to approximate dynamic programming. Operations research 51 (6), 850–865.

Deisenroth, M. P., Neumann, G., Peters, J., et al., 2013. A survey on policy search for robotics. Foundations and Trends in Robotics 2 (1–2), 1–142.

Díaz, H., Armesto, L., Sala, A., 2019. Metodología de programación dinámica aproximada para control óptimo basada en datos. Revista Iberoamericana de Automática e Informática industrial 16 (3), 273–283.

Díaz, H., Armesto, L., Sala, A., 3 2020. Fitted Q-function control methodology based on takagi-sugeno systems. IEEE Transactions on Control Systems Technology 28 (2), 477–488.

Díaz, H., Sala, A., Armesto, L., 2020. A linear programming methodology for approximate dynamic programming. International Journal of Applied Mathematics and Computer Science 30 (2).

Duarte-Mermoud, M., Milla, F., 2018. Estabilizador de sistemas de potencia usando control predictivo basado en modelo. Revista Iberoamericana de Automática e Informática industrial.

Fairbank, M., Alonso, E., 6 2012. The divergence of reinforcement learning algorithms with value-iteration and function approximation. In: The 2012 International Joint Conference on Neural Networks (IJCNN). pp. 1–8. DOI: 10.1109/IJCNN.2012.6252792

Grüne, L., 1997. An adaptive grid scheme for the discrete hamilton-jacobi-bellman equation. Numerische Mathematik 75, 319–337. DOI: 10.1007/s002110050241

Hornik, K., Stinchcombe, M., White, H., 1989. Multilayer feedforward networks are universal approximators. Neural Networks 2 (5), 359 – 366.

Inc, T. M., 2021. Matlab delaunay documentation. URL: https://www.mathworks.com/help/matlab/ref/delaunay.html

Lewis, F. L., Liu, D., 2013. Reinforcement learning and approximate dynamic programming for feedback control. Wiley, Hoboken, NJ, USA.

Lewis, F. L., Vrabie, D., 2009. Reinforcement learning and adaptive dynamic programming for feedback control. Circuits and Systems Magazine, IEEE 9 (3), 32–50.

Li, W., Todorov, E., 2007. Iterative linearization methods for approximately optimal control and estimation of non-linear stochastic system. International Journal of Control 80 (9), 1439–1453.

Liberzon, D., 2011. Calculus of variations and optimal control theory: a concise introduction. Princeton university press.

Munos, R., Moore, A., 2002. Variable resolution discretization in optimal control. Machine learning 49 (2-3), 291–323.

Rubio, F. R., Navas, S. J., Ollero, P., Lemos, J. M., Ortega, M. G., 2018. Control óptimo aplicado a campos de colectores solares distribuidos. Revista Iberoamericana de Automática e Informática industrial.

Santos, M., 2011. Un enfoque aplicado del control inteligente. Revista Iberoamericana de Automática e Informática Industrial RIAI 8 (4), 283–296.

Sherstov, A. A., Stone, P., 2005. Function approximation via tile coding: Automating parameter choice. In: International Symposium on Abstraction, Reformulation, and Approximation. Springer, pp. 194–205.

Sutton, R. S., Barto, A. G., 1998. Reinforcement learning: An introduction. Vol. 1. MIT press Cambridge.

Ziogou, C., Papadopoulou, S., Georgiadis, M. C., Voutetakis, S., 2013. On-line nonlinear model predictive control of a pem fuel cell system. Journal of Process Control 23 (4), 483–492.