ISSN 1688-2806

Universidad de la República
Facultad de Ingeniería

Optimización del Ruteo en Redes Sobrepuestas con Sistemas de Decisión en base a Medidas

Tesis presentada a la Facultad de Ingeniería de la Universidad de la República por

Martín Randall Carlevaro

en cumplimiento parcial de los requerimientos para la obtención del título de Magister en Ingeniería Eléctrica.

Director de Tesis
Dr. Ing. Pablo Belzarena ..... Universidad de la República

Tribunal
Dr. Ing. Federico La Rocca ..... Universidad de la República
Dr. Ing. Pedro Casas ..... Austrian Institute of Technology
Dr. Ing. Alberto Castro ..... Universidad de la República

Director Académico
Dr. Ing. Pablo Belzarena ..... Universidad de la República

Montevideo
martes 15 septiembre, 2020

Optimización del Ruteo en Redes Sobrepuestas con Sistemas de Decisión en base a Medidas, Martín Randall Carlevaro.

ISSN 1688-2806

Esta tesis fue preparada en LaTeX usando la clase iietesis (v1.1).
Contiene un total de 135 páginas.
Compilada el martes 15 septiembre, 2020.
`http://iie.fing.edu.uy/`

Es imposible calcular con precisión los hechos que son fruto del azar.

Tucídides

Esta página ha sido intencionalmente dejada en blanco.

Agradecimientos

Esta tesis fue realizada gracias al apoyo de una beca de Maestría, otorgada por la ANII, y financiada con fondos públicos, por lo que va un primer agradecimiento a la institución y un segundo agradecimiento a los contribuyentes.

Fue fundamental contar con la guía permanente y paciente de Pablo Belzarena, a quien se agradece especialmente.

Finalmente agradezco al Instituto de Ingeniería Eléctrica de la Universidad de la República, que promovió el desarrollo de este trabajo, alentando la dedicación y postulación a la Maestría, dictando los cursos realizados, y contando con los valiosos consejos de sus integrantes en cuantiosas ocasiones.

Esta página ha sido intencionalmente dejada en blanco.

A mis seres queridos

Y sin agregar latencia,
le dedico este trabajo,
mas hoy le advierto de cuajo:
no encontrará gran sapiencia,
aunque si tiene paciencia,
léalo, no se me espante!
Consejo espero expectante,
de familia, amigos, ella;
lo recibo sin querella,
mientras dé para adelante.

Esta página ha sido intencionalmente dejada en blanco.

Resumen

El tema de esta Tesis es el diseño de sistemas de decisión recurrentes en el tiempo y basados en medidas. El objetivo del tomador de decisiones es optimizar alguna función de desempeño, minimizando el costo de las mediciones y de la incertidumbre asociada al sistema. En particular, se trabaja sobre una aplicación al ruteo en redes sobrepuestas con calidad de servicio.

Las redes sobrepuestas son redes virtuales compuestas por nodos pertenecientes a diferentes redes (subyacentes), conectados entre sí por enlaces virtuales. En general, la política de ruteo entre las redes subyacentes suele no ser óptima, por lo que puede convenir establecer políticas propias.

En esta aplicación se busca elegir la mejor ruta en cuanto a algún parámetro de calidad de servicio. Para decidir cuál es la mejor de las rutas posibles, es necesario medir el parámetro de calidad en cuestión. Estas mediciones habitualmente tienen costos asociados, por ejemplo, la interferencia que se genera para realizar la medida en cada ruta, que impacta en el tráfico de los usuarios. Lo ideal sería no tener que medir en todos los tiempos de decisión y poder predecir cuál es la calidad de servicio en función de las medidas anteriores. Sin embargo, el "no medir" genera una incertidumbre en la calidad de servicio y es posible que se elija una ruta diferente de la óptima en el momento de decisión, por lo que también la decisión de "no medir" tiene un costo asociado: el de la calidad perdida por no escoger la ruta óptima. El objetivo es decidir en cada tiempo de decisión cuáles rutas medir y qué camino elegir, minimizando el costo total acumulado en el tiempo.

En un primer abordaje se modela el problema como un Proceso de Decisión Markoviano, se prueban algoritmos de programación dinámica y se propone una solución innovadora: la aproximación por horizonte errante. Luego se liberan las asunciones sobre modelos y se propone una formulación para la utilización de técnicas de aprendizaje supervisado, para lo que se emplean clasificadores bien conocidos como son los árboles de decisión.

El método de horizonte errante alcanza resultados casi-óptimos, que permiten reducir el costo de medida manteniendo el menor tiempo de ida y vuelta posible. El algoritmo de aprendizaje supervisado logra un rendimiento comparable, con otras propiedades como robustez frente a escenarios no-markovianos y un menor tiempo de procesamiento.

Esta página ha sido intencionalmente dejada en blanco.

Tabla de contenidos

Capítulo 1

Introducción

1.1. Motivación

Ningún sabio dijo nunca que cuando uno necesita algo es cuando lo valora realmente.

Es justo en tiempos de pandemia y distanciamiento social que toca exponer en esta Tesis porqué es importante buscar formas de mejorar el funcionamiento de Internet. Desde hace unos meses, la casi totalidad de las comunicaciones humanas se realizan mediante Internet. Si bien el autor espera sinceramente que la red social no reemplace al encuentro, es innegable que desde la educación hasta la salud, desde las transacciones financieras hasta los procesos industriales, y así se puede seguir, la "red de redes" ha sacudido las formas de relacionamiento humano tomando un rol protagónico.

Sin embargo, la estructura de Internet, que ya tiene unos 25 años de uso masivo, no se encuentra exenta de problemas. Una arquitectura que ha crecido tanto conlleva a lo que se conoce como la "osificación": la dificultad de cambiar estructuras grandes ya en funcionamiento para adaptarlas a las necesidades actuales. Al mismo tiempo, tanto el tráfico que demanda cada usuario como la cantidad de usuarios se han multiplicado, lo que también genera tensiones sobre una estructura pensada para tareas más específicas que su uso actual (y en sus inicios directamente diferentes). Es difícil pensar que quien diseñara los protocolos de Internet hace tres décadas previera la transmisión de la copa Libertadores por *Facebook*, con la consecuente exigencia en la calidad de experiencia que demanda semejante audiencia.

Dentro de las múltiples tensiones que existen entre esta creciente demanda y los recursos disponibles, hay una en particular en la que se hará foco. Para esto se debe comenzar por explicar que lo popularmente conocido como Internet, es una colección de Sistemas Autónomos (AS), administrados por proveedores de servicios de Internet (ISP, por sus siglas en inglés). Un ISP se encarga de la interconexión de las redes que alberga con el resto de los AS y sus redes, es decir con el resto del mundo. Cómo se realiza esta interconexión, y qué camino debe seguirse para ir desde un AS a otro se establece utilizando un protocolo llamado Border Gateway

Protocol (BGP, definido en la RFC 4271).

Son pocos los casos en que Internet tiene un único protocolo que se ha instalado de forma tan definitiva: no en vano se dice popularmente que *"BGP is the glue of the Internet"*[1]. Si BGP propusiera siempre la mejor ruta según algún criterio deseado por el usuario, no habría problemas a nivel del ruteo interdominio. El asunto es que BGP elige la ruta según la cantidad de saltos entre AS o según criterios económicos del ISP, y esta elección suele no ser la ruta más rápida, con menos pérdidas, etcétera.

¿Cúal es el problema al que se enfrenta BGP? El rol de BGP es la interconexión de, a la fecha de consulta[2], 68737 Sistemas Autónomos. Esto exige que sea un protocolo más escalable que eficiente en la búsqueda de la ruta más rápida. BGP sigue la siguiente secuencia para determinar la ruta entre dos nodos: 1) si existe una política de ruteo al respecto se sigue esa (por ejemplo por acuerdos económicos del ISP con otros ISPs), 2) de no existir, buscar la ruta de mínima cantidad de saltos entre los nodos, 3) finalmente en caso de empate utiliza una serie de prioridades propias o heredadas de otros protocolos de los cuales recibe los paquetes. El utilizar la métrica "cantidad de saltos", es decir ser un protocolo de los llamados "vector-distancia", no garantiza que la ruta elegida sea la más rápida. Por otro lado los cambios de rutas en BGP demoran tiempo en propagarse, lo que puede llevar a fallas y a la demora en la resolución de estas fallas. Esto se da porque BGP filtra o desestima detalles sobre la topología de la red para evitar oscilaciones que pudieran causar daños mayores.

Justamente ahí radica parte de la dificultad: modificar BGP para ajustarlo a las necesidades actuales requeriría de la colaboración -o por lo menos de la coordinación- de los proveedores de servicio, que son empresas en competencia. Los proveedores de servicio ocupan un lugar estratégico: poseen un recurso cuyo valor se ha incrementado muchísimo, y si bien buscan por competitividad ofrecer mejores servicios, no parecen tener la mira puesta en el beneficio colectivo que reportaría a los usuarios del mundo un Internet más ágil y mejor coordinado. Al mismo tiempo, no hay un oligopolio claro que permita el acuerdo entre unas pocas empresas involucradas, si bien no se logró dar con un número concreto de cuántos ISPs hay en el mundo, son del orden de los miles (y justamente con poca información disponible). Muchas veces son compañías locales, nacionales o regionales, y establecen acuerdos económicos entre sí que impactan directamente en las rutas interdominio que se establecen. Varias discusiones se han dado entre ISPs, entre usuarios e ISPs, y entre otras empresas e ISPs. En particular recuérdese la reciente controversia sobre la neutralidad de la red, o el debate más cercano de si la infraestructura de fibra óptica que instaló ANTEL deberían usarla el resto de los proveedores en competencia[3].

[1] "BGP es el pegamento que mantiene unido Internet".

[2] Se consultó la página https://www.cidr-report.org/as2.0/ el 01/07/20.

[3] Recordar también los casos de espionaje, utilización con fines comerciales de datos privados de usuarios y el bloqueo de usuarios anónimos https://venturebeat.com/2014/05/15/facebook-akamai-respond-to-nsa-slides-alleging-massive-cdn-vulnerability/ y https://nakedsecurity.sophos.com/2016/02/29/tor-

Por otro lado, la implementación de una nueva arquitectura exigiría una adecuación y uniformización de los equipos y del software utilizado. Esto es, no sólo los ISP deben ponerse de acuerdo, también los proveedores de hardware y software.

Finalmente, modificar protocolos que son tan centrales al funcionamiento de una arquitectura tan grande como Internet es un riesgo grande. Existen varios trabajos, como se verá en los antecedentes, que realizan propuestas de cambios parciales o totales de la arquitectura interdominio, pero muy pocas o ninguna se ha implementado. Y pese a los vaticinios de un futuro próximo lleno de calamidades, ya sea por fallas en BGP o por el fin de las direcciones IPv4, la verdad es que Internet funciona, y cada vez mejor.

Aquí permitan que el autor aventure algunas hipótesis que pueden tener que ver. Por un lado la participación activa de usuarios, tanto del sector privado, de la academia o simplemente internautas, que han encontrado míriadas de soluciones para "emparchar" la red. Internet mismo como medio de relacionamiento alimenta esa máquina de encontrar respuestas que se llama comunidad y se visualiza en los foros. Por otro lado, las capacidades técnicas de los equipos en juego se han multiplicado (enrutadores, procesadores, comunicaciones inálambricas), dándole un poco de aire a la red, además de una sobreutilización de las capacidades iniciales que ofrecían los protocolos.

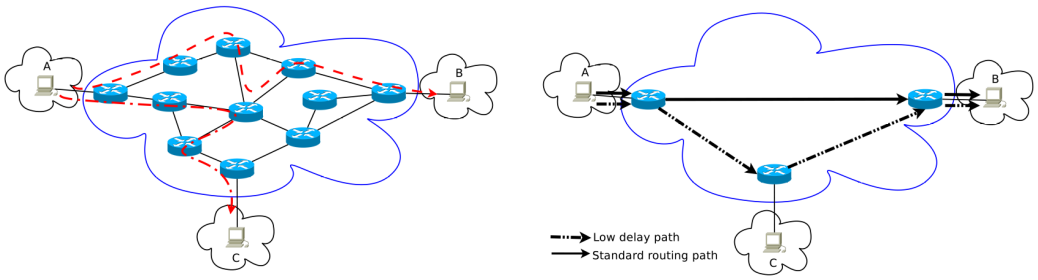
La gran cantidad de sistemas autónomos y la creciente interconexión entre los mismos también permite la existencia de múltiples rutas paralelas para considerar entre dos nodos. Está bien demostrada no sólo la existencia de estas rutas, sino que en muchas ocasiones estas rutas son más rápidas que la ruta elegida por BGP, y tienen menos pérdidas.

Esta Tesis se centrará en una métrica en particular, que es el tiempo de ida y vuelta (RTT, por sus siglas en inglés: *Round-Time Trip*). Es decir el tiempo que demora en ir y volver un paquete desde el origen hasta el destino. Cuanto más chico el RTT, más rápida la comunicación. Según la aplicación que se le quiera dar a la red se priorizan ciertas características frente a otras, en lo que se llama calidad de servicio (QoS, por *Quality of Service*). Un RTT bajo es bueno para todas las aplicaciones en general, pero es especialmente importante para la distribución de contenidos, una buena calidad de experiencia del usuario, y aplicaciones en tiempo real.

Desde que se detectó la existencia de rutas alternativas más ventajosas, se han desarrollado varias líneas de trabajo para hacer un mejor aprovechamiento de los recursos. Algunas proponen modificaciones al protocolo BGP, o su sustitución paulatina por protocolos más adaptables y con mayores posibilidades de realizar ingeniería de tráfico. En particular, una opción que ha tomado fuerza es la utilización de redes sobrepuestas (ON, *overlay networks*). De hecho, varias de las soluciones mencionadas sugieren la utilización de ON para ir reemplazando al "obsoleto" BGP, dado que se monta sobre la arquitectura existente y es compatible con los desarrollos actuales.

---

users-being-actively-blocked-on-some-websites/. Sobre el debate ANTEL-Movistar https://www.180.com.uy/articulo/83566_movistar-quiere-arrendar-fibra-optica-de-antel-para-desarrollar-el-5g.



(a) a (b) b

Figura 1.1: Ilustración de los dos esquemas de conexión: (a) una red BGP y (b) una red ON conectando A, B y C.

Las redes sobrepuestas son conjuntos de nodos interconectados, montados sobre otra red. Como Internet en sus inicios, cuando se montaba sobre la red telefónica. Imagine una institución con sede en varios países (y entonces seguramente diferentes AS), que desea tener una comunicación con ciertas garantías de calidad de servicio o privacidad entre sus nodos, para lo que establecerá "túneles" entre cada uno de estos. Estos "túneles" no son más que la especificación de una ruta determinada y la reserva de recursos para esa ruta entre los dos nodos. Sin embargo, al establecer un túnel esta ruta se tomará siempre, y con garantías de seguridad entre otras. En la figura 1.1 se aprecian tres nodos conectados por BGP y por una red sobrepuesta. Es importante tener en cuenta que las redes sobrepuestas forman una red por abstracción, ya que los paquetes nunca dejan de viajar sobre la red y utilizando BGP para transitar entre los AS.

Las redes sobrepuestan son muy utilizadas, en particular en las llamadas *Content Delivery Networks* (CDN, redes de distribución de contenidos). Dado que estas redes tienen grandes volúmenes de tráfico para distribuir a multitud de usuarios, son unas de las principales interesadas en tener tráfico veloz, sin latencia y con pocas pérdidas. Para esto, por ejemplo pero no únicamente, distribuyen en Data Centers el contenido para acercarlo a los usuarios. Se estima que Akamai, una de las gigantes de Internet que ofrece servicios de CDN, ocupa entre el 15 y el 30 % del tráfico de Internet. Los servicios de Akamai son utilizados por casi todas las otras grandes empresas de distribución de contenidos [4]. Esta empresa, así como en general las CDN, han implementado redes sobrepuestas para establecer políticas propias de ruteo entre sus nodos (y así realizar ingeniería de tráfico).

El objetivo de estos gigantes de la red es optimizar los recursos de que dispone: en este caso los que contrata a los ISP y los propios. Con el establecimiento de políticas propias de ruteo, pueden ajustar las rutas dinámicamente de manera más eficiente que BGP. Un caso sencillo de ejemplo sería el de la figura 1.1: entre A y B hay dos rutas posibles. Una es la ruta por defecto asignada por BGP, que es la que tiene menor cantidad de saltos (y no pasaría por C). La otra tiene más saltos, pero podría ser más rápida. De contar con una política de ruteo inteligente, se

[4]Se consultó la página https://www.akamai.com/us/en/our-customers/ el 01/07/20.

podría ir tomando en cada momento la ruta más rápida.

Ahora, ¿cómo saber cual es la ruta más rápida? En general, los cambios en el tiempo de ida y vuelta de las rutas se deben a dos fenómenos: cambios en la elección de la ruta IP, o congestión en algún enlace. Para conocer cambios en la elección de la ruta, es necesario realizar mediciones permanentemente, y para saber si va a haber congestión, no sólo es necesario realizar mediciones, sino que estas deben manejar un volumen de tráfico similar al que se desea soportar.

El estudio de mediciones en Internet es un tema muy vasto, y con décadas de conjeturas, resultados e implementaciones. En particular, algunas soluciones se basan en el modelado de las rutas. Otras consisten en formas de medición que permiten inferir o conocer el estado de las rutas.

Las redes sobrepuestas deben intercambiar paquetes de medida entre sus nodos si quieren conocer el estado de las rutas, dado que no acceden al conocimiento de la topología completa de la red (que sí tienen, al menos parcialmente, los ISP). Enviar permanentemente estos paquetes entre todos los nodos puede provocar congestión, por lo que usualmente se busca un balance entre el costo de realizar las medidas y el beneficio de conocer el estado de la red.

En este trabajo se busca solucionar el problema de elegir la ruta más rápida minimizando el costo de medida para una red sobrepuesta. En los próximos párrafos se describen antecedentes y soluciones respecto de las limitaciones de BGP y del ruteo en redes sobrepuestas, así como sobre el desarrollo de algoritmos de ruteo inteligentes basados en medidas.

1.2. Antecedentes

A continuación se describen brevemente algunos trabajos académicos que abarcan el área de análisis de rutas interdominio y BGP, el desarrollo de ingeniería de tráfico entre sistemas autónomos y el surgimiento y utilización de redes sobrepuestas para esto. Luego se pasa al estudio de trabajos sobre la medición y el modelado de la latencia en particular, y la metrología de Internet en general.

1.2.1. Optimización del ruteo en redes sobrepuestas

Quienes primero detectaron y demostraron la existencia de rutas mejores que las elegidas por BGP, en el milenio pasado, fueron Paxson y Labovitz (con colaboradores), en múltiples trabajos al respecto. En [1], Paxson plantea que el 20 % de los caminos interdominio demora más de 10 minutos en recuperarse ante fallas. También desarrolla sobre el surgimiento de BGP y los ISP como concentradores de información. En [2], Labovitz et al. sostienen que el 40 % de las fallas demoran hasta 30 minutos en resolverse. Estos artículos, junto a [3], [4], y [5] demuestran ya a finales del siglo pasado y principios de este los inconvenientes que presenta BGP, y la posibilidad de usar políticas de ruteo más inteligentes para aprovechar rutas paralelas más rápidas. En [6] se muestra que BGP tiene una respuesta lenta a fallas (el algoritmo de vector distancia es lento para converger), y esto puede

provocar demoras del orden de hasta decenas de minutos en la recuperación ante las mismas.

Trabajos más recientes como [7], [8], [9] y [10] estudian el vínculo entre variaciones en los retardos (vistos en general desde el usuario, es decir *end-to-end*) y la elección de la ruta.

Los primeros trabajos que buscan resolver el problema de elegir mejores rutas entre AS utilizando redes sobrepuestas son los de RON [11] y Detour [12], [13]. Es importante destacar las posibilidades que ofrece una red *overlay* en términos de la aplicación de ingeniería de tráfico para atacar directamente al problema de la elección de rutas por mecanismos subóptimos como en BGP. A esto se agrega que no exige cambios en la arquitectura de Internet, dado que las redes sobrepuestas se montan sobre la red existente, por lo que ha tomado impulso como solución implementable y con beneficios inmediatos.

Con los *Resilient Overlay Network* (RON), se busca que la red sea menos susceptible a fallas. Para esto, luego de desarrollar sobre las limitaciones de BGP, proponen el desarrollo de una red sobrepuesta, que pueda hacer uso de la multiplicidad de caminos entre nodos para obtener una red más estable y con mejor rendimiento. Dado que es uno de los primeros trabajos en el área, y que plantea sucintamente los problemas que acarrea BGP, se copia un fragmento de esta descripción:

"The information shared with other providers and AS's is heavily filtered and summarized using the Border Gateway Protocol (BGP-4) running at the border routers between AS's, which allows the Internet to scale to millions of networks. This wide-area routing scalability comes at the cost of reduced fault-tolerance of end-to-end communication between Internet hosts. This cost arises because BGP hides many topological details in the interests of scalability and policy enforcement, has little information about traffic conditions, and damps routing updates when potential problems arise to prevent large-scale oscillations. As a result, BGP's fault recovery mechanisms sometimes take many minutes before routes converge to a consistent form, and there are times when path outages even lead to significant disruptions in communication lasting tens of minutes or more. The result is that today's Internet is vulnerable to router and link faults, configuration errors, and malice - hardly a week goes by without some serious problem affecting the connectivity provided by one or more Internet Service Providers (ISPs)."

A su vez, los autores plantean que el problema de mandar paquetes de prueba en este esquema tiene una cota de 50 nodos conectados *full-mesh*, dado que es la topología de un grafo completo y es de complejidad $O(n^2)$. Se encuentra en [14] una forma de disminuir la complejidad a $K \leq O(nlog(n))$ a través de medir en una selección de K caminos entre los n posibles.

En Detour [13], los autores proponen una red sobrepuesta que utiliza datos reales de tráfico para ajustar las rutas. Justifican que para la mitad de las rutas consideradas existe una ruta alternativa más rápida. Si bien consideran utilizar *multipath routing*, que es dividir el tráfico entre varias rutas origen-destino, esto puede implicar problemas de reordenamiento para ciertos protocolos (como ser TCP). Para estimar la ruta más rápida utilizan **traceroute**, y ya estiman las

dificultades que puede inducir realizar medidas permanentemente, adelantando las dificultades que encontraría esta propuesta para escalar a escenarios con más nodos.

Un trabajo interesante defiende a las redes sobrepuestas como la única alternativa viable para solucionar los problemas de BGP [15]. En el mismo, los autores desarrollan el debate que se da entre "pluralistas" y "puristas". Esquemáticamente, los "puristas" desean mantener un Internet centrado en el protocolo IP, alrededor del cual todo debe girar, y para ello proponen arquitecturas flexibles y sólidas, que perduren en el tiempo. Por otro lado, los "pluralistas" imaginan un Internet en que IP es no más que un protocolo conviviendo con múltiples protocolos, y para lo que sugieren alternativas más dinámicas y ajustadas a obtener un beneficio inmediato. Más allá del debate, que es interesante, los autores explican cómo las redes sobrepuestas permiten la coexistencia de políticas propias y políticas generales, permitiendo una evolución gradual hacia una red de redes sobrepuestas.

Otros artículos como [16] y [17] proponen cómo debe ser la arquitectura de la red sobrepuesta. En este último trabajo separan además ruteo y *forwarding*. Con ruteo nos referimos a la constitución de la ruta a seguir, y con *forwarding* al acto de reenviar un paquete que llega hasta un enrutador. En este caso, las rutas se elegirían centralmente, y a cada enrutador se le indicaría cómo hacer el *forwarding* de los paquetes según el destino de cada paquete. En [18] también se propone el desarrollo de redes sobrepuestas apuntando a mejorar la comunicación *end-to-end* (punta a punta), es decir, orientadas al usuario.

Por otro lado, [19] analiza la coexistencia de políticas en ON con los protocolos comunes de Internet. Estudian el impacto que tiene establecer políticas de enrutamiento "egoístas" frente a tener las políticas de los protocolos comunes, a la realización de ingeniería de tráfico, etc. Analizan si cada comunicación que se establece sobre Internet buscara su máximo beneficio, cómo impactaría en el funcionamiento global de la red. Concluyen que este esquema de utilizar *selfish routing* funciona bien (cerca de la latencia media óptima), pero al costo de generar congestión en algunos enlaces y de tornar la red un poco más inestable (y por ende resulta más difícil realizar ingeniería de tráfico).

Hay autores que se preguntan qué tanto beneficio pueden reportar las arquitecturas *multi-homed* y *overlay* [20]. En particular desean analizar si son ciertas las suposiciones sobre la independencia entre las rutas posibles y sobre las ventajas de tomar rutas paralelas para mitigar fallas. Así como comparten origen y destino, observan que las rutas comparten típicamente uno o más AS (sin contar origen y destino). Concluyen: *"These findings demonstrate that simply placing overlay nodes in different ISPs cannot provide enough control over path selection at the IP layer to provision disjoint paths to destinations. To mitigate this limitation, we believe that topology-aware overlay placement is critical."* De manera similar, [21] busca confirmar la ventaja de tomar caminos alternativos *one-hop source routing* basados en los trabajos de RON y Detour. Para esto utilizan paquetes de prueba, que miden regularmente todas las rutas, y que en caso de pérdida de uno de estos paquetes realizan medidas más agresivas (triplicando la frecuencia de las medidas). En caso de detectar una falla, buscan con una política "K-random" en que

elijen aleatoriamente K caminos posibles para probarlos y ver si es posible llegar a destino, y muestran que con esta política se pueden obtener resultados casi óptimos.

Otro trabajo enriquecedor es [22], en que los autores dicen ser los primeros en cuantificar los beneficios de utilizar ON y un sistema centralizado para la toma de decisiones de ruteo. Si bien varios trabajos anteriores habían comprobado la existencia de rutas mejores, en este se utilizan nodos distribuidos para confirmar estos resultados a nivel global. Muchos de los estudios inciales se realizaron concentrados en Estados Unidos. Observan que en Asia la latencia de un cuarto de los caminos podría bajar hasta 50 %, mientras que en USA menos del 10 % de las rutas tiene esa posibilidad de reducción. Con esto hacen hincapié en que la comunidad científica e ingenieril estaría subdimensionando los beneficios de implementar ON e ingeniería de tráfico. Proponen un algoritmo sencillo para elegir las K mejores rutas disponibles, y distribuir el tráfico por estas K rutas. Si bien el problema de obtener el menor RTT posible se aproxima al óptimo con esta solución, no es escalable, dado que multiplica el tráfico por K caminos.

Otros trabajos como [23] proponen usar redes definidas por software para aprovechar las capacidades de las redes sobrepuestas. Las redes definidas por software son redes en que el plano de ruteo se separa completamente del plano de *forwarding*. Las decisiones de ruteo se toman centralmente por un controlador, y se informa a los enrutadores de los caminos que deben seguir los distintos flujos. SDN tiene aplicación directa para redes *overlay* y el problema estudiado, y hay varios trabajos que proponen distintas formas de realizar ingeniería de tráfico para optimizar el ruteo en base a algún criterio, criterios que suelen estar asociados a métricas. En [23] proponen una arquitectura SDN que incorpora la posibilidad de realizar mediciones y a partir de las mismas organizar el tráfico.

Otro trabajo que utiliza SDN para atacar el problema de ruteo orientado a la calidad de servicio en redes sobrepuestas es [24], en que se propone a los ISP la tercerización del ruteo a un administrador. Este administrador manejaría una SDN que incluiría a todos los AS con los que tiene contrato. Entre estos AS realizaría ingeniería de tráfico para optimizar la elección de rutas, aprovechando el conocimiento topológico de manejar varios AS y la posiblidad de utilizar varios caminos paralelos. Por otro lado, [25] propone utilizar SDN para dividir en varias rutas los distintos tipos de tráfico según sus requerimientos de QoS, realizando medidas para cada ruta y hacia cualquier destino.

Hay varios artículos que utilizan el paradigma de las "redes cognitivas" o *cognitive networks*. Estas redes fueron definidas por Ryan W. Thomas en su Tesis doctoral **Cognitive networks** [26] y propone: *"A cognitive network is a network composed of elements that, through learning and reasoning, dynamically adapt to varying network conditions in order to optimize end-to-end performance. In a cognitive network, decisions are made to meet the requirements of the network as a whole, rather than the individual network components."* Es decir ajustar el aprovechamiento de los recursos de la red según las necesidades del momento. Para esto se requiere de cierta inteligencia (algoritmos de ruteo), así como de control sobre las decisiones de ruteo. Las SDN pueden verse como un caso particular de redes

cognitivas en que las decisiones se toman centralmente.

Una publicación que resultó particularmente interesante fue [27], en que los autores proponen un esquema de medición y aprendizaje del estado de la red para elegir la mejor ruta. En este trabajo utilizan paquetes de prueba (sin carga útil) y de tráfico (con carga útil según el tipo de tráfico). Para elegir las rutas, lo que se realiza centralmente, se utiliza una red neuronal recursiva. Esta red define rutas para distintas clases de tráfico según el origen y destino, utilizando la opinión de los nodos como oráculos (cuyos pesos se actualizan usando aprendizaje por refuerzo). La información sobre retardos y pérdidas se obtiene enviando mensajes de prueba a los vecinos. Si los vecinos saben llegar hasta el destino, envían su ruta al controlador, que definirá centralmente la ruta a seguir. Si no la tienen, repiten el proceso con sus vecinos, hasta que se dé una de las condiciones que se obtenga una ruta, se llegue al destino, o a un máximo de saltos. Con cada mensaje de prueba y su respectivo reconocimiento (*Ack*) se obtiene información del RTT y de las pérdidas del tramo correspondiente (que es siempre sólo entre vecinos).

En [28] (y [29]) se busca resolver un problema muy similar al que se desea abarcar en la Tesis: encontrar la mejor ruta entre dos nodos que pertenecen a una red sobrepuesta, de manera dinámica y en base a medidas. En este caso, los autores disponen de una cantidad fija K de medidas para realizar en cada tiempo de decisión, deben entonces seleccionar cuales serán las K rutas a medir de entre todas las rutas posibles. Formulan el problema como un *multi-armed bandit* y utilizan el algoritmo EXP3[5] con modificaciones. Es decir que en cada momento se elijen los caminos a medir siguiendo una distribución de probabilidad con pesos hallados según el algoritmo mencionado.

Muchos de los trabajos vistos usan exploraciones parciales: mandar paquetes de medida sólo en algunas de las rutas posibles, o no periódicamente sino en ciertos casos. La búsqueda de disminuir el tráfico de medida responde a evitar sobrecargar la red con este tipo de tráfico. Esto sucede a partir de dos motivos fundamentalmente: cuando se desea estimar el efecto de enviar un determinado tipo de tráfico, se debe enviar un tráfico con características similares. Esto puede implicar tráfico pesado y causar congestión en los enlaces de la ruta testeada [30]. Por otro lado, tener intercambios entre todos los nodos en una topología *full-mesh*, aunque de paquetes ligeros de medida (sin carga útil), genera una cantidad de tráfico que crece a tasa $\mathcal{O}(n^2)$ [11] [31] [32], por lo que también puede degradar el estado de la red. Ambas causas no sólo degradan el estado de las rutas pudiendo causar congestión, sino que al mismo tiempo son susceptibles de alterar las medidas realizadas.

Se ha visto que existen muchas soluciones y herramientas que permiten realizar ingeniería de tráfico orientada a la calidad de servicio. En particular, las redes sobrepuestas presentan características que las han llevado a sobresalir entre las alternativas. El paradigma de las redes definidas por software es propicio para que la toma de decisiones sea inteligente, combinando una mirada global de la red y las necesidades de conexión *end-to-end*. Sin embargo, muchas de las soluciones

[5]Es un algoritmo de pesos exponenciales con un estimador sesgado para los pesos *exponential weighting with a biased estimate on the gains*

planteadas no son escalables, en particular por la necesidad de realizar medidas *full-mesh*. Esto resulta en un esquema de grafo completo, por lo que es un problema computacionalmente muy costoso, que al mismo tiempo puede causar congestión en la red. Otras parten de asunciones muy rígidas sobre el modelado de la red.

Recuérdese que este trabajo busca la solución al problema de elegir la mejor ruta minimizando el costo de medida. Habiendo justificado la ventaja de elegir rutas mediante la utilización de redes sobrepuestas y posibles implementaciones, se presentarán a continuación algunos trabajos que estudian la toma de decisiones en base a medidas, las mediciones y el modelado de los caminos.

1.2.2. Mediciones en redes sobrepuestas

Dado que se busca disminuir la cantidad de medidas a realizar, es necesario estimar o predecir el estado de los caminos en que no se realizarán medidas. Hay varios esfuerzos en el sentido de explicar a qué se deben las fallas y retardos en Internet. Hay dos causas principales que provocan variaciones importantes en los retardos de la red: cambios en los flujos de tráfico y cambios en la topología de la red.

Es necesario comenzar con una referencia a Paxson, que en [1] demuestra que con medir las conexiones extremo a extremo se puede inferir en buena medida el estado de la red. Esto facilita mucho el análisis de las rutas BGP (o más en general entre AS), dado que los ISP pueden retener información sobre la topología por considerarla confidencial, o simplemente descartarla o filtrarla.

En particular, [8] busca asociar cambios en el RTT a eventos de ruteo, tanto para cambios en el ruteo interno a un sistema autónomo como en el ruteo entre sistemas autónomos. Si bien los cambios en el ruteo intradominio son más frecuentes, tienen el mismo (o menos) impacto que los cambios en las rutas AS, que aunque son más estables propagan las fallas y demoran en volver a converger.

Los autores de [10] buscan desarrollar un método de medición de las fallas interdominio en BGP. Asumen que se realizan medidas activas y que hay recolectores de información distribuidos en un ISP. Quieren determinar cúales de los recolectores de información son los mejores candidatos para estudiar la correlación entre los cambios en las medidas del tráfico y los cambios interdominio. Proponen una solución que prueban utilizando la mediciones de RIPE Atlas[6] [33].

En [9] los autores buscan discernir cuales cambios en el RTT se deben a cambios en las rutas BGP y cuales son consecuencia de variaciones del tráfico. Para esto consideran el conjunto de rutas paralelas y plantean que si las rutas tienen solapamiento en los valores de RTT, la principal causa de variación del RTT es la elección de la ruta, mientras que si no lo tienen, la principal causa de variación serán las fluctuaciones de tráfico para la ruta elegida.

Otro trabajo que busca estimar los efectos tanto del ruteo como de la congestión en la red es [30]. Los autores proponen una forma de medir que consiste en el envío periódico de *light probes* (pilotos livianos de prueba), para conocer el estado de la

[6]RIPE Atlas es un conjunto de nodos distribuidos que reúne datos del estado de Internet para investigación.

red, principalmente debido a la topología y políticas de ruteo. Por otro lado, desean evaluar la capacidad de las posibles rutas para soportar distintos tipos de tráfico sin generar congestión, para lo que se envían ráfagas de prueba de características similares a las que se desea transmitir.

En [34] se utilizan redes neuronales recurrentes para modelar el tiempo de ida y vuelta, buscando minimizar la esperanza del error cuadrático medio $\mathbb{E}(RTT - \hat{RTT})^2$. Estas redes se entrenan con los RTT observados en un día y predicen el RTT del día siguiente, pero aunque logran buenos resultados para los tiempos más cercanos al período de entrenamiento, a medida que crece esta distancia los resultados empeoran.

Dentro del área de tomografía de Internet [7] [35], se encuentran trabajos que se acercan al problema de esta Tesis, dado que estiman el estado de la red a partir de medidas. Artículos como [36] y [37] utilizan modelos markovianos para la estimación del RTT. En [38], buscan la cantidad mínima de puntos de medida para obtener una aproximación buena al estado de la red, con el objetivo de disminuir la complejidad del problema de tener los nodos como un grafo completo, que es *NP-hard*[8]. De forma similar, [32] apunta a reducir la topología de la red sobrepuesta, evitando enlaces duplicados y permitiendo entonces mayor escalabilidad. En [39] y [31] se exploran alternativas para resolver el problema de buscar el camino más corto en grafos completos, y proponen algoritmos para mitigar este problema, buscando minimizar las mediciones de ciertas aristas: ya sea por ser costosas, por no corresponder a alguno de los caminos posibles, o por ya haber sido medidas (por pertenecer esa arista a más de una ruta).

Recientemente, en [40] y [41] se propone la utilización del proceso jerárquico de Dirichlet para modelos markovianos de estados ocultos (*Dirichlet Herarchichal Process Hidden Markov Models*, DHP-HMM) para estimar los niveles y las matrices de transición del proceso markoviano. Los procesos de Dirichlet, definidos en [42], [43] y vinculados a las HMM en [44] son procesos bayesianos no paramétricos, y los modelos de Márkov de estados ocultos asumen que existen estados ocultos (con ruido alrededor de los niveles posibles) y por esto se adaptan a la estimación de estados posibles a partir de una traza cuyos estados no se conocen a priori.

Como argumentan los autores en [41], la pertinencia de utilizar modelos de Márkov de estados ocultos para modelar el RTT se debe a:

"From the analysis of these data sets we affirm that RTT time series have a typical behavior that can be captured by a Markov chain (MC) or a Hidden Markov Model (HMM). The observed behavior is that RTT values switch among several probability distributions. [...] We have noticed that, for a given Origin Destination pair, there exist few probabilistic laws according to which RTT take their values,and that the law of the RTT remains stable for some time. This behavior is in part explained by load-balancing and routing configuration changes in operator networks."

[7] El área de estudio que se ocupa de inferir el estado de la red a partir de mediciones indirectas.

[8] NP-hard significa que el problema es al menos tan complejo como NP (se resuelve en tiempos no polinómicos).

Con un planteamiento similar, [45] aborda el problema conocido como *pilot allocation*, en que una radiobase debe distribuir el tiempo en que ubicar pilotos de medidas de los distintos usuarios. Proponen la utilización de un algoritmo de *multi-armed bandit*, dado que consideran que el *fading* de cada ruta sigue un proceso markoviano (modelan el canal como una cadena de Márkov de K estados parcialmente observables POMP). En el artículo [46] se formula un problema muy similar al del ruteo con limitación en la medida y se resuelve a partir del método de optimización ADMM (*alternating direction method of multipliers*). Su objetivo es encontrar la agenda óptima para ubicar periódicamente sensores de medidas, que permitan estimar el estado de un sistema dinámico en tiempo discreto. Estos trabajos nos permiten pensar en una perspectiva más amplia sobre aplicaciones del problema de tomar decisiones en base a medidas con costo.

En esta Tesis se consideran dos escenarios para resolver el problema de ruteo en base a mediciones. En el primero se utilizará el modelado markoviano introducido en los párrafos anteriores. Es decir que se parte de la suposición de que se puede aproximar la dinámica de los retardos de cada ruta por una cadena de Márkov, lo que nos permite modelar el problema como un Proceso de Decisión Markoviano. En el segundo escenario, se prueban herramientas de aprendizaje supervisado para la optimización del ruteo, sobre un modelado del sistema en base a medidas simultáneas y periódicas.

1.3. Síntesis de la propuesta y principales resultados

Esta propuesta de optimización del ruteo en base a medidas busca resolver el problema de elegir la ruta más rápida minimizando el costo de medida. Se formulará este problema como un problema de optimización. De esta forma, el problema consiste en minimizar la esperanza de la suma del tiempo de ida y vuelta más el costo de las medidas realizadas.

Se responderá, en cada época de decisión:

- ¿qué ruta(s) debo medir?
- ¿qué ruta debo elegir para encaminar el tráfico?

Se parte de los resultados de la colaboración conjunta entre el grupo ARTES[9] con la Universidad IMT Atlantique[10] y la Universidad de Toulouse[11], que en [41] plantean que los retardos de cada ruta siguen procesos markovianos (ver también [40]). En este artículo se formula el problema general como un Markov Decision Problem (MDP), gracias a lo cual se puede encontrar la solución exacta al problema utilizando programación dinámica [47].

[9]Grupo ARTES: grupo de investigación de Análisis de de Redes, Tráfico y Estadísticas de Servicio, del IIE, FING. https://iie.fing.edu.uy/investigacion/grupos/artes/

[10]IMT Atlantique, Brest, Francia https://www.imt-atlantique.fr/fr.

[11]LAAS-CNRS, *Université de Toulouse*, CNRS, INSA, Toulouse, Francia, https://www.laas.fr/public/.

Sin embargo esto es posible cuando hay pocas rutas con pocas variaciones en el RTT, pues el cálculo de esta solución no escala a topologías más complejas, por caer en el *curse of dimensionality*. El principal aporte de esta Tesis se encuentra en los capítulos 3 y 4, en que se proponen dos soluciones más escalables que ofrecen muy buenos resultados para resolver el problema planteado.

Primero se propone una solución aproximada bajo el método de horizonte errante (*receding horizon*, RH). Esta política es estacionaria y de horizonte móvil: considera la evolución del sistema H pasos hacia el futuro (decimos que tiene horizonte H). Este algoritmo permite mayor escalabilidad, a pesar de también tener sus restricciones, dado que creciendo H se vuelve a caer en el *curse of dimensionality*.

Luego se realiza una formulación del problema para la utilización de técnicas de aprendizaje supervisado para clasificación. Este planteo busca eliminar la asunción de que los retardos de las rutas son procesos markovianos. Para implementar este procedimiento se desarrolla un algoritmo que utiliza el clasificador de *random forest* y permite elegir la ruta más rápida minimizando el costo de medida. Se imponen dos restricciones: que se miden todas las rutas en simultáneo y con periodicidad T. En esta formulación, las características (*features*) del sistema son el RTT obtenido al medir y hace cuanto tiempo se realizó esta medida, y la clase es la ruta a elegir. Se busca primero el T^* óptimo según un función de costo (retorno esperado) que considera la suma del costo de medida y el RTT esperado. Luego se emplea ese T^* para realizar las medidas gracias a las que se elegirá la ruta.

Parte de los resultados presentados en este trabajo fueron publicados en la conferencia NOMS 2020 [48]. En particular, el planteamiento del problema como un MDP y su resolución mediante el algoritmo de horizonte errante.

Tanto aplicado a trazas sintéticas como a trazas reales se obtienen ventajas al utilizar los algoritmos propuestos. En particular, con el algoritmo de horizonte errante sobre mediciones reales provenientes de *NLNOG* se obtienen reducciones muy importantes de hasta el 20 % del tiempo de ida y vuelta medio respecto de la ruta por defecto. Esta reducción se realiza además con un número muy reducido de medidas.

La solución propuesta de aprendizaje supervisado (usando *random forest*) obtiene valores muy cercanos a los del horizonte errante, incluso superando a este en algunos casos de rutas poco markovianas. Los resultados alcanzados por este algoritmo aún se encuentran pendientes de publicación.

1.4. Organización del documento

En el capítulo 2 se presenta el problema como un proceso de decisión markoviano hallando la solución óptima y analizando su complejidad numérica.

Luego, en el capítulo 3 se propone la solución aproximada por horizonte errante y describimos su implementación al problema de ruteo y metrología.

El capítulo 4 está dedicado al aprendizaje supervisado aplicado a nuestro problema. Introducimos brevemente la motivación de utilizar otras herramientas de inteligencia artificial y una implementación utilizando el algoritmo de *random forest*.

En el capítulo 5 se analizan los resultados obtenidos. Se detallan las pruebas realizadas en trazas sintéticas y reales. También se proponen variantes y mejoras para los algoritmos de horizonte errante y aprendizaje supervisado.

Finalmente, en el capítulo 6 se presentan las principales conclusiones y posibles líneas de trabajo a futuro.

Capítulo 2

Formulación como un Proceso de Decisión Markoviano

> Cuando puedes medir aquello de lo que hablas, y expresarlo con números, sabes algo acerca de ello; pero cuando no lo puedes medir, cuando no lo puedes expresar con números, tu conocimiento es pobre e insatisfactorio: puede ser el principio del conocimiento, pero apenas has avanzado en tus pensamientos a la etapa de ciencia.
>
> William Thomson Kelvin

Este capítulo está dedicado al estudio de los llamados sistemas sin memoria, o markovianos[1]. Los procesos markovianos se definen como sistemas en que toda la información pasada se resume en el estado actual. Esta tesis se centrará en los problemas de decisión markovianos (*Markov Decision Problems*, MDP), problemas de control regidos por una dinámica markoviana del sistema, en particular una cadena de Márkov de tiempo discreto (DTMC). La utilización de procesos markovianos para modelar retardos en redes de datos es abundante en la literatura ([41], [40], [51], [36], [37], [38]).

El funcionamiento de un MDP puede verse como el sistema de la figura 2.1 [47]. En estos sistemas, en cada intervalo temporal un tomador de decisiones sigue una política para elegir acciones, relativas al estado en que se encuentra el sistema y con el objetivo de maximizar un retorno. En cada instante de tiempo, decimos que el sistema se encuentra en el estado S_t, el agente toma la acción A_t y el sistema

[1]Por Andréi Márkov (14 de junio de 1856-20 de julio de 1922), matemático ruso. Entre otros aportes significativos, su estudio de los procesos estocásticos (1906) dará lugar al origen de las cadenas de Márkov (nombre acuñado en 1926). Profeso ateo y militante, fue llamado el "académico militante" por los medios al rechazar reconocimientos del Zar y al solicitar su excomulgación de la Iglesia Ortodoxa (a la cual no pertenecía), luego de la excomulgación de Gorky, quien fuera su amigo [49]. En 1913, cuando el gobierno festejaba los 300 años de la dinastía Románov, Márkov organizó una contra-celebración usando el 200 aniversario de la Ley de los Grandes Números de Bernoulli [50].

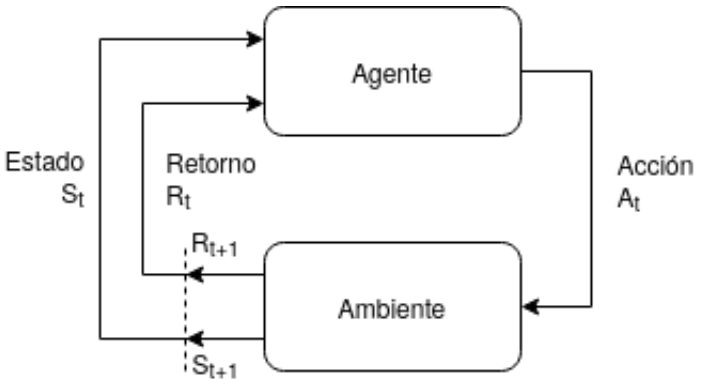


Figura 2.1: Interacción agente/ambiente. Ilustración basada en la figura 3.1 de [47].

pasa al estado S_{t+1}, obteniendo un retorno R_{t+1} y con esto, una observación del sistema. De no contar con una observación directa, el estado S_{t+1} puede inferirse estadísticamente usando las propiedades markovianas del sistema.

En la siguiente sección se definirán términos y establecerán notaciones, para luego presentar el modelado del problema como un proceso markoviano discreto, finalmente proponiendo los algoritmos utilizados.

2.1. Definiciones y formulación

2.1.1. Notación y asunciones

En esta sección se definen conceptos y notaciones que facilitarán el planteamiento del problema como un proceso de decisión markoviano [51]. Se utiliza la notación propuesta en [47].

En primer lugar, se define un MDP como el conjunto de 5 elementos: **estados**, **acciones**, **retornos**, **probabilidades de transición** y **épocas de decisión**. El objetivo de un MDP consiste en optimizar el retorno (maximizar una recompensa o minimizar un costo) a lo largo del tiempo; siendo que en cada época de decisión se puede realizar una acción sobre el sistema, observar el estado del mismo, y que el estado del sistema cambia según la acción realizada y la dinámica propia del sistema. Se definen entonces:

- **épocas de decisión:** en los sistemas markovianos discretos, son los puntos del tiempo en que se realizan acciones y en que ocurren las transiciones entre estados del sistema. Se llama T al conjunto de las épocas de decisión, y t a una época de decisión genérica. En los sistemas homogéneos, los retornos y las probabilidades de transición son independientes de t, así como las políticas, que ya serán definidas.

- **estados:** el conjunto de valores que puede tomar la información que describe al sistema dinámico. Sea $\mathcal{S}$ el conjunto de estados posibles y S_t el estado en

la época t. Referido a una época, se usará s para designar el estado actual y s' un posible estado siguiente.

- **acciones:** el conjunto de acciones que puede realizar el tomador de decisiones sobre el sistema. Se llamará $\mathcal{A}$ al conjunto de acciones posibles, A_t a la acción tomada en la época t, y a a una acción genérica.

- **probabilidades de transición:** la probabilidad de pasar de un estado al siguiente. Es una probabilidad condicionada a la acción y al estado actual. Se escribirá como $p(s'|s,a)$ la probabilidad de pasar al estado s' dado que el sistema se encuentra en el estado s y se realiza la acción a. En un sistema markoviano las transiciones entre estados están dadas por las matrices de probabilidad de transición P, de dimensión la cantidad de estados, y cuya fila indica el estado actual y columna el estado siguiente. La probabilidad de pasar del estado $s = s_i$ al estado $s' = s_j$ vale $p(s_j|s_i) = P[i,j]$.

- **retorno:** el objetivo del tomador de decisiones es optimizar una función objetivo, ya sea maximizar una recompensa o minimizar una pérdida. El retorno depende de la época de decisión, del estado actual, de la acción tomada y del estado siguiente. Se define a $\mathcal{R}$ como el conjunto de retornos posibles, r a un retorno genérico, y R_t al retorno obtenido en la época t. Se considera $R(s,a)$ al retorno inmediato esperado al tomar la accion a partiendo del estado s y llegando al estado s': $R(s,a) = \mathbb{E}[R_t|S_{t-1} = s, A_{t-1} = a] = \sum_{r\in\mathcal{R}} r \sum_{s'\in\mathcal{S}} p(s', r|s, a)$.

La colección de elementos $\{T, \mathcal{S}, \mathcal{A}, \mathcal{R}, P\}$ define un MDP.

En cada época de decisión el sistema se encuentra en un estado determinado, se toman acciones y obtienen retornos. En algunos problemas las acciones aportan información sobre el sistema, como observaciones. El árbol que se puede construir a partir de recorrer los estados de un MDP es como el de la figura 2.2, en el que se sigue una trayectoria particular $\{S_t, S_{t+1}, S_{t+2}\}$ cuyo recorrido siempre es de la forma $\{s, a, r, s', a', r', ...\}$.

Se define a la política π como el conjunto de acciones a tomar para cada estado. El objetivo consiste en encontrar la política π^* que optimiza una función objetivo, que en este caso será minimizar el retorno esperado acumulado y con descuento $\mathbb{E}_{\pi^*}[G_t]$ con $G_t = \sum_t^\infty \rho^t R_t$. El factor de descuento ρ garantiza la convergencia de la suma (infinita en principio) de los retornos a lo largo del tiempo. Por otro lado, permite asignar mayor importancia al retorno inmediato frente a retornos futuros. Siguiendo la política π se obtiene la acción $a = \pi(s)$ si esta política es determinística. En el problema a estudiar se trabaja con políticas markovianas y determinísticas (ver [51]).

En los problemas de horizonte infinito, el tiempo t irá desde el instante inicial que se considera $t = 0$ hasta ∞. Se asumirá que el problema a atacar es de horizonte infinito, dado que esto no sólo aporta elementos de convergencia para los algoritmos, sino que además es una suposición razonable para valores de t muy altos.

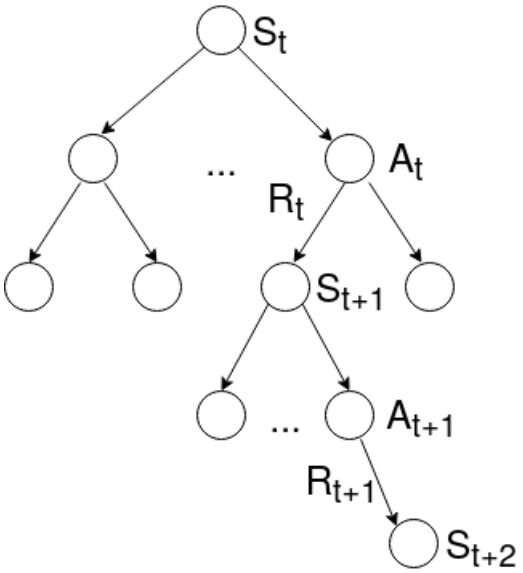


Figura 2.2: Evolución posible un ambiente, partiendo de un estado S_t.

Se verá a continuación cómo los problemas de medición y ruteo pueden representarse como MDP, a través de dos ejemplos sencillos y del planteo del problema genérico luego.

2.1.2. Calentando motores: buscando la política miope en dos ejemplos ilustrativos

Una ruta fija y una de retardo variable

Se comenzará por estudiar un problema en que solamente nos interesa realizar la acción que maximiza el retorno inmediato esperado. Es decir que no se considera el futuro, y en vez de buscar la política óptima para $\mathbb{E}_{\pi^*}[G_t]$ que considera los retornos futuros, se optimiza únicamente considerando $\mathbb{E}_{\pi^*}[R_t]$.

Considere el problema de elegir la ruta con menor tiempo de ida y vuelta (*round-time trip*, RTT), en una red compuesta por dos nodos y dos rutas (ver figura 2.3). La ruta 1 es de retardo fijo $L_1(t) = l$, mientras que la ruta 0 tiene dos valores de retardo posibles: $L_0(t) \in \{l_0, l_1\}$, con $l_0 < l < l_1$. La ruta con retardo variable se puede modelar por una cadena de Márkov de matriz de transiciones P conocida (ver [40], [41]), tal que:

$$P = \begin{bmatrix} p_{0,0} & p_{0,1} \\ p_{1,0} & p_{1,1} \end{bmatrix}$$

Donde p_{ij} representa la probabilidad de pasar de estar la ruta en l_i a l_j.

Medir en la ruta 0 tiene un costo constante c, mientras que en la otra ruta no se mide, ya que el retardo es fijo. Sea $a = \{0; 1\}$ a la acción de medir ($a = 1$) o no medir ($a = 0$). El problema que se desea resolver es cuándo medir, para obtener

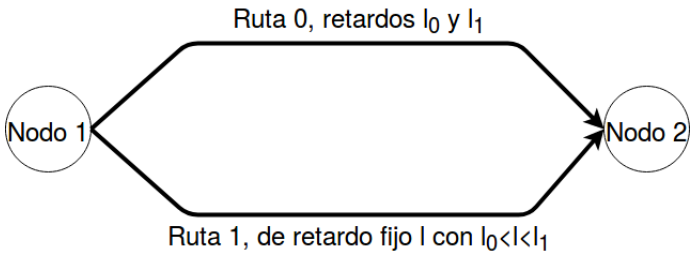


Figura 2.3: Esquema sencillo de dos rutas posibles. En este ejemplo sencillo, la ruta 0 tiene dos estados posibles (dos niveles de retardo) y la ruta 1 uno sólo.

una buena estimación del estado de la ruta markoviana, y poder elegir la ruta con menor tiempo de ida y vuelta. Es decir en cada época: decidir si se debe o no medir, y tomar la mejor decisión de ruteo posible.

Se define al estado del sistema como $s = (L_{Last}, \tau)$, donde L_{Last} es el último estado observado de la ruta 0, y τ es la cantidad de épocas de decisión transcurridas desde que se realizó la observación. Observe que se conoce el sistema dinámico con estos datos, dado que se puede saber con qué probabilidad se encuentra la ruta estocástica en cada nivel. Es decir que si se mide, la probabilidad de encontrar la ruta variable en el nivel l_i vale:

$$p(s' = (l_i, 0)|s = (L_{Last}, \tau), a = 1) = e_{l_i}^T P^{\tau+1} e_{L_{Last}}$$

Donde e_{l_i} y $e_{L_{Last}}$ son vectores columna de 0s, con un 1 en la posición l_i y L_{Last} respectivamente. En el caso en que no se mide, la ruta se encontrará en el estado $s' = (L_{Last}, \tau+1)$ con probabilidad $p(s' = (L_{Last}, \tau+1)|s = (L_{Last}, \tau), a = 0) = 1$.

Para elegir si medir o no, se busca minimizar el problema que combina el costo de medición con el riesgo de que la ruta tenga un retardo esperado menor/mayor a la ruta de retardo fijo. Esto es:

$$R(s, a) = a * c + \text{mín}(\mathbb{E}[L_0(t)], l)$$

Se analiza este retorno. Para esto, vea que los dos casos posibles son: se mide o no se mide la ruta 0. Suponga un estado genérico de esta ruta $s = (L_{Last}, \tau)$. Como reflexión general, sólo interesa tomar la ruta 0 si tiene un retardo esperado menor a l. Además, se debe realizar una estimación del resultado de medir, antes de haber realizado la medida, es decir: se debe considerar con qué probabilidad se hallará el sistema en el nivel bajo/alto cuando se mida, dado que el problema planteado se realiza previo a medir.

Ejemplificando: si con alta probabilidad la ruta variable se encuentra en su nivel alto (es decir en $L_0 = l_1 > l$), no será conveniente medir, y se elegirá dirigir el tráfico por la ruta de retardo fijo.

En caso en que no se mide la ruta 0, con probabilidad 1 pasará al estado $s' = (L_{Last}, \tau + 1)$. En este caso la esperanza del retardo tomando la ruta 0 vale:

$$\overline{L_0(t)} = \mathbb{E}[L_0(t)] = l_0 * e_{l_0}^T P^{\tau+1} e_{L_{Last}} + l_1 * e_{l_1}^T P^{\tau+1} e_{L_{Last}}$$

Casos	$\overline{L_0} \geq l$	$\overline{L_0} \leq l$
$a = 1$	(a): $x \in [\frac{c}{l-l_0}, \frac{l_1-l}{l_1-l_0}]$	(c) : $x \in [\frac{l_1-l}{l_1-l_0}, 1 - \frac{c}{l_1-l}]$
$a = 0$	(b): $x \leq \text{mín}\{\frac{c}{l-l_0}, \frac{l_1-l}{l_1-l_0}\}$	(d): $x \geq \text{máx}\{1 - \frac{c}{l_1-l}, \frac{l_1-l}{l_1-l_0}\}$

Tabla 2.1: Cuadro de decisiones según los valores de c, l, l_0, l_1, y x.

Es decir que se tomará la ruta 0 si $\overline{L_0(t)} < l$, y la ruta 1 si $\overline{L_0(t)} > l$. El retorno en caso de no medir vale entonces:

$$R(s, a = 0) = \text{mín}(\overline{L_0(t)}, l)$$

En el caso en que se mide, el estado del sistema será: $s' = (L_0(t+1), 0)$. Se debe calcular con qué probabilidad $L_0(t+1)$ será l_0, y con qué probabilidad será l_1. Nuevamente, se utilizan las probabilidades de transición, que definen la dinámica del sistema. Es decir que en caso de medir, la ruta 0 estará en el nivel bajo con probabilidad $P(L_0(t+1) = l_0) = e_{l_0} P^{\tau+1} e_{L_{Last}}$ y en nivel alto con probabilidad $P(L_0(t+1) = l_1) = e_{l_1} P^{\tau+1} e_{L_{Last}}$. Obsérvese que estas dos probabilidades son de suma 1, dado que es un proceso markoviano y son los únicos dos estados posibles de la ruta. Se llama por simplicidad $x(t) = P(L_0(t+1) = l_0) = e_{l_0} P^{\tau+1} e_{L_{Last}}$. Se elegirá la ruta 0 con probabilidad x de estar en nivel bajo, mientras que con probabilidad $1 - x$ se encuentra en nivel alto y se tomará la ruta de retardo fijo. El retorno en este caso vale entonces:

$$R(s, a = 1) = c + x(t) * l_0 + (1 - x(t)) * l$$

Se obtiene el cuadro 2.1 de decisiones según los valores de c, l, l_0, l_1, y x. Dado que de estos parámetros el único variable es x, la decisión se puede saber de antemano: dime hace cuánto se midieron las rutas y te diré qué debes medir, primero, y por dónde debes rutear, después. Lo que se obtiene en la tabla 2.1 tiene una interpretación directa en el aspecto siguiente: si x es muy grande o muy pequeño no conviene medir. Si es conocido que la probabilidad de que la ruta 0 esté en el estado bajo es muy baja o muy alta, ya es sabido por dónde conviene rutear, sin necesidad de incurrir en un costo para obtener mayor información.

En cada época de decisión, entonces:

- El sistema está en un estado $s = S_t$.
- Elegimos si medir o no. Para eso se resuelve el problema de optimización:

$$a = \arg\min_a R(s, a)$$

- Con la información obtenida si se mido, y si no con la evolución del sistema al estado $s = (L_{Last}, \tau + 1)$, se elige la $ruta$. Para eso se resuelve el problema de optimización:

$$ruta = \arg \text{mín}(\mathbb{E}[L_0(t)], l)$$

El último paso merece ser descrito. Siendo que en esta época se realizó la medida, de haber medido se conoce con probabilidad 1 al estado del sistema, y si no se midió se conoce la estimación de $\overline{L_0}$ según la matriz de transiciones y el valor de τ. Se toma la decisión de ruteo considerando que hasta la época siguiente las rutas seguirán en ese estado y minimizando el RTT para esta época, por lo que se elegirá la ruta cuyo valor esperado de RTT es menor. La política de ruteo en toda la formulación del problema como MDP seguirá esta lógica, es decir se realiza época a época, sin considerar el futuro.

De igual forma que con una ruta de retardo fijo y otra con retardo variable, se puede generalizar el problema a varias rutas con retardos variables. Sigue siendo un problema de optimización lineal, que tiene una solución cerrada y que se puede calcular (ver apéndice A).

Dos rutas con retardos variables

Se emplea este algoritmo en un ejemplo sencillo de dos rutas variables propuesto en [41].

Los valores para la ruta 0 son $l_0 = 0{,}5$, $l_1 = 2$, $c_0 = 0{,}05$, $P_0 = \begin{bmatrix} 0{,}7 & 0{,}3 \\ 0{,}3 & 0{,}7 \end{bmatrix}$; y para la ruta 1: $l_0 = 1$, $l_1 = 3$, $c_1 = 0{,}15$, $P_1 = \begin{bmatrix} 0{,}9 & 0{,}1 \\ 0{,}1 & 0{,}9 \end{bmatrix}$.

Se obtiene una división del mapa de estados que se puede apreciar en la imagen 2.4, donde se indica la acción a realizar según el estado del sistema. El estado del sistema se formuló a partir de $x_i = P(L_i(t+1) = l_0)$ la probabilidad de encontrar a la ruta i en nivel bajo.

Obsérvese que se ha considerado la acción óptima desde el punto de vista del retorno inmediato esperado para tomar la decisión de medida, sin tomar en cuenta que el conocimiento aportado por la medida afecta decisiones futuras (dado que aporta información).

Esta formulación es conocida como la política miope, y se pueden obtener muy buenos resultados utilizando esta política, que entre otras ventajas es escalable: es la resolución de un problema de optimización lineal. Por otro lado, hay casos en que la política miope no ofrece buenos resultados, como se describe en [48].

Otro problema a pensar en esta introducción, es que no siempre son razonables las suposiciones markovianas para el comportamiento de las rutas. Si bien este capítulo se centra en la formulación markoviana, que hay una asunción de modelo no debe perderse de vista.

En lo que queda del capítulo se complejizará el problema planteado bajo la óptica markoviana, se encontrará una forma de obtener la solución óptima, y se analizará la complejidad numérica de los algoritmos que permiten alcanzar esta solución. En el capítulo siguiente se propondrá una solución aproximada más escalable.

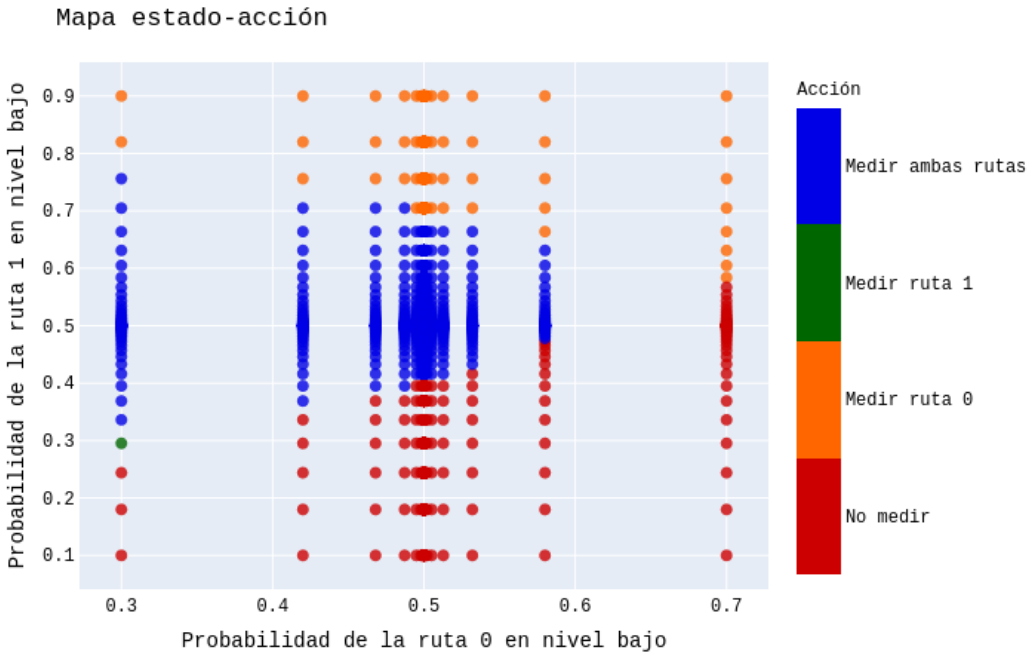


Figura 2.4: Caso de dos rutas con dos estados cada una. Mapeo de estados posibles – acciones. Obsérvese que cuanto más probable es que la ruta 0 esté en nivel bajo, menos se medirá. Por otro lado, en la zona céntrica, en que se tiene menor probabilidad de conocer los estados de las rutas, se opta por medir ambas rutas. Así se podría continuar el análisis, incluso encontrando de forma cerrada los bordes de cada región.

2.1.3. Formulación del problema como MDP

Se vuelve al problema general de varias rutas con varios niveles de retardo posibles, con el fin formularlo como un Proceso de Decisión Markoviano (MDP), y buscar su política óptima usando técnicas de programación dinámica bien conocidas.

Recordando: se desea minimizar el tiempo de ida y vuelta entre dos nodos, asumiendo medidas con costo que aportan información del sistema ([41], [48]). Considere que el tiempo está ranurado y las transmisiones de paquetes y mediciones se realizan en cada ranura temporal. Sea el conjunto $\mathcal{N} = \{1...N\}$ de posibles rutas entre los nodos como en la figura 2.5.

Cada ruta i tiene su retardo modelado como una cadena de Márkov de tiempo discreto con K^i niveles posibles, y probabilidades de transición P^i. Se llamará $L^i(t)$ a la variable aleatoria que representa el retardo en la ruta i y en tiempo t, y L^i_{Last} al último estado medido para esa ruta.

El agente debe elegir si medir o no medir cada ruta. El conjunto de acciones posibles $\mathcal{A}$ está compuesto por vectores de dimensión N que representan la concatenación $A_t = \{A^i_t\}^{i=1...N}$ de las acciones A^i_t para cada ruta i en tiempo t. Sea a una acción genérica, compuesta de las acciones a_i en cada ruta i, que puede valer $a_i = \{0; 1\}$, siendo $a_i = 0$ cuando no se mide, y $a_i = 1$ cuando se mide. Al medir se incurre en un costo c^i, y se llamará al vector de costos $C = \{c^i\}^{i=1...N}$.

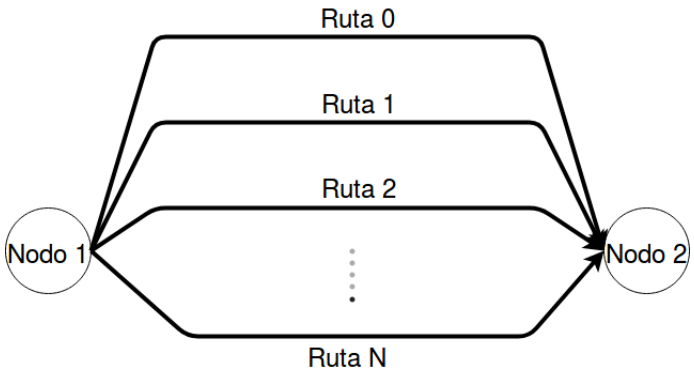


Figura 2.5: Problema general: N rutas con hasta K^i posibles retardos (estados) cada una.

El sistema cuenta con $\mathcal{S}$ posibles estados, siendo este un valor discreto. El estado del sistema en t $s(t)$ se encuentra definido por el último valor medido y la cantidad de intervalos temporales desde que se realizó esa medida, es decir el estado s^i de la ruta i vale $s^i(t) = (L^i_{Last}, \tau^i)$. Luego de un tiempo largo sin medir $\tau^i_{\text{máx}}$ las probabilidades de transición P^τ y $P^{\tau+1}$ difieren en menos de un ϵ, para cuyo caso, se considera que valen $P^{\tau_{\text{máx}}} = P^{\tau_{\text{máx}}+i}$, $\forall i \geq 0$. Esta matriz $P^{\tau_{\text{máx}}}$ representa una aproximación de la matriz de transiciones estacionaria P^∞. Se llama a este tiempo $\tau^i_{\text{máx}}$, y se obtiene entonces un conjunto discreto y además finito de estados posibles.

En cada ranura de tiempo, el sistema se encuentra en estado $s = \{s^i\}_{i=1...N}$, y el tomador de decisiones puede tomar las acciones $a^i = \{0; 1\}$ para las N rutas, como se observa en la figura 2.6:

- el estado del sistema evoluciona según el proceso markoviano discreto para cada ruta
- se toman las decisiones de medición, incurriendo en un costo y obteniendo información del estado actual para las rutas medidas
- el tomador de decisiones elije el camino j de menor retardo esperado, y obtiene un retorno por rutear $L^j_r(t)$

Las transiciones entre estados contiguos $s^i \rightarrow s^{i\prime}$ dependen únicamente del estado actual s^i, y de la acción a^i. Dado un estado s, el tomador de decisiones puede estimar para cada ruta i su creencia del estado de esta ruta, que se llamará: $b_{L^i_{Last},x}(s^i) = e^T_{L^i_{Last}} P^{i\tau^i+1} e_x$, y es la probabilidad de que la ruta i se encuentre en el nivel x desde el estado $s^i = (L^i_{Last}, \tau^i)$. Los vectores columna $e_{L^i_{Last}}$ y e_x tienen 0 en todos los valores excepto el correspondiente a L^i_{Last} y x respectivamente. En caso de no medir se pasa con probabilidad 1 a un único estado $s^{i\prime} = (L^i_{Last}, \tau^i + 1)$. Las probabilidades de transición valen:

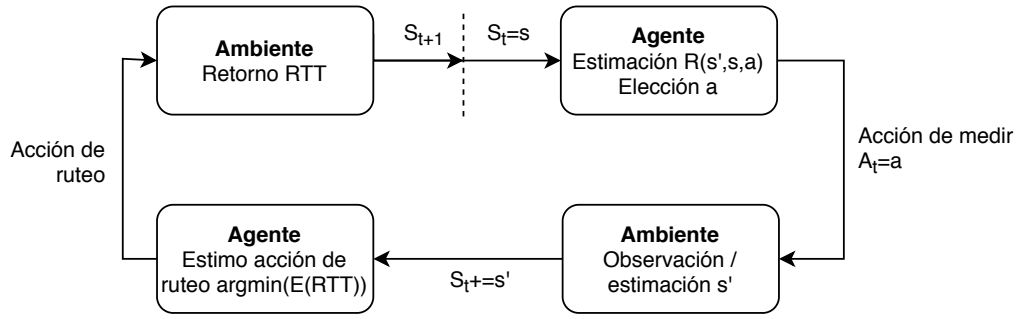


Figura 2.6: Evolución del proceso de decisión markoviano. Hay dos decisiones a tomar: las rutas a medir, que se toman considerando al MDP, y la ruta a elegir, para lo que simplemente se estima (o conoce si se midió) el RTT esperado en cada ruta.

$$p(s^{i\prime}|s^i, a^i) = \begin{cases} 1 & \text{si } a^i = 0,\ s^{i\prime} = (L^i_{Last}, \tau^i + 1) \\ b_{L^i_{Last},x}(s^i) & \text{si } a^i = 1 \text{ y } s^{i\prime} = (x, 0) \\ 0 & \text{en otro caso} \end{cases} \tag{2.1}$$

Observe que las probabilidades de transición cumplen con las hipótesis markovianas (suma 1 y memoria resumida en el estado inmediatamente precedente). En la figura 2.7 se aprecia como son las transiciones entre estados para una ruta con dos niveles posibles. Al medir se vuelve a alguno de los estados $(L_0, 0)$ o $(L_1, 0)$, y de no medir se va con probabilidad 1 del estado $s = (L_{Last}, \tau)$ al estado siguiente $s' = (L_{Last}, \tau + 1)$.

El objetivo del tomador de decisiones es minimizar el costo acumulado de medir y del retardo obtenido en la elección de las rutas. El problema consiste en elegir la ruta más rápida, para lo cual se desea minimizar la incertidumbre respecto del estado de las mismas, mientras que se busca minimizar el costo de medida, por lo que se mantendrá igualmente un cierto grado de incertidumbre. El costo de medida tiene entonces un peso importante en las decisiones. Con un costo de medida de 0 las rutas serán siempre medidas, mientras que con un costo lo bastante alto nunca se realizarán medidas. En este sentido es que es una variable importante a calibrar con cierto cuidado del problema de optimización. Hay reflexiones y pruebas en este sentido en el capítulo de 5.

Para la decisión de ruteo, como se realiza época a época, se considerará el mínimo retardo inmediato esperado:

$$D(s) = \min_i \mathbb{E}[L^i(t)|(L^i_{Last}, \tau^i(t))]$$

Definimos al retorno inmediato esperado como la suma del mínimo retardo inmediato esperado $D(s)$ y del costo de medición para el estado s:

$$R(s, a) = D(s) + \sum_i c^i * a^i \tag{2.2}$$

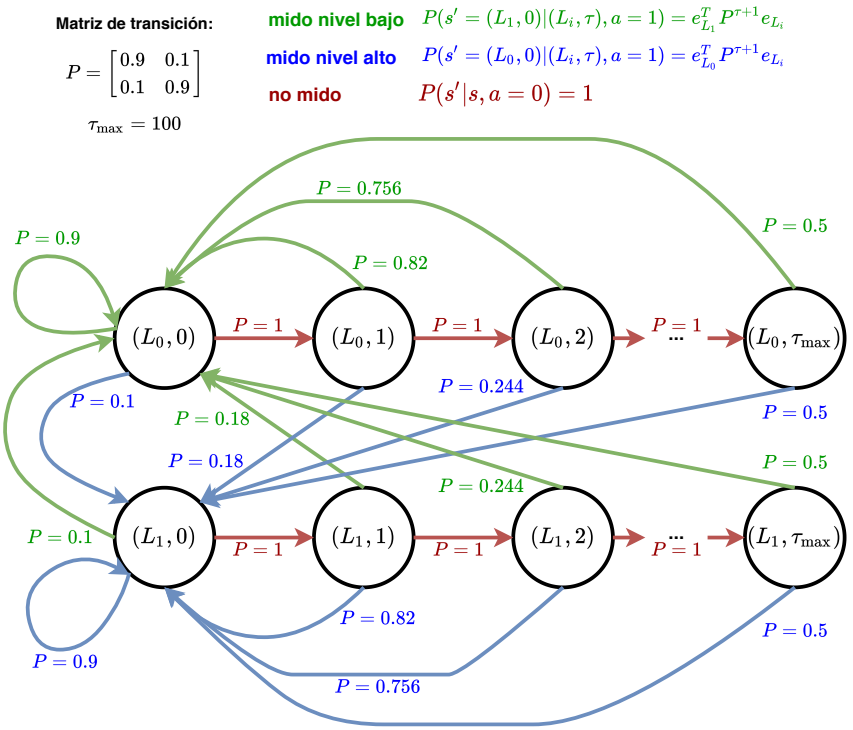


Figura 2.7: Evolución de la cadena de Márkov con la que se representa el estado de una ruta con dos niveles $L_0 < L_1$ posibles. Se consideró una matriz de transición sencilla de ejemplo, que es simétrica (no tiene porqué serlo), y un valor de $\tau_{\text{máx}} = 100$ que es razonable para dicha matriz. Los vectores columna de la imagen valen: $e_{L_0} = \begin{bmatrix} 1 & 0 \end{bmatrix}$, $e_{L_1} = \begin{bmatrix} 0 & 1 \end{bmatrix}$.

Se define al retorno acumulado con descuento desde el instante t y hasta el final (recuerde que se está en un problema de horizonte infinito) como:

$$G_t = \sum_{k=0}^{\infty} \rho^k R_{t+k} \tag{2.3}$$

Observar que G_t es una variable aleatoria. El objetivo del tomador de decisiones será, para medir, en el tiempo t y el estado s seguir la política π^* que minimiza la esperanza del retorno acumulado $\mathbb{E}_{\pi^*}[G_t|S_t = s]$. Luego elegirá la ruta i de menor retardo esperado.

2.2. Función de Valor y solución exacta

Se anhela entonces minimizar la combinación de las medidas realizadas y el retardo en las rutas elegidas, desde el momento y estado presente e incluyendo al futuro.

Para los MDP en tiempo discreto, se llama $V_\pi(s)$ a la función de valor del estado s (*state-value function*), que es el retorno esperado de seguir la política π:

$$V_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \rho^k R_{t+k}|S_t = s] \tag{2.4}$$

Obsérvese que el objetivo es hallar la política que minimiza la función de valor. Separando los términos correspondientes al retorno esperado inmediato y la función de valor para estados futuros, se obtiene la *ecuación de Bellman para* V_π:

$$V_\pi(s) = \sum_a \pi(a|s) \sum_{s'} p(s'|s,a)[R(s,a) + \rho V_\pi(s')] \tag{2.5}$$

Se desea encontrar la política π^* tal que $V^*(s) = \min_\pi V_\pi(s) \forall s \in \mathcal{S}$. Para esto se emplea la *ecuación de optimalidad de Bellman* para la función de valor de estado [47]:

$$V^*(s) = \min_a \sum_{s'} p(s'|s,a)[R(s,a) + \rho V^*(s')] \tag{2.6}$$

De manera análoga, se define la función de valor de estado-acción (*action-value function*) para la política π:

$$Q_\pi(s,a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a] = \sum_{s'} p(s'|s,a)[R(s,a) + \rho \max_{a'} Q_\pi(s',a')] \tag{2.7}$$

Los algoritmos llamados de programación dinámica ([51], [47]) utilizan la ecuación de optimalidad de Bellman para converger a la política óptima:

$$V^{n+1}(s) = \min_a \sum_{s'} p(s'|s,a)[R(s,a) + \rho V^n(s')] \tag{2.8}$$

Luego la política π es mejorada a π' con:

$$\pi'(s) = \arg\min_a \sum_{s'} p(s'|s,a)[R(s,a) + \rho V_\pi(s')] \tag{2.9}$$

Existen métodos recursivos para hallar la política y la función de valor óptimas o ϵ-óptimas (óptima en tiempo infinito, llega a una V tal que $|V^*(s) - V(s)| \leq \epsilon$, $\forall s \in \mathcal{S}$ en tiempo finito, con ϵ tan pequeño como se desee). Los métodos se basan en combinar dos aspectos: la evaluación de una política (*policy evaluation*) y su mejora (*policy improvement*). La evaluación de una política π se hace utilizando la convergencia de la ecuación de Bellman (eq. (2.8)), comenzando con un valor cualquiera de V, usualmente $V(s) = 0 \forall s \in \mathcal{S}$, hasta la convergencia de V_π (eq. (2.5)). La mejora de una política π se realiza eligiendo las acciones que reportan mayor beneficio que $V_\pi(s)$, generalmente de manera *greedy* como se presentó en la eq. (2.9).

Se presenta a continuación un algoritmo de la familia *Policy iteration* llamado *modified policy iteration* y propuesto por Puterman en 1978 [52] [51], que sigue la lógica siguiente:

$$\pi_0 \xrightarrow{E} V_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} V_{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \ldots \xrightarrow{I} \pi^* \xrightarrow{E} V^*$$

En las flechas se indica por E la evaluación de la política: utilizar π en la ecuación (2.8) hasta la convergencia; y por I la mejora de una política: cambiar π_n a π_{n+1} a través de la ecuación (2.9).

Este algoritmo recursivo parte de un valor inicial de π_0 aleatorio y $V_0(s) = 0$ (por ejemplo), y así hasta la convergencia cuando $|V_{n+1}(s) - V_n(s)| \leq \epsilon \; \forall s \in \mathcal{S}$. El algoritmo converge a tasa polinómica en el espacio de estados según plantea Puterman en [51]. En vez de utilizar la función de valor de estado $V(s)$ se puede realizar la recurrencia sobre la función de valor de estado-acción $Q(s, a)$ (*action-value function* definida en (2.7)) y su respectiva ecuación de optimalidad de Bellman, verificándose la relación $V_\pi(s) = \text{máx}_a \, Q_\pi(s, a)$. Estos algoritmos tienen múltiples versiones, el lector interesado encontrará más detalles en [47]. La versión descrita se encuentra detallada en el algoritmo 1.

A pesar de obtener la solución exacta con el mismo, no es tan sencilla su aplicación. Esta formulación debe lidiar con el famoso "*curse of dimensionality*", que en resumidas cuentas hace que no escale el algoritmo para varias rutas con varios niveles. El "*curse of dimensionality*" es el incremento del espacio de estados y acciones que se da en problemas de programación dinámica (y en general en el manejo de datos de varias dimensiones), dado que se deben recorrer todos los estados y acciones posibles, y esto impide su resolución en tiempo real. En nuestro problema particular, la cantidad de rutas, la cantidad de niveles y el valor de $\tau_{\text{máx}}$ definen la cantidad de estados posibles.

A continuación se verá su aplicación al ejemplo presentado de dos rutas con dos niveles cada ruta. Esto servirá para ejemplificar el impacto que tiene considerar al futuro en la toma de decisiones, lo que nos conducirá a enfrentarnos al problema de dimensionalidad mencionado.

2.2.1. Continuando con el ejemplo: considerar al futuro y hallar la solución óptima

Se va a retomar el ejemplo 2.1.2, esta vez para comparar el mapa de políticas para el algoritmo miope y la solución óptima usando el *modified policy iteration*. Con miope se hace referencia a la política que optimiza sobre la esperanza del retorno inmediato, sin tomar en cuenta el futuro. Es decir utilizando sólo el primer paso de la ecuación (2.8), y con $V^0(s) = 0$, $\forall s \in \mathcal{S}$.

En la figura 2.8 se observa la diferencia que se obtiene por no tomar en cuenta al futuro. El peso que tendrá el futuro está directamente relacionado a ρ, el factor de descuento, dado que la función de valor h pasos después V^h está multiplicada por ρ^h.

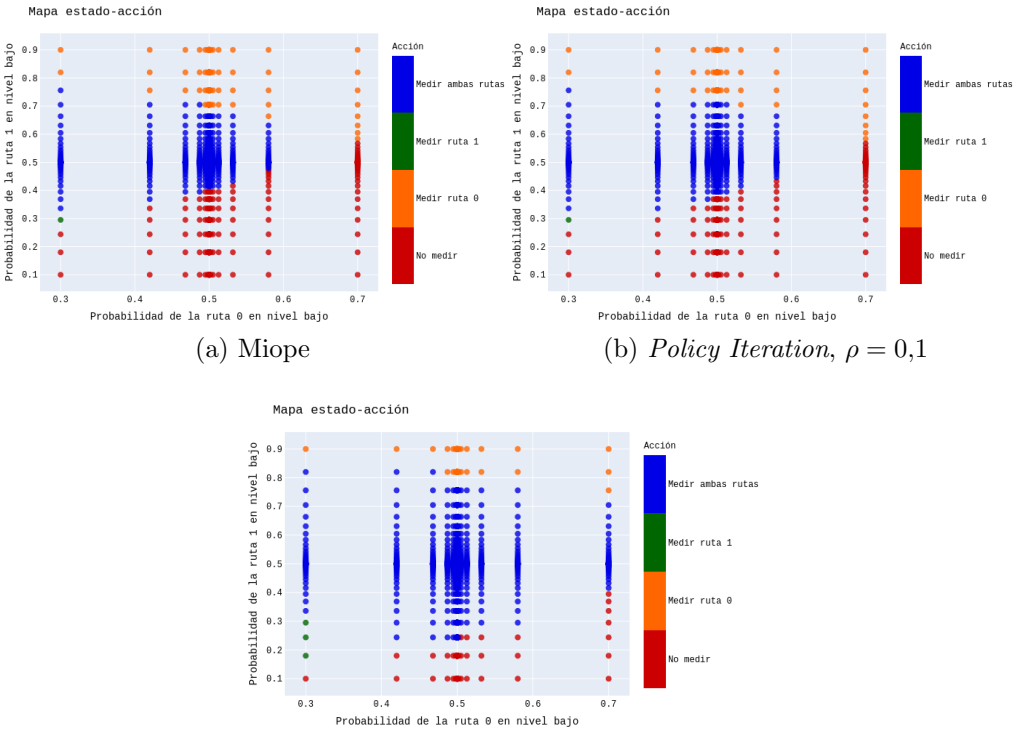
El factor de descuento no sólo garantiza la convergencia de los algoritmos, a través de hacer converger la esperanza del retorno acumulado con descuento (eq. (2.3)), sino que también tiene un rol en el peso que se le desea asignar al futuro: cuanto más alto el valor de ρ, más nos interesa el futuro.

Algorithm 1 Policy Iteration (usando *iterative policy evaluation*), de [47] Sección 4.3

Inicialización
$V(s) = 0 \forall s \in \mathcal{S}$,
$\pi(s) \in \mathcal{A}$ aleatorio $\forall s \in \mathcal{S}$
function POLICY EVALUATION($\mathcal{S}, \pi, V$)
    Repito
    $\Delta \leftarrow 0$
    for cada $s \in \mathcal{S}$ **do**
        $v \leftarrow V(s)$
        $V(s) \leftarrow \sum_{s'} p(s'|s, \pi(s))[R(s, a) + \rho V(s')]$
        $\Delta \leftarrow \text{máx}(\Delta, |v - V(s)|)$
    end for
    Hasta que $\Delta < \epsilon$, ϵ algún valor pequeño.
end function
function POLICY IMPROVEMENT($\mathcal{S}, \pi, V$)
    policy-stable $\leftarrow$ *True*
    for cada $s \in \mathcal{S}$ **do**
        old-action $\leftarrow \pi(s)$
        $\pi(s) \leftarrow \arg \text{máx}_a \sum_{s'} p(s'|s, \pi(s))[R(s, a) + \rho V(s')]$
        if *old-action* $\neq \pi(s)$ **then**
            policy-stable $\leftarrow$ False
        end if
    end for
    if *policy-stable* **then**
        parar y retornar $V \leq V^*$ y $\pi(s) \leq \pi^*(s)$
    else
        ir a POLICY EVALUATION($\mathcal{S}, \pi, V$)
    end if
end function

Suele considerarse, sobre todo en las formulaciones económicas de estos problemas, que un retorno inmediato vale más que el mismo retorno en el futuro, y por eso el valor de $\rho < 1$. La política miope es óptima de considerar que $\rho = 0$, y se aleja más de la política óptima al aumentar el valor de ρ. Para el ejemplo, se utilizó $\rho = 0{,}9$, que es un valor razonable y usado en la literatura, y se compara con $\rho = 0{,}1$, que es un valor bajo para problemas de este tipo, pero que permite analizar el peso del futuro y comparar las políticas obtenidas.

A continuación se verá cómo se complejiza el problema al agregar rutas y niveles posibles en el MDP, y en el próximo capítulo se propondrá una solución aproximada que permite mayor escalabilidad al problema de ruteo y medición en su formulación como un MDP.



(a) Miope

(b) *Policy Iteration*, $\rho = 0{,}1$

(c) *Policy Iteration*, $\rho = 0{,}9$

Figura 2.8: Comparación de las políticas obtenidas con el algoritmo de *modified policy iteration* y usando la política miope, con un valor de $\rho = 0{,}9$ y un valor de $\rho = 0{,}1$. Nótese que las políticas óptima y miope son más similares cuando el futuro tiene menor peso (ρ menor).

2.3. Comentarios sobre la complejidad numérica de la solución por programación dinámica

La complejidad numérica del algoritmo de *modified policy iteration* (2.8) crece muy rápido al incrementar la cantidad de rutas.

Recordando, el estado del sistema se representa como $s = (s_1, s_2, \ldots, s_N)$, con $s_i = (L^i_{Last}, \tau^i)$. Como fuera mencionado, para obtener una cantidad finita de estados (y poder aplicar el algoritmo, que recorre todos los estados), se fija en $\tau^i_{\text{máx}}$ un máximo de *time-slots* (épocas de decisión) sin medir para cada ruta, tras el cual el estado permanece en $(L^i_{Last}, \tau^i_{\text{máx}})$ de no ser medido. La idea detrás de esto es que dada la matriz P^i de esa ruta, $P^{i\tau^i_{\text{máx}}} = P^{i\tau^i_{\text{máx}}+1}$, y luego de $\tau_{\text{máx}}$ la cadena llegó a un estado muy cercano al valor estacionario. Según los valores de la matriz P^i, se puede hallar un $\tau^i_{\text{máx}}$ suficientemente grande para el que esta aproximación sea razonable. Esta aproximación además resulta en que el espacio de estados $\mathcal{S}$ sea finito.

Si se asume además que cada ruta tiene un conjunto finito K^i de niveles posibles, como hasta ahora, la cantidad de estados posibles para cada ruta es: $S^i = K^i * \tau^i_{\text{máx}}$. La cantidad de estados posibles para el conjunto del problema es:

$$S = \prod_{i=1}^{N} \tau_{\text{máx}}^{i} K^{i}$$

Si se considera, por simplicidad y solo a modo de ejemplo, que las rutas tienen la misma cantidad de niveles posibles y el mismo $\tau_{\text{máx}}$, se observa que la cantidad de estados S crece exponencial con la cantidad de rutas:

$$S = (\tau_{\text{máx}} K)^{N}$$

Por otro lado, el espacio de acciones posibles vale $A = 2^N$, dado que para cada ruta tengo dos acciones posibles (medir o no medir).

El algoritmo de *modified policy iteration* tiene complejidad polinomial en la cantidad de estados, y lineal en la cantidad de acciones, como se prueba en [53]. Sin embargo el espacio de estados crece exponencialmente con la cantidad de rutas, así como el espacio de acciones. Esto hace que la complejidad del algoritmo crezca exponencialmente sobre la cantidad de rutas, por lo que rápidamente deja de ser factible su cálculo. Por otro lado, la cantidad de niveles y el valor de $\tau_{\text{máx}}$ también inciden en el espacio de estados, aunque con tasa polinómica. Finalmente, el valor de ρ también participa en la cota de la velocidad de convergencia, dado que para un ρ muy pequeño, el algoritmo convergerá más rápido en la evaluación de política, y vice-versa.

Un ejemplo sencillo de cuatro rutas de cuatro niveles alcanza para probar el punto. Considerando $\tau_{\text{máx}} = 50$, que aunque un poco justo es razonable para matrices como ser $P = \begin{bmatrix} 0{,}9 & 0{,}1 \\ 0{,}1 & 0{,}9 \end{bmatrix}$, hay una cantidad de estados: $S = (50 * 4)^4 = 1{,}6 \cdot 10^9$.

Sumado a este inconveniente, el cálculo de estos algoritmos se realiza una primera vez hasta la convergencia de π^* y V^*, y luego se utilizan los resultados de la política óptima para un estado dado del sistema. Es decir que no funcionan en línea sino que se obtiene la política óptima para todos los estados del sistema, y luego se utiliza esta política. Sin embargo no todos los estados del sistema son igualmente visitados, muchos de ellos tal vez nunca lo sean.

En el próximo capítulo se verá cómo reducir el cálculo de la función de valor a los estados visitados y sus vecinos, obteniendo una aproximación a la función de valor óptima mediante un algoritmo más escalable.

Capítulo 3

Aproximación por Horizonte Errante

> Ella está en el horizonte. Me acerco dos pasos, ella se aleja dos pasos. Camino diez pasos y el horizonte se corre diez pasos más allá. Por mucho que yo camine, nunca la alcanzaré. ¿Para que sirve la utopía? Para eso sirve: para caminar.
>
> Fernando Birri

3.1. El algoritmo de Horizonte Errante

Vimos que el problema planteado como un MDP de horizonte infinito tiene solución, pero que el cálculo de la solución óptima implica su estimación para todos los estados del sistema, y esto no escala. Se propone en este capítulo una aproximación que considera un horizonte finito época de decisión a época de decisión, y que converge a la solución óptima estacionaria.

Recuerde que se desea minimizar la función de valor $V(s) = \mathbb{E}[G_t|S_t = s]$, donde $G_t = \sum_{k=0}^{\infty} \rho^k R_{t+k}$ es la suma infinita de los retornos acumulados (eq. (2.3)). Si se acota esta suma a un intervalo $k \in [0, T]$, se tiene el problema equivalente de horizonte finito. Esta formulación cuenta con una desventaja: la política obtenida no tendrá porqué ser estacionaria, y por lo tanto, resultará en acciones diferentes para tiempos diferentes pasando por los mismos estados. En particular, lo más cerca se encuentre el tiempo de T, menos acertada será la solución obtenida, dado que a través del factor de descuento ρ se está indicando menor importancia a las decisiones futuras (se ponderan por un factor $\rho^k < 1$). Otra forma de verlo es que sólo las primeras acciones estarán cerca de la política óptima, y a medida que pasa el tiempo se irá alejando de esta (que vale la pena recordar, es estacionaria para el problema de horizonte infinito).

En [54], los autores describen una solución conocida como de horizonte errante (*receding horizon* en inglés[1]). Demuestran además que esta solución converge a la

[1]Si bien no está clara la traducción del nombre original en inglés *receding horizon*, y se encontraron en la literatura otros nombres como horizonte "móvil" o "deslizante", se ha elegido "errante", por encontrarla más a gusto del autor.

solución óptima del problema de horizonte infinito. Esta propuesta es conocida en el área de control de sistemas dinámicos como *Model Predictive Control* (control por modelo predictivo), que se desarrolla como una variante con ventana dinámica de los algoritmos LQR discretos y de horizonte finito[2].

El algoritmo, en su formulación discreta, consiste en las siguientes etapas:

- En el tiempo t y para el estado s_t hallar la solución de horizonte finito para $[t; t+H-1]$, considerando restricciones actuales y futuras.
- Tomar solamente la primera acción de la solución hallada.
- Medir o evaluar el estado s_{t+1} del sistema en el tiempo $t+1$.
- Volver al primer ítem, resolviendo el problema de optimización con horizonte finito sobre $[t+1; t+H]$, empezando desde el nuevo estado $s' = s_{t+1}$.

Haciendo tender $H \to \infty$, se converge a la función de valor óptima, y se pasa a estar en el caso de los algoritmos de programación dinámica, siendo que se visitan todos los estados del sistema. En particular para el paso H se puede encontrar una cota al error ϵ: $\epsilon \leq \frac{\rho^H sup|V^n(s)|}{1-\rho}$.

La aplicación de esta aproximación al algoritmo de *value-iteration* fue presentada en [48], en un artículo en colaboración con los grupos de Toulouse y Brest. La idea es aplicar la acción óptima resultante de tomar en cuenta el futuro, pero solamente hasta cierto punto. Se aplica la acción sólo sobre ese primer estado, y luego se vuelve a calcular la política de horizonte H para el estado siguiente. Esto resulta en una política estacionaria, a diferencia de la obtenida con horizonte finito. En los problemas de horizonte finito se toman las K acciones a partir del estado inicial, por lo que pasando dos veces por un mismo estado, en épocas distintas, se pueden obtener acciones diferentes.

Para el horizonte errante, el valor de H (horizonte) hasta el cual se considera el futuro depende de varios elementos, que serán objeto de análisis particular más adelante. Se explica a continuación cómo se aplica al MDP el algoritmo de *receding horizon*, a partir de la ecuación de optimalidad de Bellman (ecuación (2.6)) para la función de valor de estado.

Se usan los primeros pasos del algoritmo de *value-iteration* (ecuación (2.5)):

$$V^{n+1}(s) = \min_a \left[\sum_{s'} p(s'|s,a)[R(s,a) + \rho V^n(s')] \right]$$

Se comienza con $V^0 \equiv 0$, $\forall s \in \mathcal{S}$.

$$V^1(s) = \min_a \left[\sum_{s'} p(s'|s,a)[R(s,a) + \rho V^0(s')] \right]$$

[2]Los algoritmos *Linear-Quadratic Regulators* minimizan una función de costo para problemas en ecuaciones diferenciales lineales y en que el costo es una función cuadrática.

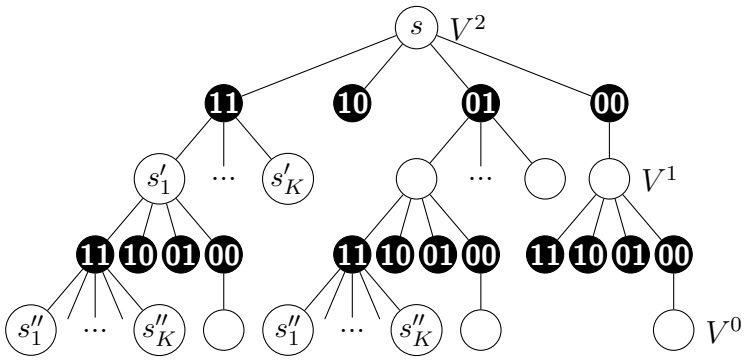


Figura 3.1: Ilustración de los estados visitados al avanzar en el algoritmo de horizonte errante, con $N = 2$ rutas, K niveles para cada ruta y un horizonte de $H = 2$. Los nodos negros representan las posibles acciones a tomar en cada estado (en blanco). Se hace un abuso en la notación para referirnos a un posible estado que está dos pasos después como s''.

Este primer paso da como resultado la política miope, que sólo toma en cuenta el retorno inmediato. Es equivalente a utilizar un $\rho = 0$, que también evita considerar el futuro, y fue presentada en el ejemplo sencillo del capítulo 2. Con esta política se obtienen muy buenos resultados en ciertos contextos, como ya fue anticipado y como se verá en los próximos capítulos.

Si se itera hasta el paso H, se obtiene la función de valor V^H y la política π^H para el horizonte H:

$$V^H(s) = \min_a \left[\sum_{s'} p(s'|s,a)[R(s,a) + \rho V^{H-1}(s')] \right] \tag{3.1}$$

$$\pi^H(s) = \arg\min_a \left[\sum_{s'} p(s'|s,a)[R(s,a) + \rho V^{H-1}(s')] \right] \tag{3.2}$$

Es posible utilizar la función de estado-valor $Q(s,a)$ (definida en eq. (2.7)) para obtener $V(s)$ y $\pi(s)$ de manera *greedy*:

$$V^H(s) = \min_a Q^H(s,a)$$

$$\pi^H(s) = \arg\min_a Q^H(s,a)$$

Es decir que se deben recorrer las ramificaciones desde el estado original hasta todos los estados posibles en H pasos, que se llamará $s^{H\prime}$. Para esto se construye un árbol de posibles estados, como se puede ver en la imagen 3.1, de [48]. En este caso por motivos de presentación se redujo a dos rutas (cuatro acciones posibles), K niveles en cada ruta y $H = 2$. Los círculos negros representan las acciones, mientras que los blancos son los estados alcanzables.

Una vez construido el árbol, se pueden recorrer los H pasos hacia atrás desde el conjunto de los $s^{H\prime}$ estados posibles. La función de valor de estado para estos será $V^0(s^{H\prime}) = 0$.

Este método recursivo es presentado en el algoritmo 2. El mismo es recursivo y consta de dos funciones principales: **RecedingV** que evalúa la función de valor de estado $V^h(s)$ de manera recursiva, y que va actualizando los valores de $V^h(s)$ para los estados recorridos desde $h = H$ y hasta 0. La otra función **MAIN** evalúa para la primera ramificación los posibles valores de V^H y devuelve $V^{H*}(s) = \text{mín}_a\, Q^{H*}(s,a)$ y $\pi^{H*}(s) = \arg\text{mín}_a\, Q^{H*}(s,a)$.

Algorithm 2 Receding Horizon

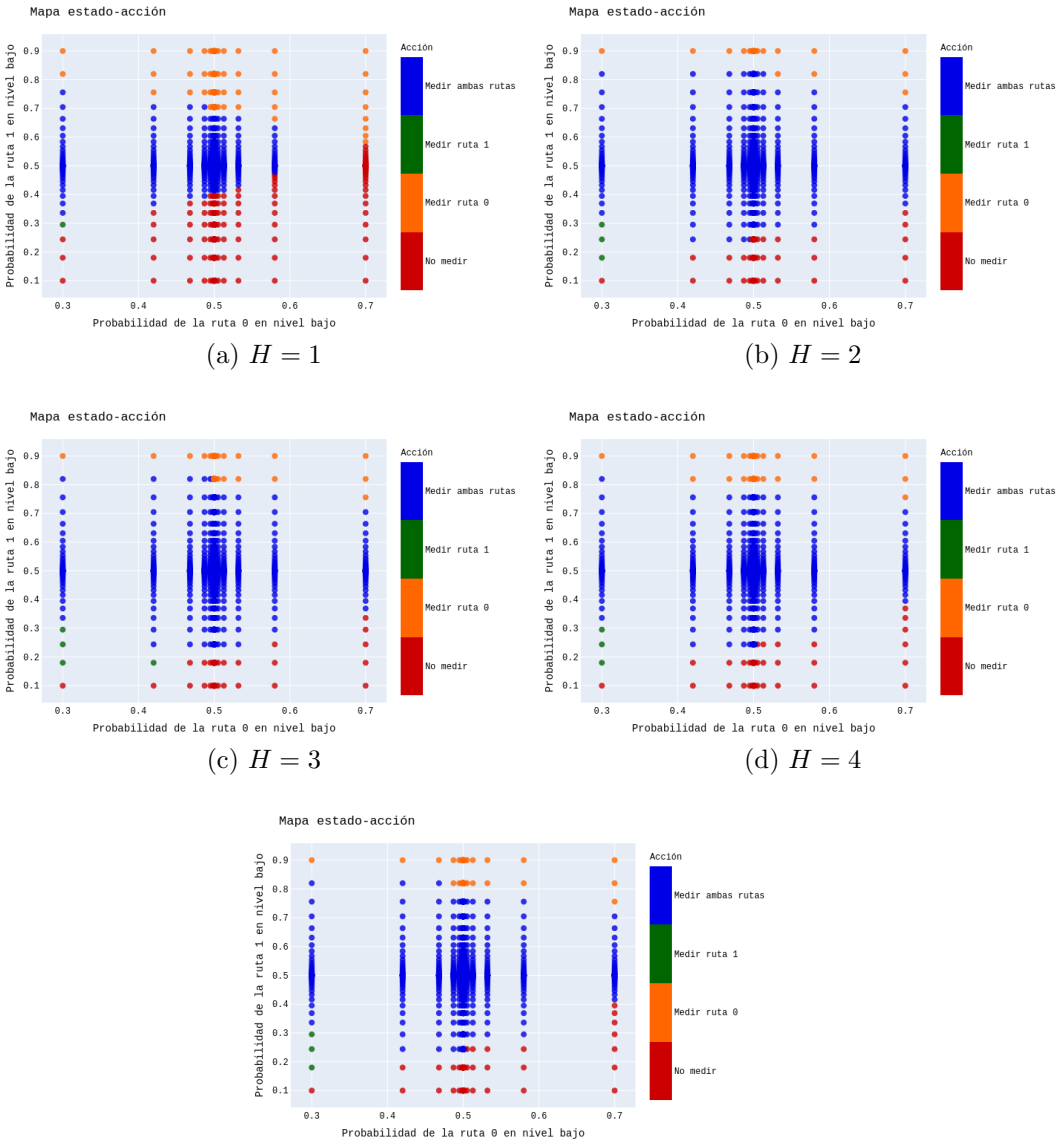
function RecedingV(s, h)
  entradas: estado s, profundidad h
  if $h = 0$ **then**
    $V^0(s) \leftarrow 0$
    return $V^0(s)$, $\pi(s) = a$ con a cualquiera
  else
    for cada estado s' alcanzable desde s **do**
      $V^{h-1}(s'), \pi^{h-1^*} \leftarrow$ RecedingV(s',$h-1$)
    end for
    $R(s,a) \leftarrow$ resolver Eq. (2.2) para cada acción a
    $V^h(s) \leftarrow \text{mín}_a \sum_{s'} p(s'|s,a)\left[R(s',s,a) + \rho V^{h-1}(s')\right]$
    $\pi^{h^*}(s) \leftarrow \arg\text{mín}_a \sum_{s'} p(s'|s,a)\left[R(s',s,a) + \rho V^{h-1}(s')\right]$
    return $V^h(s), \pi^{h^*}(s)$
  end if
end function
function Main(s, H)
  entradas: estado actual s, profundidad del horizonte $H > 0$
  $V^H(s), \pi^{H^*}(s) \leftarrow$ RecedingV(s,H)
  return $\pi^{H^*}(s)$
end function

Fue presentado un algoritmo con el que se logra una aproximación tan buena como se quiera a la solución óptima para procesos markovianos (alcanza con agrandar H), y con posibilidades de escalar sobre problemas más complejos. Se verá a continuación un ejemplo para afianzar lo planteado hasta el momento, y luego se analizará la complejidad numérica que implica esta solución aproximada, así como algunas ideas para seguir potenciando esta solución.

3.2. Continuando con el ejemplo de dos rutas con dos niveles.

Se sigue trabajando sobre el ejemplo presentado en 2.1.2, que es un caso de dos rutas con dos niveles utilizado en [41].

Recuérdese brevemente las características de cada ruta. Los valores para la ruta 0 son $l_0 = 0{,}5$, $l_1 = 2$, $c_0 = 0{,}05$, $P_0 = \begin{bmatrix} 0{,}7 & 0{,}3 \\ 0{,}3 & 0{,}7 \end{bmatrix}$; y para la ruta 1: $l_0 = 1$,



(a) $H = 1$ (b) $H = 2$

(c) $H = 3$ (d) $H = 4$

(e) *Policy Iteration*

Figura 3.2: Mapa de estado-acción para las políticas óptima (con *policy iteration*), y de horizonte errante con $H = 1 \ldots 4$. La política de horizonte errante con $H = 1$ es la política miope. Los estados están representados por la probabilidad de estar en nivel bajo, $x_i = P(\text{ruta i en } l_0)$.

$l_1 = 3$, $c_1 = 0{,}15$, $P_1 = \begin{bmatrix} 0{,}9 & 0{,}1 \\ 0{,}1 & 0{,}9 \end{bmatrix}$.

Ahora se verá cómo converge a la política óptima el algoritmo de horizonte errante. Para ello observe la figura 3.2, en que se aprecia el acercamiento a la política óptima. En este caso, se utilizó un $\rho = 0{,}9$.

En este ejemplo sucede una particularidad que vale la pena aclarar. Puede ocurrir que para algún estado s la política del paso n coincida con la óptima:

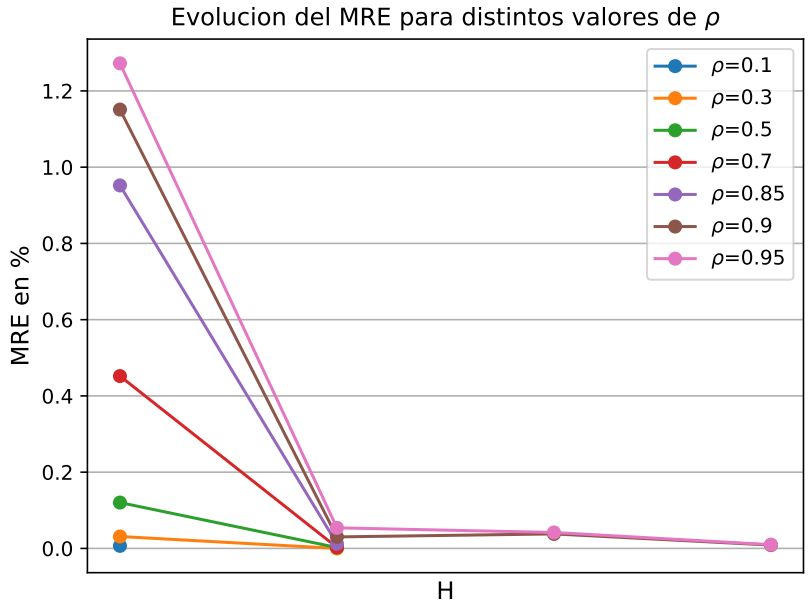


Figura 3.3: Error relativo medio entre el valor esperado al aplicar las políticas π^* y π^H para el conjunto de estados del sistema. En la figura se muestra la evolución del error para varios valores de ρ. Observar que a medida que ρ se acerca a 1, son necesarios más pasos para acercarnos a la solución óptima.

$\pi^n(s) = \pi^*(s)$, y sin embargo el paso siguiente no lo sea: $\pi^{n+1}(s) \neq \pi^*(s)$. Estas variaciones son cada vez más débiles a medida que crece n, dado que se va convergiendo a la política óptima. Para algunos estados de la figura 3.2 se puede observar que mientra que la acción óptima es "naranja" (medir ruta 0), con $H = 1$ la acción es naranja (medir ruta 0), luego en $H = 2$ es "azul" (medir ambas rutas), en $H = 3$ se mantiene azul y finalmente en $H = 4$ se estabiliza en la acción óptima.

Esta convergencia se puede apreciar en la figura 3.3, en que se grafica la evolución del error relativo medio (MRE) del retorno esperado en función del horizonte. Se utilizaron diferentes valores de ρ para apreciar el efecto que tiene en el algoritmo de horizonte errante: un futuro con más peso exige más pasos. La forma de estimar el impacto de H no es a través de la comparación de V^H con V^*, sino calculando la esperanza $\mathbb{E}_{\pi^H}[G_t]$ y comparándola con $\mathbb{E}_{\pi^*}[G_t]$. Es decir, se utiliza la evaluación de política como fue descrita en el capítulo 2 para estimar el retorno acumulado esperado de la política hallada por el algoritmo de horizonte errante.

Otra forma de ver qué tanto se acerca a la política óptima es comparando las acciones para los estados del sistema. En la figura 3.4 se aprecia la cantidad de acciones diferentes para la política π^H y π^* según diferentes valores de ρ. La política es en definitiva lo que interesa hallar: qué acción debe tomarse en cada estado. Sin embargo, esta medida no tiene en cuenta que algunos estados serán muy pocos visitados, y entonces tener acciones diferentes para estos estados tendrá poco impacto en el resultado final.

Se puede observar que no necesariamente tener más acciones diferentes conduce

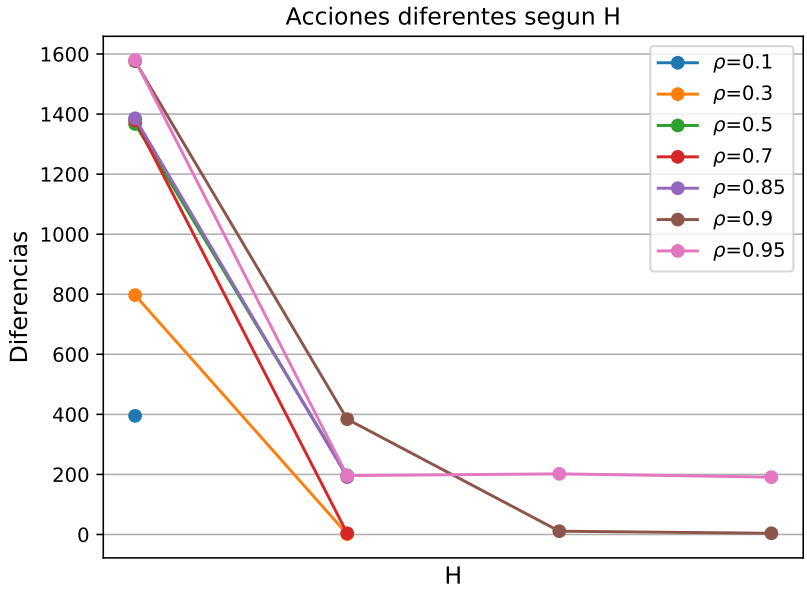


Figura 3.4: Cantidad de acciones diferentes entre las políticas π^* y π^H para el conjunto de estados del sistema. En la figura se muestra esta evolución para varios valores de ρ. Observar que a medida que ρ se acerca a 1, son necesarios más pasos para acercarnos a la solución óptima.

a un mayor nivel del MRE, por ejemplo con con $H = 2$ el caso $\rho = 0{,}95$ tiene más acciones diferentes que la política con $\rho = 0{,}9$, pero el MRE es menor para este último. Al mismo tiempo, el estancamiento que se observa en la cantidad de acciones diferentes termina yendo a 0 al crecer H. En la figura se graficaron las políticas hasta un valor del MRE de 0,002 %, que es realmente bajo. Sin embargo como se ve en la figura algunas políticas llegan a ese valor manteniendo acciones diferentes (acciones con muy poco impacto). Además, se compara contra el retorno esperado de la política óptima, pero cuyo cálculo es aproximado también. En el *policy iteration*, se elige un valor de corte hasta el cual se itera para la evaluación de la política. Exigiendo un valor del MRE tan bajo, se corre el riesgo de estar alrededor del valor de corte, y entonces se puede incurrir en pequeños errores de aproximación.

En ambos casos se observa un gran acercamiento a la política óptima con sólo incluir un paso del futuro en consideración: con horizonte $H = 2$, y aún para valores altos de ρ la política hallada tiene diferencias en menos de 400 estados de los 16000 posibles. A nivel del MRE (error relativo medio), en 2 pasos ya se encuentra debajo del 0,1 %, lo cual es muy bajo. Ya la política miope ofrecía buenos resultados, cercanía a la solución óptima y un mapa de acciones sobre los estados muy similar lo deseado; el algoritmo de horizonte errante se acerca mucho más y con sólo unos pocos pasos.

Esto es una gran noticia, dado que considerar un horizonte grande puede ser un problema. Si bien la política converge hacia la política óptima a medida que crece el valor de H, aumentar el horizonte implica recorrer más estados, es decir abrir

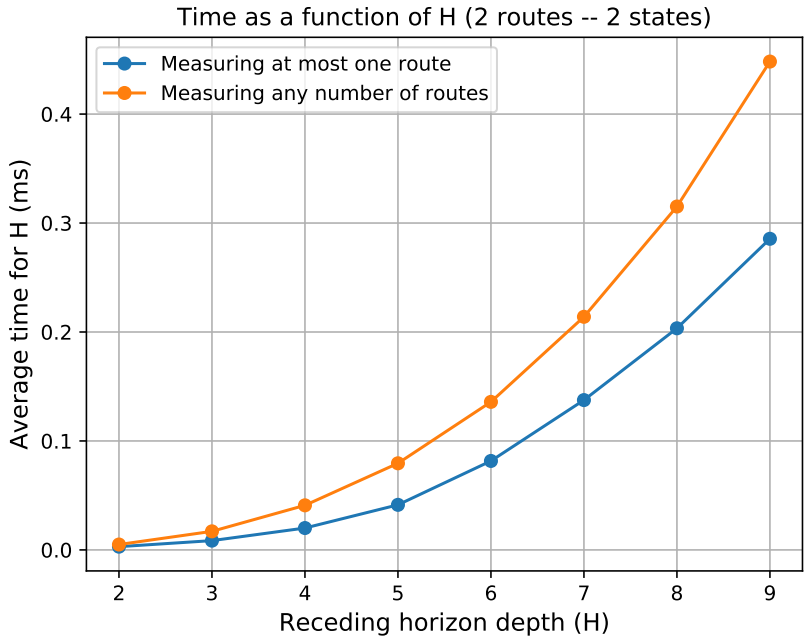


Figura 3.5: Tiempo de cálculo de la solución en función del incremento del horizonte. La curva azul se explicará más adelante, pero es resumidamente establecer la restricción de sólo medir una ruta por época como máximo, y permite reducir el tiempo de procesamiento. Observar la forma polinomial sobre el horizonte H.

más ramas del árbol, y eso puede ser costoso computacionalmente y en tiempo, como se aprecia en la figura 3.5.

3.3. Comentarios sobre la complejidad numérica

Como fue visto, la solución por técnicas tradicionales de programación dinámica implicaba, por los menos, encontrar la función de valor y acción para todos los estados del sistema. En sistemas con no más de 2 o 3 caminos, esto puede que sea aplicable, pero en escenarios más complejos y realistas esta solución deja de ser factible. Se propuso un método de aproximación que evita este recorrido obligatorio por todos los estados, sino que se limita a los estados visitados en H pasos.

Se analiza a continuación qué tanto se simplifica el problema. Suponga que el sistema se encuentra en un estado s, y el problema cuenta con N rutas todas de K niveles (por simplicidad y sin perder generalidad). En cada época de decisión hay $\mathcal{A} = 2^N$ posibles acciones, consistiendo en las posibles combinatorias de no medir ninguna ruta, medir una ruta, medir dos rutas, etc., en el conjunto de las N rutas. Considere una sola ruta $s^i = (L_{Last}, \tau)$. Esta ruta puede ser medida, pasando a valer uno de K posibles valores $\{L_i\}_{i=1...K}$, o no ser medida y pasar directamente al estado $s^{i\prime} = (L_{Last}, \tau + 1)$. Es decir que para esta ruta se tiene $K + 1$ posibles estados. Esto llevado a las N rutas significa que hay $(K + 1)^N$ posibles estados en

nuestro sistema, con cada paso que se da al ramificar el árbol.

Si en vez de dar un paso se dan H pasos, una primera mirada parece indicar que se visitan $(K+1)^{NH}$ estados. Esto haría que el problema creciera exponencialmente tanto sobre la cantidad de rutas como sobre la profundidad del algoritmo.

Sin embargo, a muchos de los estados visitados se llega desde distintas ramas. En la figura 3.1 se puede ver, por ejemplo que cada vez que se realiza la acción $[1, 1]$ (medir ambas rutas) se llega a los mismos estados. Esta repetición de estados posibles no es inocente en la figura: medir en todas las rutas siempre lleva a alguna de las combinaciones de los K^i niveles de los caminos a un valor de $\tau^i = 0$ para todas las rutas. Si hay dos rutas con dos niveles, se tiene los mismos 4 estados posibles cada vez que se evalúe la acción medir en ambas rutas.

Para ver como impacta este fenómeno en la cantidad de estados que se alcanzan H pasos después, se parte nuevamente del estado de la ruta i: $s^i = (L_{Last}, \tau)$. Ahora se precisará $L_{Last}(t) = L_i(t)$ cuando el último valor fue medido en la época t. El estado que describe a la ruta es la combinación de la última medida realizada y el tiempo que ha pasado desde esa medida. Es decir que si una ruta se midió y luego se volvió a medir, solamente esta última medida se verá reflejada en el estado de la ruta H pasos después. Interesa entonces el estado final, no así el camino realizado para llegar hasta allí. Esta ruta, H pasos después, puede haber tomado los siguientes valores:

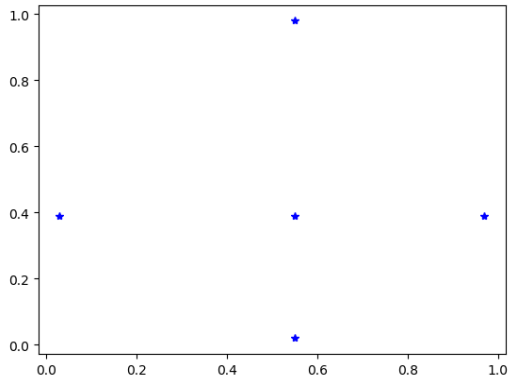
- si nunca fue medida, valdrá $s^{iH\prime} = (L_{Last}(t), \tau + H)$
- si fue medida por última vez en el primer paso $h = 1$, valdrá alguno de los K posibles estados $s^{iH\prime} = (L_{Last}(t+1), H-1)$, donde $L_{Last}(t+1)$ vale uno de los K posibles niveles de RTT de esa ruta.
- si fue medida por última vez en el segundo paso $h = 2$, valdrá alguno de los K posibles estados $s^{iH\prime} = (L_{Last}(t+2), H-2)$, donde $L_{Last}(t+2)$ vale uno de los K posibles niveles de RTT de esa ruta.
- ...
- si fue medida por última vez en el último paso $h = H-1$, valdrá alguno de los K posibles estados $s^{iH\prime} = (L_{Last}(t+H-1), 1)$, donde $L_{Last}(t+H-1)$ vale uno de los K posibles niveles de RTT de esa ruta.

Es decir que H pasos después, esa ruta puede tomar un total de $(1 + KH)$ valores posibles. Llevado a las N rutas, se obtiene un total de $(1 + KH)^N$ estados diferentes visitados. Es decir que el problema que se creía exponencial es polinómico sobre la profundidad H, aunque se mantiene exponencial sobre el número de rutas. Este fenómeno de repetición de estados visitados se aprecia en la figura 3.6, donde se muestran los estados visitados según la profundidad de H.

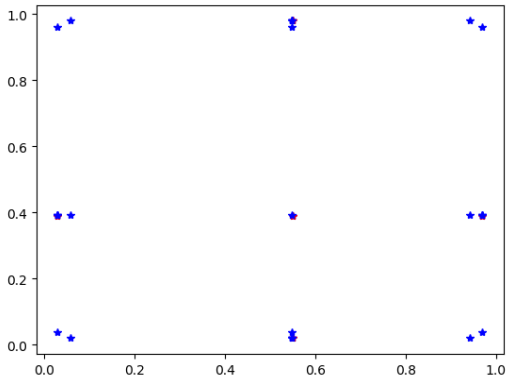
En particular, se puede apreciar en la figura 3.6 otro fenómeno interesante. Las esquinas de la figura representan los posibles valores resultantes de medir en ambos caminos (hay K^N esquinas). A partir de estas esquinas surgen hipercubos, que crecen hasta tener dimensión H^N. El número de estados posibles si se mide

Figura 3.6: Estados visitados según la profundidad H, caso de dos rutas ($N = 2$) y dos estados por ruta ($K = 2$). En particular observar que los hipercubos que se forman desde las esquinas corresponden a las trayectorias que surgen de haber medido ambas rutas en esos niveles. Los hipercubos que surgen de las aristas nacen a partir de haber medido una ruta. En un problema de dos rutas de dos niveles es fácilmente apreciable, pero la lógica subsiste al aumentar las dimensiones de N y K.

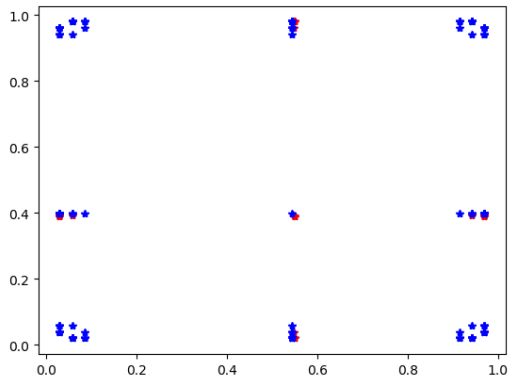
(a) $H = 1$



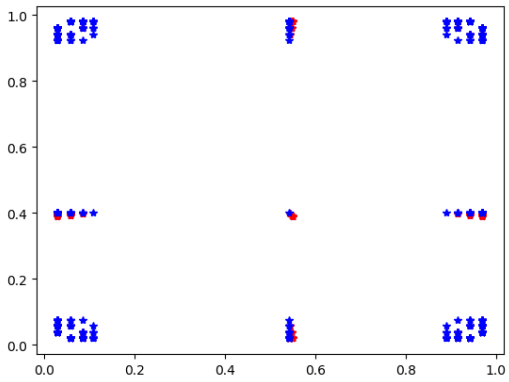
(b) $H = 2$



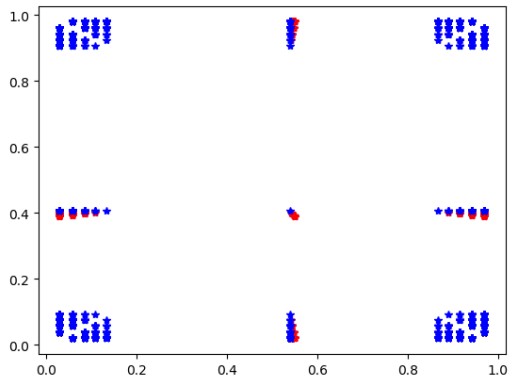
(c) $H = 3$



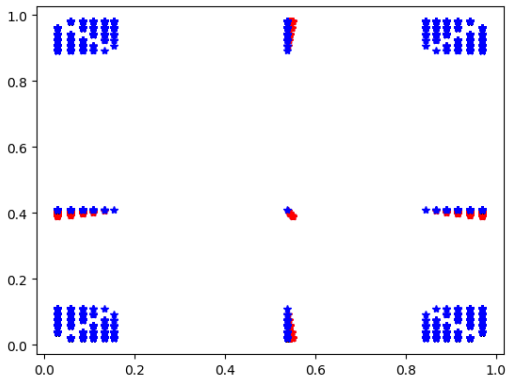
(d) $H = 4$



(e) $H = 5$



(f) $H = 6$



en una sola ruta será $\binom{N}{N-1}K^{N-1}$ y originarán hipercubos de dimensión H^{N-1} y así según las posibles combinaciones de medidas sobre los caminos. Siempre estará la serie de estados s, s', s'' resultante de no medir ninguna ruta, que recorrerá el interior del espacio de estados, nunca acercándose a un borde. Estar en un borde significa haber medido por lo menos alguno de los restantes caminos. Entonces el número total de estados en el nivel H del árbol es como ya fue planteado:

$$S^{H\prime} = \sum_{i=0}^{N} \binom{N}{i} (KH)^i = (1 + KH)^N. \tag{3.3}$$

Esta otra formulación del cálculo de los estados visitados presenta una ventaja: para algunos estados que seguro serán visitados se puede calcular de antemano la función de valor, por ejemplo en las esquinas. Si bien puede que el sistema dinámico no recorra esos estados, el algoritmo de horizonte errante seguro necesitará realizar esos cálculos, y esto permite ahorrar tiempo de cálculo, a la vez que obliga a reservar espacio de memoria.

Existe un compromiso entre guardar en memoria un diccionario posiblemente grande de valores de $Q(s, a)$ (o en su defecto $V(s)$ y $\pi(s)$), o calcular estos valores cada vez, lo que es más lento pero consume menos memoria. Con el crecimiento exponencial sobre la cantidad de caminos, se debe tener cuidado al elegir la implementación más adecuada según las características del problema. Por otro lado, la memoria ocupada se puede calcular según H, N y K, y se podría regular la cantidad de estados guardados en memoria para sólo guardar los más visitados, o sólo los visitados con cierta profundidad ($h_{\mathbf{umbral}}$).

Antes de concluir estos comentarios, es importante señalar que el algoritmo de horizonte errante cuenta con una ventaja considerable sobre los algoritmos de programación dinámica: se puede implementar en tiempo real. Recuerde que los algoritmos de programación dinámica iteran sobre todos los estados posibles, en la práctica se estiman la política y función de valor óptimas y luego se utilizan estos valores para el estado recorrido en tiempo t. Sin embargo, el *receding horizon* calcula la política sólo para el estado que toma el sistema en el tiempo t, y a través de recorrer $\sum_{h=0}^{H}(1 + Kh)^N$ estados.

Una propuesta de mejora al respecto fue establecer una restricción de que máximo se puede medir una ruta por época de decisión. Es decir: restringir el espacio de estados mediante restringir el espacio de acciones.

Se estudian los estados alcanzables H pasos después, aplicando esta restricción. Si se llama h a la cantidad de rutas medidas se obtiene la ecuación (3.4)

$$\sum_{h=0}^{H} P(H,h)\binom{N}{h}K^h = \sum_{h=0}^{H} \frac{H!}{(H-h)!}\frac{N!}{(N-h)!h!}K^h, \tag{3.4}$$

donde $P(H,h) = \frac{H!}{(H-h)!}$ representa el número de $h-$permutaciones de H.

La intuición detrás de (3.4) es la siguiente. Desde la raíz hasta el nivel H, considere h diferentes rutas que se miden, esto lleva a K^h diferentes estados en el nivel H. Además, estas h rutas pueden ser cualquiera de las N rutas disponibles, habrá por lo tanto combinaciones de h rutas a elegir en N. Además, el orden

en que se miden las rutas proporciona diferentes estados alcanzables. Se agregan entonces h-permutaciones de H épocas de medida. Finalmente, h toma todos los valores de 0 a H. Tenga en cuenta que si $H > N$, (3.4) sigue siendo válida ya que los términos desde $N + 1$ en adelante son iguales a 0.

Esto permite reducir los tiempos de procesamiento del algoritmo, pero a su vez se encuentra una solución aproximada a la solución óptima de un problema que no es el nuestro. Un posible razonamiento sería que en caso de tener que medir más de una ruta, estas medidas se encolarían en las épocas de decisión siguientes. Sin embargo, esta mejora no resultó tan buena: a pesar de obtener mejores tiempos de procesamiento, la solución se alejó de la solución óptima, llegando a tener en ocasiones resultados peores a los de la política miope. En el apéndice B se puede encontrar una explicación más detallada de la ecuación 3.4.

3.4. Implementación y límites

El algoritmo de horizonte errante fue implementado en **python** [55], así como el *modified policy iteration*.

Ambos funcionan utilizando un objeto **cadena de markov**, que se utilizará para representar el estado de cada ruta. Al mismo tiempo, utilizan un generador de ambientes **NetworkEnvironment** en el que se incorporan y combinan los objetos **cadena de markov** como un problema de ruteo. Este ambiente es el sistema: toma un estado determinado, que será la evolución del sistema dada una acción, devuelve una recompensa al tomar la acción.

Durante el desarrollo de trabajos conjuntos con grupos de Francia, el algoritmo de horizonte errante fue adaptado a un sistema de análisis de trazas de *NLNOG*[3] sobre el que se trabajó para [41] y [48].

La programación sigue los pasos planteados en el algoritmo 2. La clase **Receding Horizon** recibe los siguientes parámetros:

```
class RecedingHorizon(object):
        def __init__(self, matrixPtot, qos, costs,
                max_slots, discount_factor, depth,
                one_route=False, pruning=0, nomemory=True):
```

Donde los parámetros son:

- **matrixPtot:** las matrices de transición de Márkov de las rutas.
- **qos:** vectores con los niveles de RTT de cada ruta.
- **costs:** costo de medir cada ruta.
- **max_slots:** valor de $\tau_{\text{máx}}$ para cada ruta.
- **discount_factor:** valor de ρ.

[3]El *NLNog ring* es una red de 559 nodos repartidos en 459 sistemas autónomos distribuidos en 56 países (ver https://ring.nlnog.net).

- **depth:** valor de H.
- **one_route:** opción de restricción a medir máximo una ruta en cada época. Se desarrolla en el apéndice B.
- **pruning:** opción para realizar *pruning*, es decir limpiar el árbol de las ramas que menos aportan a la función de valor. Se desarrolla brevemente en el apéndice B.
- **nomemory:** opción de ahorro de memoria. De estar en **False** reutiliza valores ya calculados al revisitar estados.

Esta clase tiene una función a la cual se le debe indicar el estado actual, y que retorna la acción a tomar. El código utilizado para el estudio de trazas reales se encuentra disponible en https://github.com/MartinRandallC/SALMON.

El algoritmo fue utilizado en una computadora de 4 núcleos Intel i5-6200 @ 2.3 GHz. En la misma se logra resolver utilizando *policy iteration* un sistema de 3 rutas y 2 niveles con un $\tau_{\text{máx}} = 20$ y $\rho = 0{,}99$, demorando alrededor de 45 minutos. Sistemas más complejos con un valor de ρ similar no se pudieron resolver. Obsérvese que el valor de $\tau_{\text{máx}}$ planteado es muy bajo, por lo que muy probablemente no se llegue a una buena aproximación de la matriz estacionaria. En sistemas de dos rutas, este valor puede llegar a ser razonable (50, 100, 200), si el valor de ρ no es muy cercano a 1. Como ejemplo, resolver el ejemplo de dos rutas con dos niveles que se ha presentado con $\rho = 0{,}9$ demora unos 20', mientras que con $\rho = 0{,}99$ demora casi 1 hora. Se realizaron pruebas con valores de $\rho = 0{,}999$, pero fueron hechas en otras computadoras con mayor capacidad de procesamiento y memoria, y demoraron horas.

El algoritmo de horizonte errante se pudo usar en línea con 4 rutas de hasta 8 niveles, $\tau_{\text{máx}} = 100$ y un paso de $H = 3$, demorando alrededor de medio minuto por época. Se está cerca del límite para su funcionamiento en línea, dado que en las trazas utilizadas las épocas duran 2 minutos, como se explicará en el capítulo 5. Por otro lado, si se utilizan menos rutas o con menos niveles el valor de H puede crecer. En el mismo escenario, utilizando la política miope se obtiene un tiempo de procesamiento de menos de 1 segundo por época de decisión.

En este capítulo se presentó un algoritmo innovador, que combina la posiblidad de incorporar el futuro en la estimación del retorno, pero con un grado de complejidad limitado. Como se verá en la sección de Resultados5, este algoritmo de horizonte errante, fue implementado sobre trazas sintéticas y reales, logrando muy buenos resultados: alcanza con mirar uno o dos pasos hacia adelante para acercarse mucho a la política óptima. Las soluciones por programación dinámica no eran escalables más allá de problemas de 2, a lo sumo 3 rutas, mientras que la solución presentada con un horizonte de $H = 2$ fue probada con hasta 10 rutas, cumpliendo con los requerimientos temporales de durar menos tiempo que media época. Sin embargo, se sostiene en un supuesto markoviano que no siempre se cumple, por lo que se trabajará en el próximo capítulo sobre soluciones al problema de interés que no realizan esta hipótesis.

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 4

Una aproximación desde el Aprendizaje Supervisado

Los árboles siempre han sido los predicadores más persuasivos para mí. Los adoro cuando están en poblaciones y familias, en el bosque y en los bosques. Y aún más, los amo cuando están aislados. Son como hombres solitarios. No como ermitaños que huyeron por alguna debilidad, sino como grandes hombres solitarios, como Beethoven y Nietzsche.

Herman Hesse

4.1. Introducción

El incremento de la capacidad de cómputo y de la cantidad de datos disponibles ha permitido el desarrollo masivo de lo que se conoce como aprendizaje automático. Esta área del conocimiento está orientada hacia el establecimiento de procesos computacionales en que un agente aprende en base a la experiencia (en línea o pasada). Este agente puede aprender a predecir un valor, a clasificar, a distinguir o a tomar una acción.

La ingeniería de tráfico no ha sido ajena a la utilización de técnicas de aprendizaje automático. Como se ha visto en los antecedentes, muchos esfuerzos se han puesto en el aprendizaje o entrenamiento automático de modelos para las rutas, otros directamente en la toma de decisiones de medida o ruteo [29] [56] [57].

En particular los problemas de decisión markovianos están particularmente asociados a la aplicación de técnicas de aprendizaje por refuerzo, por el modelado de ambiente/entorno y el entrenamiento de los algoritmos mediante la interacción entre estos. También si es analizado como un problema de programación dinámica (con la solución obtenida a partir de la ecuación de optimalidad de Bellman), o de control discreto (utilizando aproximaciones como de horizonte errante) se puede hablar de aprendizaje automático aplicado al ruteo en sistemas de medición.

Pero los análisis basados en modelos tienen restricciones fuertes: no hay garantía alguna de que los retardos de las rutas sigan procesos markovianos, sino que se observa que muchas veces es el caso, y que considerarlos así puede reportar ventajas, dado que a partir de ahí se obtienen soluciones cerradas y óptimas, o aproximaciones de las mismas.

¿Qué pasaría si el modelado markoviano de una ruta no fuera correcto? Se obtendría un resultado no sólo probablemente subóptimo, sino además engañoso. Pensando que se está cerca de la mejor elección, se puede estar sin embargo considerando un escenario o modelo que fuera completamente diferente a la realidad.

Por este motivo, y dada la gran cantidad de datos disponibles para muchos problemas hoy en día, muchas técnicas no realizan asunciones sobre los modelos, sino que aprenden a partir del pasado diferentes características de los procesos en cuestión, ya sea de manera supervisada o no supervisada. En el caso del aprendizaje no supervisado, se aplican los algoritmos sin referencia del resultado deseado, un ejemplo son las técnicas de *clustering*.

En el caso del aprendizaje supervisado, se entrena a los algoritmos con una base de verdad (*groundtruth*). Es decir, hay un período en que se entrena al algoritmo comparando la salida obtenida con la salida real, lo que permite el aprendizaje de ciertos parámetros para determinadas entradas. Esto se realiza minimizando una función objetivo, que marca la diferencia entre la salida del algoritmo y la salida deseada. Hay en particular dos objetivos habituales: predecir o clasificar.

En el caso de predecir, se busca que la salida del algoritmo sea lo más similar posible a lo que será la salida real del proceso bajo estudio. Cuando se desea clasificar, se discrimina en clases o etiquetas según los valores o características que toma la serie analizada. Ambos casos generalmente conducen a tomar acciones en consecuencia, por ejemplo, si el tráfico es clasificado como de tal tipo, se elije esta ruta, o si se predice que el retardo en una ruta tendrá tal valor, se dirige el tráfico por esta otra ruta cuyo retardo estimado es menor.

Los más populares de estos métodos son probablemente las redes neuronales (aprendizaje profundo), pero dentro del aprendizaje supervisado hay gran variedad de algoritmos. En esta tesis se emplea un algoritmo llamado *random forest* (RF, definido en [58]). Se trata de un conjunto de árboles de decisión combinados, y es muy utilizado para analizar estructuras jerárquicas, como suele ser el caso de las redes de datos con procesos encadenados en tomas de decisiones secuenciales.

No se tratará de ubicar dentro de los algoritmos de aprendizaje supervisado los algoritmos más apropiados al problema de estudio, sino que se utiliza directamente un algoritmo conocido por su aplicación y buena performance en problemas de redes de datos para poder comparar la solución propuesta de horizonte errante con un algoritmo que no realiza asunción de modelo markoviano. Es decir que es una aproximación a la formulación del problema con algoritmos de aprendizaje supervisado, ejemplificando en uno en particular (*random forest*), lo que podría abrir la puerta a trabajos a futuro con otros algoritmos.

El principal aporte del capítulo es la propuesta de un procedimiento de procesamiento de los datos y determinación de un T óptimo de realización de medidas periódicas y simultáneas para minimizar el retorno esperado utilizando técnicas de

aprendizaje automático. Se muestra el buen desempeño de la propuesta utilizando *random forest*, pero no se restringe a su uso. La elección de *random forest* está motivada en su simplicidad, su robustez y su buen desempeño.

Un detalle no menor es que algunos algoritmos de aprendizaje supervisado podrían escalar más que los obtenidos con MDP, en que el recorrido por estados nos acerca al *curse of dimensionality*; sin embargo una limitante de estos algoritmos es que suelen necesitar una gran cantidad de muestras para su entrenamiento.

En el capítulo de introducción se presentaron varios trabajos que utilizan métodos diversos de aprendizaje automático para problemas de metrología y ruteo [29] [27]. Sin embargo, muchos autores utilizan estas técnicas como herramientas para la resolución de problemas secundarios o el ajuste de parámetros de un problema principal. Recientemente, el aprendizaje automático tomó nuevo impulso, al obtener con métodos de aprendizaje profundo resultados sorprendentes en áreas como el tratamiento de imágenes y audio, y el procesamiento de lenguaje natural.

Ya hace dos décadas [59], Breiman promovía el aprendizaje estadístico sin la utilización de modelos, como contraposición a la cultura mayoritaria en la estadística (en ese entonces). Breiman tuvo un gran impulso en el desarrollo de técnicas de aprendizaje automático sin modelado del sistema, como se verá más adelante.

En un artículo reciente se realiza un sumario estado del arte referente a dos décadas de utilización de técnicas de inteligencia artificial / aprendizaje automático para redes de datos (AI4NETS) [57]. Luego se discuten frenos existentes para su implementación en la práctica, y se proponen medidas y orientaciones a seguir para lograr un salto en calidad al vincular el trabajo académico en el área (extenso) y su aplicación (escasa).

El autor separa la utilización de algoritmos de *shallow learning* de los de *deep learning*. Los primeros son algoritmos bien conocidos y con décadas de estudio y aplicación, y si bien reportan buenos resultados exigen cierta "experticia" en el manejo de los datos y el entrenamiento, mientras que los segundos aprenden de los datos "en crudo" (para lo que necesitan muchos datos). En [56] los autores comparan diferentes métodos de aprendizaje para estimar cambios en las rutas, tiempo de vida restante de las rutas y latencia de las rutas. Destacan las virtudes del predictor de *random forest*, que es seleccionado por sus buenos resultados para el algoritmo de predicción propuesto.

Se presentará a continuación el esquema propuesto para la aplicación de técnicas de clasificación del aprendizaje supervisado para nuestro problema de interés.

4.2. Modelado del problema para aprendizaje supervisado

El problema de decisión markoviano implicaba la utilización de un modelo para los retardos de las rutas. A su vez, este modelo permite inferir información acerca del futuro, y estimar la mejor acción de medida incorporando ese futuro.

En los algoritmos de aprendizaje automático, no se realizan asunciones fuertes de modelo, y no hay acciones a tomar por parte de un agente que aporten información del sistema. Se busca directamente cuál es la ruta óptima a partir de una

serie de características que definen el estado de la red. Se cuenta una clase para cada ruta posible, y el estado de la red (las características) será una combinación del estado de las rutas y el tiempo que pasó desde que se conoció ese estado. Los algoritmos son entrenados con una serie de entrada x_i para cada característica i del sistema, y a estas combinaciones de características se les asocia y_j, correspondiente a la etiqueta (clase) j.

Para el problema estudiado, se dispone de la información de entrenamiento siguiente: se conoce el RTT de todas las rutas de largo T_{ent}. Sea x_i el retardo de la i-ésima ruta de las N rutas posibles: $\{x_1, x_2, \ldots, x_N\}$, y $x_{i,t}$ al retardo para el camino i en tiempo t. Es decir que para cada ruta x_i hay un vector de largo T_{ent}, y se tiene al conjunto:

$$
\begin{gathered}
[x_{1,0}, x_{1,1}, \cdots, x_{1,T_{ent}}] \\
[x_{2,0}, x_{2,1}, \cdots, x_{2,T_{ent}}] \\
\vdots \\
[x_{N,0}, x_{N,1}, \cdots, x_{N,T_{ent}}]
\end{gathered}
$$

En particular, se comienza por considerar que todas las rutas se miden periódicamente cada T *time-slots*, y en simultáneo, siendo T un parámetro a definir. Las características del sistema están formadas por el conjunto de las últimas medidas realizadas y el tiempo pasado desde esas medidas. La salida (la clase) es y_t un indicador de la ruta de menor RTT en tiempo t, que es conocida durante el entrenamiento, dado que se tienen las medidas completas del RTT, como se indica en el diagrama 4.1.

Se generan las muestras de entrenamiento entonces midiendo cada T, y obteniendo la siguiente serie:

$$
\begin{gathered}
(x_{1,0}, x_{2,0}, \cdots, x_{N,0}, 0, y_0) \\
(x_{1,0}, x_{2,0}, \cdots, x_{N,0}, 1, y_1) \\
(x_{1,0}, x_{2,0}, \cdots, x_{N,0}, 2, y_2) \\
\vdots \\
(x_{1,0}, x_{2,0}, \cdots, x_{N,0}, T-1, y_{T-1}) \\
(x_{1,T}, x_{2,T}, \cdots, x_{N,T}, 0, y_T) \\
(x_{1,T}, x_{2,T}, \cdots, x_{N,T}, 1, y_{T+1}) \\
\vdots \\
(x_{1,T}, x_{2,T}, \cdots, x_{N,T}, T-1, y_{2T-1}) \\
(x_{1,2T}, x_{2,2T}, \cdots, x_{N,2T}, 0, y_{2T}) \\
\vdots
\end{gathered}
$$

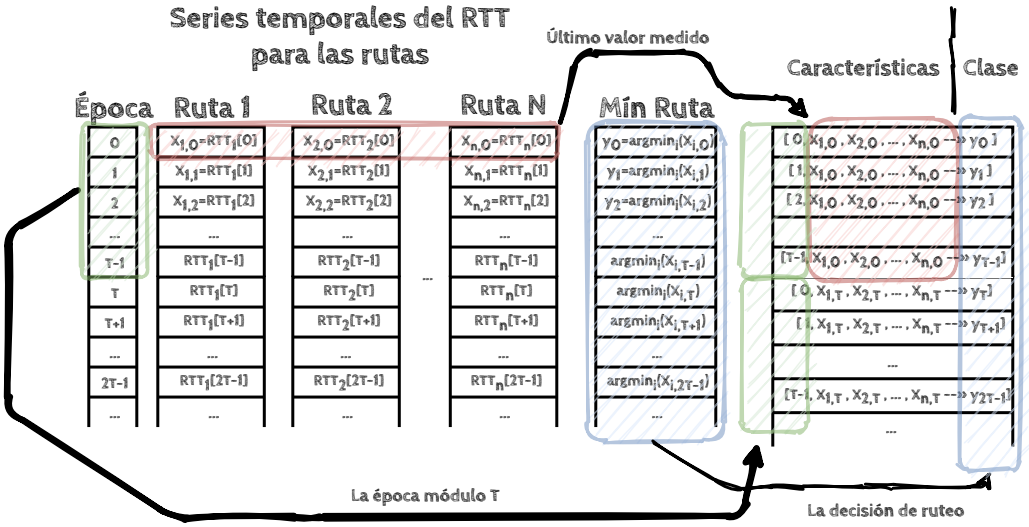


Figura 4.1: Explicación de cómo se conforman las características o *features* de entrada y su correspondiente clase o etiqueta de salida. Obsérve que se pueden desplazar los bloques una época, y se obtiene otro conjunto de muestras de entrenamiento para el problema, que es elegir la ruta más rápida habiendo medido al conjunto de rutas hace una cierta cantidad de épocas.

Esto no utiliza todo el conocimiento de que se dispone de las series temporales: se podría comenzar midiendo en la época 1, manteniendo el mismo T como período, y otra vez se dispondrá de una serie de muestras con una medida y las T mejores decisiones de ruteo en cada época siguiente. Se pasa entonces a medir cada T desde la época 1, y se obtienen otras T entradas:

$$(x_{1,1}, x_{2,1}, \cdots, x_{N,1}, 0, y_1)$$
$$(x_{1,1}, x_{2,1}, \cdots, x_{N,1}, 1, y_2)$$
$$(x_{1,1}, x_{2,1}, \cdots, x_{N,1}, 2, y_3)$$
$$\vdots$$
$$(x_{1,1}, x_{2,1}, \cdots, x_{N,1}, T-1, y_T)$$

Así se continúa, obteniendo un conjunto mayor de muestras de entrenamiento para nuestro problema. Siguiendo este procedimiento, si originalmente había L_{ent} muestras de entrenamiento, ahora se cuenta con: $T*(L_{ent}-T)$ muestras. Esto será visto con mayor detalle en la implementación, pero adelanta una dualidad entre T y L_{ent}: con valores altos de L_{ent} se pueden explorar valores altos de T, pero para valores bajos de L_{ent} sólo se podrán utilizar valores bajos de T.

Por otro lado, aún no se ha definido un valor de T. Se usará *5-fold cross-validation* para elegir este parámetro. En particular, se parte de una lista de posibles valores de T ($\{T_i\}$), sobre la que se itera buscando el T^* con el que se obtiene el retorno esperado más bajo. El retorno (o costo) se estimará utilizando los valores del retardo de las rutas elegidas y el costo de las medidas a través de:

$$\hat{V}_{rf} = \mathbb{E}[\sum_{k=0}^{K-1} \rho^k (RTT + c)] \tag{4.1}$$

Es decir que se compara utilizando la esperanza media de los retardos esperados y el costo de medir en K pasos, para lo que se utiliza validación cruzada sobre 5 divisiones.

Se introduce entonces el algoritmo **SALMON** (**S**calable **A**lgorithm for **L**atency **M**onitoring in **O**verlay **N**etworks) que se presenta en el algoritmo 3. En el mismo se propone cómo realizar el entrenamiento del clasificador y la elección del T^*. Se utiliza RF como nombre del clasificador, porque luego se utilizará *random forest*, pero podría ser cualquier clasificador.

Algorithm 3 SALMON

Dividir muestras de entrenamiento en entrenamiento y validación
for $T \in \{T_i\}$, usando *cross-validation* **do**
  Entrenar el clasificador RF.
  Evaluar el retorno esperado $\hat{V}_{rf}^{T}$ para las muestras de validación
end for
for $T^* = \text{argmín}_{T_i} \hat{V}_{rf}^{T_i}$ **do**
  Utilizar el clasificador $RF[T^*]$ en las muestras de test.
end for
Hallar el retorno esperado para las decisiones de ruteo de $RF[T^*]$.

Esta forma de presentar el problema impone dos restricciones fuertes: por un lado medir todos los caminos en simultáneo, y por otro medir los caminos periódicamente. Ambas restricciones nos alejan de posibles mejoras como ser combinaciones de $\{T_i\}_{i=1}^{N}$ medidas periódicas. Es decir que se debería, para cada combinación de períodos de medida posible, entrenar al clasificador la cantidad de veces que se realiza la validación cruzada. Si se dispone de 10 valores posibles de T_i, 4 rutas y se usa *5-fold cross-validation*, esto son $(5 * 10)^4 = 6{,}25 \cdot 10^6$ veces que se entrenaría al clasificador (con algunos miles o por lo menos cientos de muestras cada vez).

La restricción de la medida periódica es una forma de simplificar el problema a encontrar un sólo valor, que se sabe no será el óptimo para la medición, pero que es factible de hallarse. Liberar la restricción de que las medidas sean periódicas exige que se aprenda de alguna forma "cúando debo realizar una medida en cada ruta?", que es un problema muy interesante para buscar resolver si se quiere profundizar y sacar máximo provecho al aprendizaje supervisado.

Sin embargo, este acercamiento al aprendizaje automático tiene un propósito más cercano a la validación de la formulación para este tipo de herramientas, que a encontrar la solución óptima. El procedimiento mixto propuesto permite incorporar herramientas de aprendizaje automático para aprender a tomar la decisión de ruteo minimizando el costo de medida. De todas formas, se propondrán ciertas mejoras a través de liberar un poco las restricciones mencionadas.

Como primera aproximación al aprendizaje supervisado para este problema, un procedimiento sencillo puede dar buenos resultados y justificar un acercamiento

con mayor profundidad. Se presenta a continuación el algoritmo de *random forest* que fue usado para validar esta formulación del problema.

4.3. Una implementación usando *random forest*

Esta sección comienza describiendo las bases detrás de los árboles de decisión y de *random forest*. Se tomó como referencia el libro de Hastie [60] y la notación allí utilizada. Luego de esta introducción se detallan aspectos de la implementación, seguido de un ejemplo que permite ahondar en la discusión sobre la elección de parámetros. Finalmente se proponen mejoras sobre la elección de T^*.

4.3.1. Marco teórico

Los *random forest*, conocidos en castellano como bosques aleatorios, son algoritmos de aprendizaje automático, más precisamente del área del aprendizaje supervisado. Es decir que durante el entrenamiento se cuenta con la asociación entrada-salida correcta (*ground truth*). Las entradas son las *features* o características de la serie temporal, y la salida (en nuestro caso por usar un clasificador) será una etiqueta asociada a la clase a la que pertenece la entrada.

Estos algoritmos son una forma particular de *bagging*[1] para árboles de decisión, con lo que se obtiene árboles de-correlacionados, lo que permite reducir la varianza de la predicción/clasificación.

Se describen a continuación brevemente las partes estructurantes del algoritmo de *random forest*, es decir los árboles de decisión y el *bagging*.

Árboles de decisión

Los árboles de decisión son un método de clasificación o regresión conocido desde la década de 1960 (CART: *classification and regression trees* [60] [61] [62]). Consiste generalmente en sucesivas divisiones binarias (*splits*) del espacio conformado por las características (*features*). Con esto se obtienen particiones a las que corresponderán las entradas futuras.

Se presenta brevemente el esquema de utilización de CART para los problemas de regresión, y luego las modificaciones para usar árboles de decisión para clasificación. En regresión se utiliza como criterio de división (*split*) la minimización del error cuadrático medio entre la predicción y la salida.

Sean $\{x_1, x_2, \dots x_p\}$ vectores de características de las muestras de entrenamiento, con $x_i = \{x_{i1}, x_{i2}, \dots x_{iN}\}$ valores, y $y = \{y_1, y_2, \dots y_N\}$ las respuestas del sistema. El algoritmo seleccionará la partición y el valor que toma la salida en cada región. Si se cuenta con M regiones, se asigna un valor constante c_m a la predicción para cada región R_m, y se quiere minimizar el error cuadrático medio:

$$f(x_i) = \sum_{m=1}^{M} c_m I(x_i \in R_m) \quad , \quad ECM = \sum_{i=1}^{N} (f(x_i) - y_i)^2$$

[1] de **b**ootstrap **agg**regat**ing**, empaquetado en castellano.

El valor que debe tomar c_m^* es el promedio de salidas de la región R_m: $c_m^* = \mathbb{E}[x_i | x_i \in R_m]$. Para seleccionar el *split-point* o punto de separación del espacio de características s y una variable de separación j que definen dos espacios R_1 y R_2 con valor de salida c_1 y c_2, se eligen (s, j) de forma tal de minimizar el error obtenido por esa separación binaria:

$$\min_{s,j}[\min_{c_1} \sum_{x_i \in R_1}^{N} (c_1 - y_i)^2 + \min_{c_2} \sum_{x_i \in R_2}^{N} (c_2 - y_i)^2]$$

La minimización interna se resuelve con el promedio, como ya fue planteado. Seleccionado el mejor conjunto (s, j) se procede a dividir el árbol, y para cada región $\{R_1, R_2\}$ se repite el proceso, de esta forma crece el árbol.

En el caso de usarse para clasificación, se deben realizar algunos ajustes. A cada entrada corresponderá una clase, y cada hoja del árbol tendrá una clase asignada. En vez de minimizar el error cuadrático medio, se minimiza un criterio de impureza. Este criterio de impureza se utiliza como forma de minimizar el error de clasificación, que se da cuando se predice la clase equivocada.

En un nodo m, representando la región R_m con N_m observaciones, y con $\hat{p}_{mk} = \frac{1}{N_m} \sum_{x_i \in R_m} I(y_i = k)$ la proporción de observaciones de la clase k. Ese nodo asignará la clase $k(m) = \text{argmáx}_k(\hat{p}_{mk})$, que es la clase mayoritaria a la que pertenecen las entradas de la región definida en ese nodo.

En términos más geómetricos, al partir el espacio de características se busca que las regiones generadas sean lo más homogéneas posible en cuanto a que las entradas de esa región corresponden como salida a una clase. Lo que se buscará entonces minimizar es la impureza de una región. Las medidas más populares de impureza para los nodos son:

- Error de clasificación: $1 - \hat{p}_{mk}$
- Índice de Gini: $\sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$
- Entropía cruzada: $-\sum_{k=1}^{K} \hat{p}_{mk} \log(\hat{p}_{mk})$

El crecimiento del árbol debe de seguir algunos criterios, para evitar *overfitting*[2], y obtener buenos resultados de aprendizaje. Generalmente se utiliza un criterio de crecer al árbol hasta tener por lo menos N nodos, que luego es limpiado de algunas ramas por el criterio de *cost-complexity pruning*, es decir que se simplifica de las particiones que agrandan el árbol reportando el menor beneficio [60]. Este criterio considera minimizar la impureza del árbol y la cantidad de nodos terminales, y utiliza *weakest-link pruning*, que consiste en colapsar los nodos que menor impureza aportan, hasta encontrar el árbol óptimo entre tamaño e impureza.

Esta forma de hacer crecer al árbol de decisiones y luego podarlo usando *weakest-link pruning* es lo que introduce Brieman en [61], y es el método utilizado para construir las *random forest*.

[2] *overfitting:* sobreajuste, se refiere a un algoritmo que se sobre-entrena, y aprende aspectos que son particulares de una serie de entrenamiento, con lo que se obtienen malos resultados al utilizar este algoritmo ante datos desconocidos.

Random Forest

Por causa justamente de su simplicidad, los árboles de decisión son inestables, dado que una partición diferente en algún nodo del espacio de características puede generar resultados sustancialmente diferentes [63]. Hastie plantea en [60] que:

"The major reason for this instability is the hierarchical nature of the process: the effect of an error in the top split is propagated down to all of the splits below it."

Esto provoca que un árbol tenga varianza alta, para lo que se utilizan conjuntos de árboles, lo que permite disminuir este valor. El *bagging* es una forma de utilizar *bootstrap*[3] que permite reducir la varianza al construir varios predictores a partir de subconjuntos de las muestras de entenamiento [64]. Este procedimiento fue refinado por Breiman en [58] para una aplicación a árboles de decisión que consiste en la construcción de un conjunto de árboles de-correlacionados y se conoce como *random forest* (bosques aleatorios).

Los árboles se construyen a partir de considerar solamente un subconjunto aleatorio de las características en cada *split*. Luego el resultado para regresión es el promedio de la predicción de cada árbol, y si es clasificación se sigue el voto de la mayoría (*majority vote*). El algoritmo es presentado en 4.

Algorithm 4 *Random Forest*, de [60] Algorithm 15.1

1: **for** $b \in B$ **do**
2: Obtener un muestreo aleatorio Z^* de tamaño N de las muestras de entrenamiento (*bootstrap*).
3: Crecer un árbol de decisión T_b usando los siguientes pasos, hasta llegar a un tamaño mínimo de $n_{\text{mín}}$ nodos:
4: 1. Seleccionar aleatoriamente m de las p características.
5: 2. Elegir (j, s) la partición considerando las m características.
6: 3. Partir el nodo en dos nodos hijos.
7: **end for**
8: Se obtiene el conjunto $\{T_b\}_1^B$ de árboles.
9: Para regresión se usa: $\hat{f}_{rf}^B(x) = \frac{1}{B}\sum_{b=1}^{B} T_b(x)$
10: Para clasificación se usa: $\hat{C}_{rf}^B(x)$ =majority vote $\{\hat{C}_b(x)\}_{b=1}^B$, donde $\hat{C}_b(x)$ es la clase predicha por el árbol b.

Este procedimiento permite reducir sensiblemente la varianza, a la vez que mantener las ventajas de los árboles de decisión: obtienen buenos resultados siendo sencillos, robustos y escalables. Los algoritmos de *random forest* son muy populares para problemas de optimización en redes de datos [56], dado que son compatibles en problemas con estructuras jerárquicas, son interpretables, permiten manejar datos categóricos, trabajan sin métrica, entre otras características como la mencionada simplicidad.

[3] *bootstrap: es la selección aleatoria con reposición de un conjunto de las muestras disponibles, generando así diversos conjuntos de muestras que describen al proceso o estadístico bajo estudio.*

4.3.2. Implementación

A continuación se presentan los aspectos centrales de la aplicación del algoritmo SALMON al problema de ruteo con incertidumbre. A continuación se encuentra un pseudocódigo de la implementación, que sigue bastante al pie de la letra lo propuesto en el algoritmo 3.

```
Dividir series RTT en entrenamiento, validacion y test.
Usando cross-validation:
    for Ti in vector_t:
        Generar muestras de entrenamiento a partir de
        series RTT para cada ruta.
        Entrenar clasificador RF(Ti)
        Estimar retorno esperado con muestras de validacion
        incorporando costo de medida asociado a Ti.

Seleccionar T que minimiza el retorno esperado.
Aplicar clasificador de RF(T) sobre muestras de test.
```

Hay un detalle no menor en el entrenamiento del clasificador es que se deben confeccionar muestras por cada valor de T. Como se mencionó, se obtienen $T(L_{ent} - T)$ muestras para un largo L_{ent} de muestras de entrenamiento. Esto en general multiplica las muestras originales por un factor de alrededor de T, lo cual permite tener una cantidad interesante de muestras de entrenamiento para el clasificador. Pero si se quisieran utilizar valores grandes de T, por ejemplo por ser alto el costo de medida o muy estables las rutas, y L_{ent} fuera un valor pequeño, se tendría muy pocas muestras, dado que la resta $(L_{ent} - T)$ será pequeña. En esta implementación esta situación no tiene mucho impacto, como si lo tendrá en una siguiente implementación que se verá en la próxima sección.

Otra restricción importante que se observó en la implementación es que para el cálculo del retorno esperado es necesario contar con K muestras contiguas del RTT. Por otro lado, como se busca calcular una esperanza, es importante que se recorra varias veces los estados visitados, de lo contrario puede tenerse estimaciones poco realistas. Esto implica reservar una cantidad importante de muestras, tanto para la validación como para el test. Al mismo tiempo, a mayor valor de T más muestras de validación y test se necesitan: los estados se revisitan como mínimo cada T. Si $T = 100$ y hay 1000 muestras de test, la esperanza del retorno se hará promediando sólo sobre 10 valores.

Para la programación del algoritmo 3, se utilizó la librería **scikit-learn** [65] de **python 3** [55]. El número de árboles se eligió en 10 para escenarios de 2 rutas y en 100 para escenarios de más de 2 rutas, pues menos árboles podían no capturar la estructura de rutas con varios niveles, y más árboles no reportaban mejoras mientras que aumentaban el tiempo de cálculo. No se especificó un número mínimo ni máximo de hojas, y se utilizó como criterio de impureza el índice de Gini. Se realizaron pruebas utilizando como criterio de impureza la entropía cruzada, pero no se observaron diferencias sustanciales con ello. Para la construcción de cada árbol se realiza una selección aleatoria de las muestras de entrenamiento (*bootstrap*

samples).

Los códigos utilizados para el análisis de trazas reales de *NLNOG* se encuentran disponibles en https://github.com/MartinRandallC/SALMON.

4.3.3. Un ejemplo sencillo para la selección de T^*

Se vuelve al ejemplo predilecto del lector, de dos rutas y dos niveles, presentado en 2.1.2. Refrescamos el ejemplo: los valores para la ruta 1 son $l_0 = 1$, $l_1 = 3$, $c_1 = 0{,}15$, $P_1 = \begin{bmatrix} 0{,}9 & 0{,}1 \\ 0{,}1 & 0{,}9 \end{bmatrix}$; y para la ruta 2 valen $l_0 = 0{,}5$, $l_1 = 2$, $c_0 = 0{,}05$, $P_0 = \begin{bmatrix} 0{,}7 & 0{,}3 \\ 0{,}3 & 0{,}7 \end{bmatrix}$.

Para este ejemplo se generan trazas markovianas, por lo que el estudio del sistema como un MDP es directo, pero permitirá presentar la implementación del algoritmo. En este caso se generan series de 10000 muestras. Las primeras 5000 serán usadas como entrenamiento, usando *5-fold cross-validation*, y las últimas 5000 de test. El algoritmo ensaya los siguientes valores posibles de T:

```
vector_t=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 15,
    20, 25, 30, 40, 50, 60, 70, 80, 90, 100]
```

Se obtiene en la figura 4.2 la evolución del retorno esperado en función de T. Al inicio, con $T = 1$, el costo de medida se vuelve excesivo, a pesar de que la información que se obtiene del sistema permite tomar siempre la mejor ruta. Por eso el T^* óptimo se encuentra entre valores bajos de T, pero no tan bajos como para medir constantemente: $T^* = 2$.

Obsérvese que el costo de medida por época será alrededor de $\frac{\sum_i c_i}{T}$, por lo que disminuir T en los primeros pasos reduce drásticamente el costo de medida, pero permite tener suficiente conocimiento para clasificar correctamente. El costo de medida puede verse como un *offset* proporcional al costo y al período, que se suma a la decisión de ruteo. Es decir que a T fijo, aumentar el costo es simplemente sumar una constante al retorno esperado; mientras que a costo fijo, aumentar T disminuirá el retorno esperado. Esto es una desventaja en tanto el único ajuste de la política en función del costo de medida es sobre T.

Además del retorno esperado es interesante observar el desempeño del clasificador, para lo que se utilizará la tasa de aciertos o *accuracy*. En la figura 4.3 se puede apreciar la evolución de la *accuracy* según T. El resultado es el esperado: midiendo más seguido se dispone de mayor información, y al aumentar T llegará un momento en que prácticamente no se gana ni pierde información (dado que se sabe muy poco). La *accuracy* se calcula como la proporción de aciertos en la predicción de la clase para un subconjunto de muestras de entrenamiento utilizadas como test; para el problema en consideración es la cantidad de veces que elegimos la ruta rápida.

En esta ocasión se observa la convergencia de la función de valor esperada luego de superar un valor de $T > 25$. Esto se da pues las medidas tan espaciadas no aportan información significativa del sistema, por lo que el algoritmo simplemente selecciona una ruta para todo tiempo, que será la de menor valor medio de RTT.

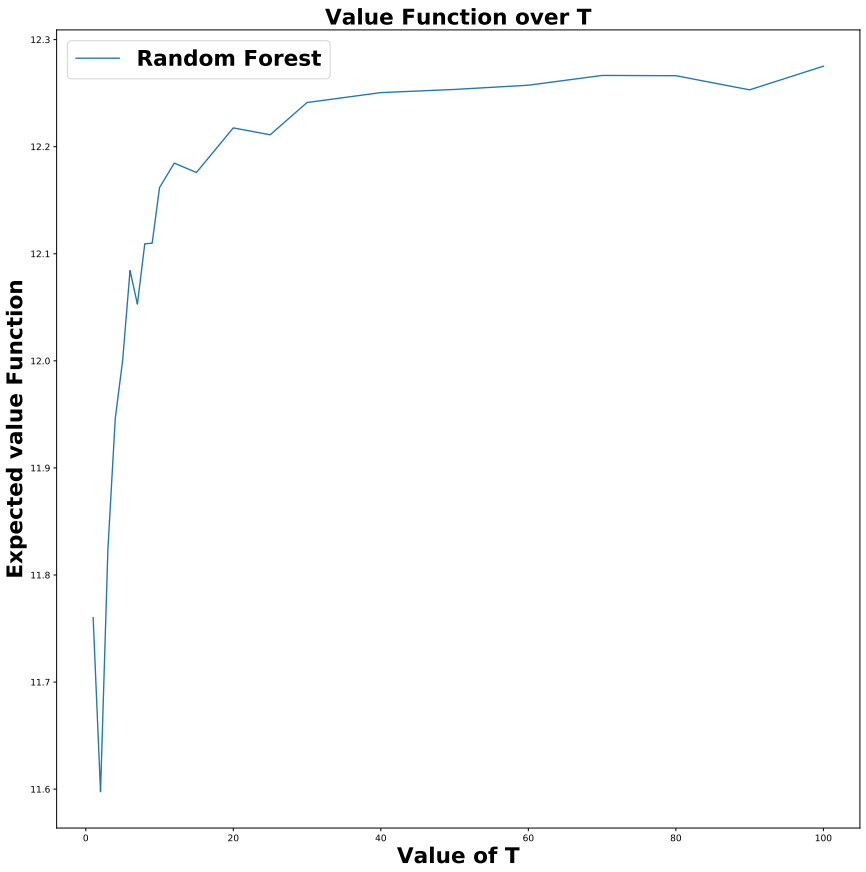


Figura 4.2: Evolución del retorno esperado según el valor del paso de medida T. Observar que a medida que T crece, el retorno esperado aumenta, excepto con paso muy pequeño ($T = 1$) en que el costo de medida es muy alto.

Por otro lado, realizar medidas con valores altos de T implica la necesidad de contar con series temporales muy largas.

Nótese la simetría entre el *accuracy* y el retorno esperado: se estabilizan luego de un cierto T que ronda 25. El peso del costo de medida disminuye a medida que T aumenta, y en particular podría verse cuándo pasa a ser despreciable. En cada época se suma el retorno esperado de las rutas, que en el peor escenario será haber elegido la peor ruta (ruta 0 en nivel alto, RTT=3) y en el mejor escenario es elegir la mejor ruta (ruta 1 en nivel bajo, RTT=0.5). El costo medio en una época vale $c[t] = \frac{\sum_i c_i}{T} = \frac{0{,}2}{T}$. Se considera despreciable como 10 veces más pequeño, entonces con $c[t] \approx 0{,}05$ se tiene impacto despreciable del costo para esa época (en el escenario en que más impactaría, es decir con el menor valor de RTT posible). Esto hace que para todo $T \geq 40$ el costo sea despreciable. Esta es una cota muy

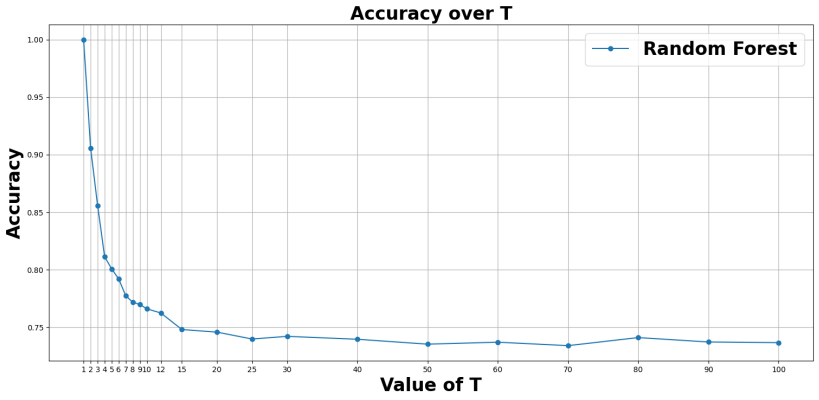


Figura 4.3: Evolución de la *accuracy* en la clasificación según el valor de T.

burda, pero permite mostrar el efecto del costo, que es muy importante para los valores bajos de T pero no pesa para valores altos.

En particular, el valor óptimo de T^* es un equilibrio entre el costo de medida, y la frecuencia de medida que demanda cada ruta. Las rutas tensionan al valor de T: si se está ante una ruta de pocos cambios y otra muy variable, se medirá innecesariamente la ruta más estable y no se medirá lo suficiente la más incierta. En la próxima sección se presenta una mejora a propósito de este problema, que busca relajar la restricción sobre medir todas las rutas en simultáneo.

A pesar de estas limitaciones, el algoritmo logró buenos resultados, comparables a los obtenidos con el algoritmo de horizonte errante, como se verá en el capítulo 5 de resultados.

4.3.4. Buscando mejores tiempos de medida

Con el fin de encontrar un mejor subconjunto de valores para cuándo medir, y principalmente, para disociar la medición en todas las rutas al mismo tiempo, se propone una modificación. Se supone en esta sección que sólo se trabaja en un escenario de 2 caminos.

Hasta ahora, se selecciona el mejor (T, T) para una medición periódica de ambas rutas. Una propuesta natural sería hacer una selección más fina de (T_1, T_2), donde T_i es el valor T para el cual se mide periódicamente la ruta i. Como fue planteado, esta búsqueda crece rápidamente a medida que crece el número de caminos, y se vuelve a caer en una especie de *curse of dimensionality*.

Un enfoque más acotado es buscar un mejor (T_1, T_2) únicamente en un intervalo alrededor del (T, T) seleccionado, tal que $T_1, T_2 \in T^* \pm I$, con $I = 3$ por ejemplo. Esto puede permitir que una de las rutas se mida con más frecuencia que la otra, pero no por una gran diferencia, y sin recorrer todas las combinaciones posibles de (T_1, T_2).

Se implementó esta relajación, que se presenta en el algoritmo 5, obteniendo resultados muy similares al algoritmo original. En muchos casos, se prefiere un par

diferente de valores de T, pero los resultados terminan siendo prácticamente los mismos. El mejor escenario para esta aplicación sería una ruta muy estable y otra más variable, para tener un valor T más bajo para la ruta cambiante y un valor más grande para la estable. Aún así, la primera selección de una pareja (T, T) ya puede ser restrictiva de las posibilidades de tal mejora. Esto es porque después de seleccionar el valor T, ya estuvo el juego de tensiones entre las frecuencias deseadas por cada ruta, y un intervalo alrededor de ese valor T seguramente no sea óptimo para ninguna de las rutas.

Algorithm 5 Relaxed SALMON

Dividir muestras de entrenamiento en 2 series de entrenamiento y validación.
for $T \in \{T_i\}$, usando *cross-validation* **do**
  Entrenar el *random forest classifier*
  Evaluar el retorno esperado $\hat{V}_{rf}^{T}$ para la primera serie de muestras de validación
end for
$T^* = \text{argmín}_{T_i} \hat{V}_{rf}^{T_i}$
for (T_1, T_2), $T_1, T_2 \in T^* \pm I$, usando *cross-validation* **do**
  Entrenar el *random forest classifier* con las segundas muestras de entrenamiento.
  Evaluar el retorno esperado $\hat{V}_{rf}^{(T_1,T_2)}$ para la segunda serie de muestras de validación.
end for
for $T^* = \text{argmín}_{(T_1,T_2)} \hat{V}_{rf}^{(T_1,T_2)}$ **do**
  Utilizar el *random forest classifier* $RF[T^*]$ en las muestras de test.
end for
Hallar el retorno esperado para las decisiones de ruteo de $RF[T^*]$.

Se aplica esta versión del algoritmo a nuestro ejemplo de dos rutas y dos niveles. En la figura 4.4 se aprecia el retorno esperado para las combinaciones de (T_1, T_2) alrededor de (T, T). En esta ocasión el algoritmo prefirió la combinación $(3, 1)$, lo cual es razonable: se medirá más la ruta 2 que es más variable, y menos la ruta 1 que es más estable. Además, la ruta que más se mide es la de menor costo, y conociendo con certeza si está en nivel alto o bajo (medir con paso 1 es medir siempre) el clasificador cuenta con suficiente información como para evitar medir siempre en la otra ruta y así poder disminuir el costo de medida.

El costo de medida pasa de valer $c = \frac{L_{test}*(0{,}15+0{,}05)}{2}$ a $c = \frac{L_{test}*0{,}15}{3} + \frac{L_{test}*0{,}05}{1}$. En este caso la ganancia no existe a nivel de costo de medida respecto a $T = (2, 2)$, dado que si bien en una ruta se mantiene la frecuencia de medidas, la medida permanente en la ruta variable aumenta el costo. Es entonces que el motivo por el que el retorno esperado sea menor para esta combinación está en la información que aporta realizar medidas permanentemente en la ruta rápida. En el capítulo de resultados se empleará esta relajación sobre una traza real para la que se obtiene un mejor desempeño del algoritmo modificado que del original.

Otra idea que se podría explorar, es encontrar de alguna manera una relación

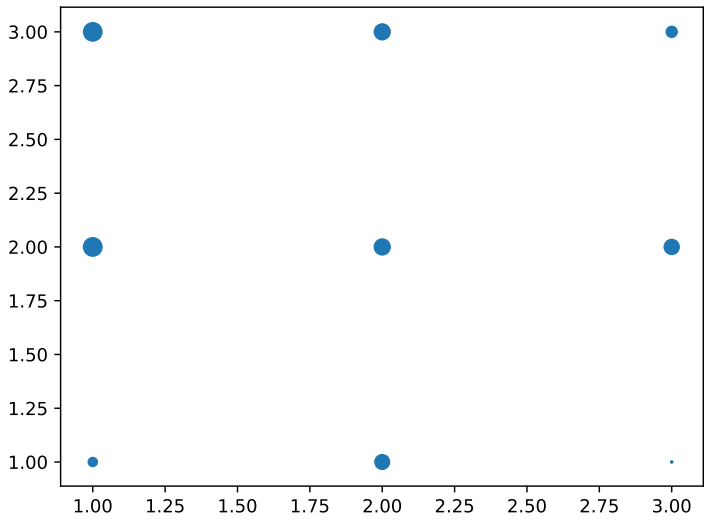


Figura 4.4: Retorno esperado en un intervalo alrededor de (T, T). El valor más bajo se logra en $T = (3, 1)$.

entre T_i y algunas estadísticas de nuestros caminos. El objetivo sería utilizar a nuestro favor el conocimiento de las variaciones de la ruta, buscando un T más alto para rutas más estables, y un T más pequeño para las más variables. Esta línea de trabajo no se ha profundizado, ya que existe el temor de que pueda conducir a un análisis de la recompensa esperada en función de T, que se acerque a un marco de análisis markoviano, a través del establecimiento de modelos y probabilidades de transición. Se vuelve sobre esta idea en los capítulos 5 y 6 de resultados y conclusiones.

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 5

Pruebas y Resultados

> De hecho, cada día podemos evidenciar que la especie humana no ha desarrollado todavía un sentido de la probabilidad muy agudo. Quizá, a corto plazo, deberíamos encarar el azar con cierta prudencia.
>
> Deborah J. Bennett

5.1. Introducción

En los capítulos anteriores se presentaron tres algoritmos para elegir las acciones de medición y ruteo en el problema de ruteo inteligente para redes sobrepuestas.

En primer lugar se consideró que los retardos en las rutas son modelables como cadenas de markov, y que se puede ver el problema como un problema de decisión markoviano, a raíz de lo cual se introdujo un algoritmo que ofrece la solución óptima, calculable para escenarios de pocas rutas y pocos niveles. Luego se propuso un algoritmo de horizonte errante que aproxima esta solución de manera más escalable.

Finalmente, dejando de lado el modelado markoviano del sistema, se implementó un algoritmo basado en la técnica de aprendizaje supervisado que es el clasificador de *random forest*. De esta forma se clasifica el estado de la red y se elige la ruta en consecuencia, realizando medidas periódicas cada T épocas cuyo valor se busca optimizar.

En este capítulo se presentan tres tipos de pruebas y resultados de validación. Primero se describen en la Sección 5.2 las pruebas sintéticas, es decir aquellas en que se generan series temporales que representan los retardos de cada ruta. El objetivo es evaluar y comparar la precisión de las soluciones de monitoreo propuestas. Estos escenarios sintéticos tienen varias ventajas para el ajuste de los algoritmos y probar su funcionamiento, y a su vez permiten el desarrollo de escenarios pequeños en que se puede obtener la solución exacta. Por otro lado, la generación de series temporales se realiza a partir de asunciones: qué distribución se usará para simular cada ruta responde al modelado del retardo que se elije. Es

decir que no permite contrastar con la realidad, de la que un modelo puede no capturar eventos o fenómenos importantes, y además da ventaja a los algoritmos markovianos, por encontrarse alineados con el modelo. Se realizaron pruebas con rutas de retardos no markovianos para considerar este aspecto.

En la Sección 5.3 se utilizan los algoritmos propuestos sobre trazas reales, obtenidas a través de *NLNOG-Ring*. Esto permite una validación mayor, dado que se puede trabajar con datos reales, y permite también testear los algoritmos en línea.

En ambos casos se realiza una comparación de los resultados obtenidos con una heurística presentada en [48] que busca resolver el mismo problema. Luego de presentadas las pruebas y los principales resultados, se realizará una evaluación de los algoritmos. A partir de este análisis y para terminar, se presentarán evaluaciones y propuestas de mejoras a los algoritmos empleados.

5.2. Trazas sintéticas

Se comenzará por estudiar el comportamiento de los algoritmos en escenarios controlables, en que se puede calcular la solución utilizando el *modified policy iteration*. Se ha utilizado a lo largo del texto el ejemplo de dos rutas y dos niveles presentado en 2.1.2. Una serie de ejemplos de características similares servirá para validar el funcionamiento de los algoritmos.

En particular se centrará nuestra atención en dos aspectos:

- ¿Se acercan a la solución óptima del MDP?
- ¿Qué tan robustos son ante escenarios no markovianos?

Se presentarán tres casos sintéticos de trazas markovianas: dos rutas muy estables, una ruta estable y una variable, y dos rutas muy variables. Sobre estas rutas se estudiará el desempeño de los algoritmos, la selección de T para el SALMON y otros parámetros de interés.

Para comparar los resultados obtenidos, se deben establecer métricas apropiadas. No es trivial, dado que son soluciones pensadas para problemas diferentes: ya no es posible recorrer los estados del sistema comparando el retorno esperado como se realizó en el capítulo 3.

Por un lado se puede estimar el retorno esperado $\hat{G}$ para la política π, que se definió igual para ambos problemas:

$$\hat{G}_\pi = \mathbb{E}[\Sigma_{k=0}^{K-1}([c[k] + RTT[k]] * \rho^k)]$$

Se utilizará un valor de K a partir del cual son despreciables los términos siguientes. Esto depende directamente de ρ, por ejemplo para un $\rho = 0{,}9$, que es un valor bastante usado en los ejemplos, se usará $K = 30$, para el cual $\rho^K \approx 0{,}04$ (tendrá un impacto menor al 5 %). Esto implica generar una traza y evaluar el retorno esperado, promediando el valor de $\Sigma_{k=0}^{K-1}([c[k] + RTT[k]] * \rho^k)$ para cada estado por el que se pasa. Si el sistema pasa por un estado s, se calcula el retorno de

	RTT		Costo
	Ruta 1 [bajo, alto]	Ruta 2 [bajo, alto]	[Ruta 1, Ruta 2]
Márkov 1	[330, 420]	[350, 370]	[20, 20]
Márkov 2	[1, 3]	[0.5, 2]	[0.15, 0.05]
Márkov 3	[350, 370]	[320, 400]	[10, 10]
Rayleigh	[320, 400]	[340, 375]	[40, 40]
Pareto	[1, 3]	[0.5, 2]	[0.15, 0.05]
Márkov 2 (b)	[1, 3]	[0.5, 2]	[0.075, 0.025]
Márkov 2 (c)	[1, 3]	[0.5, 2]	[0.3, 0.1]

Tabla 5.1: Descripción de los escenarios sintéticos utilizados para la validación de las heurísticas.

pasar por ese estado incorporando las $K-1$ muestras siguientes. Luego, al volver a pasar por ese estado, se vuelve a calcular este retorno. Tras muchas pasadas por el estado, se promedia para tener la esperanza del retorno de pasar por ese estado.

Por otro lado, es posible comparar por separado las decisiones de medida y ruteo: se desea elegir la mejor ruta, y al menor costo de medida. Por cierto que la estimación del retorno integra estos dos aspectos, pero el horizonte errante busca efectivamente minimizar este valor, mientras que el SALMON busca el T que minimiza esta esperanza pero utiliza un clasificador únicamente enfocado a la decisión de ruteo. Entonces se comparará el costo de medida promedio por época y por otro lado qué porcentaje del tiempo se realizó la elección óptima de ruta. Los escenarios sintéticos utilizados se encuentran descritos en la tabla 5.1.

En esta configuración, se puede encontrar la solución óptima para el problema de medición / enrutamiento, mediante el uso de programación dinámica. Para ese propósito, se utiliza el algoritmo de *policy iteration* descrito en el capítulo 2. Se presentan los resultados de nuestro algoritmo de horizonte errante (con un horizonte de 3), y el resultado del algoritmo SALMON. Se utiliza un bosque de 10 árboles y la selección de T se realiza mediante la validación cruzada de 5 veces del retorno esperado. Se utiliza también para estos escenarios la modificación que permite buscar el valor de $T = (T_1, T_2)$ en un intervalo de (T, T). Los resultados se presentan en las tablas 5.2-5.3.

Primero se usan los algoritmos para estimar la mejor ruta en un escenario de 2 rutas donde cada ruta tiene 2 retardos posibles, generados como un proceso markoviano. Las matrices de transición para cada ruta son:

$$P_1 = \begin{pmatrix} 0{,}95 & 0{,}05 \\ 0{,}05 & 0{,}95 \end{pmatrix}, \; P_2 = \begin{pmatrix} 0{,}95 & 0{,}05 \\ 0{,}05 & 0{,}95 \end{pmatrix}.$$

En este caso las matrices de transición tienen mucho peso en la diagonal: es decir que estando en un nivel la probabilidad de mantenerse en este nivel es alta. Por otro lado, la simetría hace que un nivel no monopolice a esa ruta, puesto que la matriz de transición estacionaria tendrá pesos iguales para cualquier movimiento

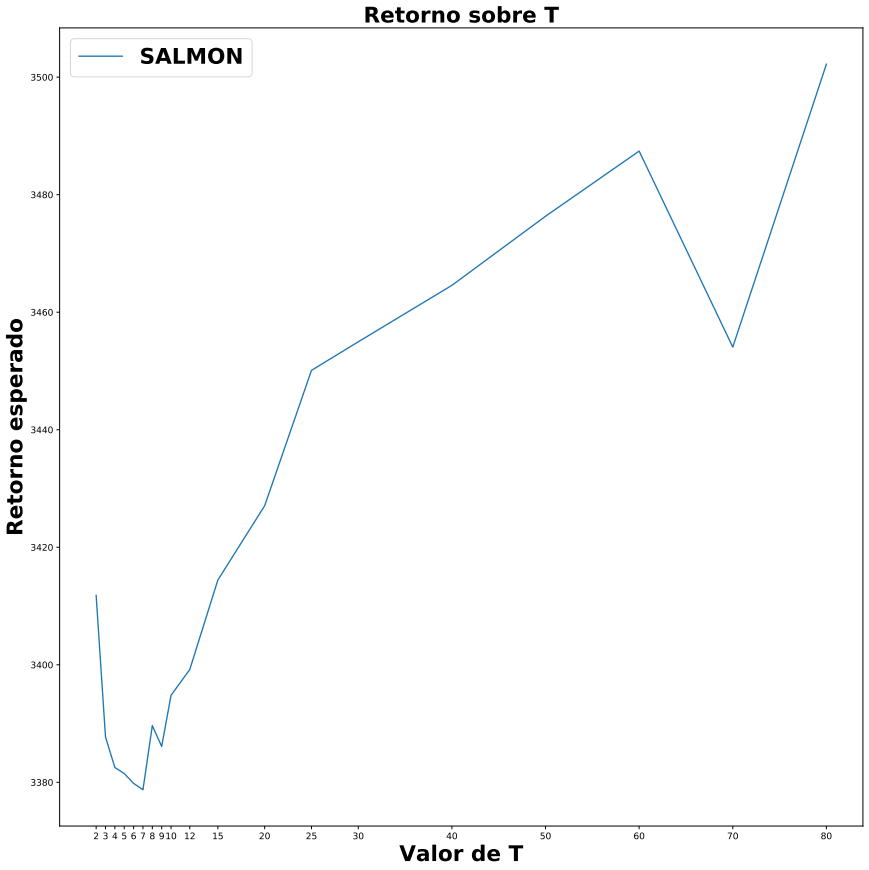


Figura 5.1: Escenario sintético número 1, con trazas generadas como retardos markovianos. Valor esperado para diferentes valores T posibles. Se alcanza el mínimo para $T = 7$. Luego de la búsqueda alrededor de este intervalo se confirma el T elegido, $(T_1, T_2) = (7, 7)$

entre estados. Los resultados de la evoluión del retorno esperado en función de T se aprecian la figura 5.1. Por otro lado, se grafica la *accuracy* en la figura 5.2.

Luego se prueba una configuración en que hay una ruta más variable y otra más estable. Las matrices de transición son para cada camino:

$$P_1 = \begin{pmatrix} 0{,}9 & 0{,}1 \\ 0{,}1 & 0{,}9 \end{pmatrix}, \; P_2 = \begin{pmatrix} 0{,}7 & 0{,}3 \\ 0{,}3 & 0{,}7 \end{pmatrix}.$$

La configuración de este MDP se corresponde con el ejemplo utilizado a lo largo del texto. Los resultados se muestran en las tablas 5.2-5.3 y en las figuras 5.3 y 5.4. En este caso se observa un valor bajo del T, aunque hay una clara búsqueda de un valor más alto para la ruta más estable y más bajo para la más variable, como fue analizado en el capítulo 4. Siendo que el valor mínimo permitido es de

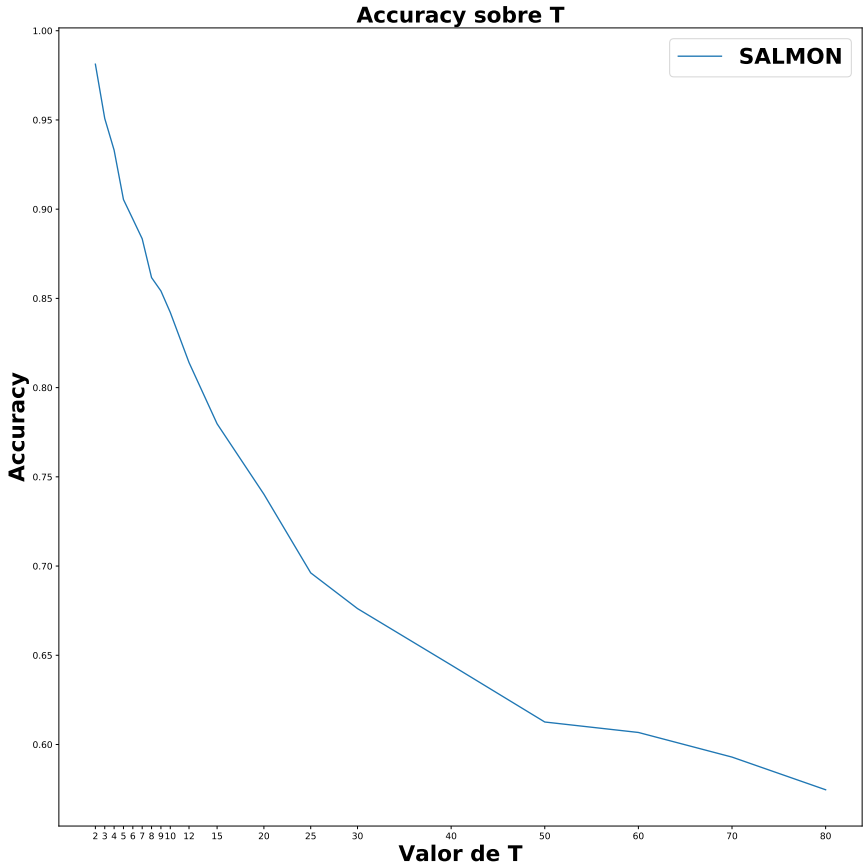


Figura 5.2: Escenario sintético número 1, con trazas generadas como retardos markovianos. *Accuracy* en función de T.

$T = 2$, dado que así se evita medir en todas las épocas, el algoritmo primero selecciona este valor mínimo. Luego, busca en el intervalo alrededor del mismo, y con medir siempre la ruta más variable y cada 3 épocas la menos variable logra la mejor combinación. Ya el hecho de medir siempre una ruta da mucha información, y entonces se aliviana el retorno esperado disminuyendo el costo de medida y casi manteniendo el conocimiento del estado del sistema.

Para este caso, que es el ejemplo que se ha utilizado a lo largo de la Tesis, multiplicamos y dividimos los costos de medida por dos. De esta forma podemos ver el impacto que tiene en ambos algoritmos, restringiendo las medidas. Los resultados se agregan en las últimas filas de las tablas 5.2-5.3, como escenarios Márkov 2 (b) y 2 (c). En el escenario Márkov 2 (b), el costo vale la mitad y se selecciona el T más bajo posible: $T = (1, 1)$. En el escenario Márkov 2 (c) el costo de medida vale el doble, y la selección de T es coherente: $T = (14, 14)$. Los algoritmos markovianos

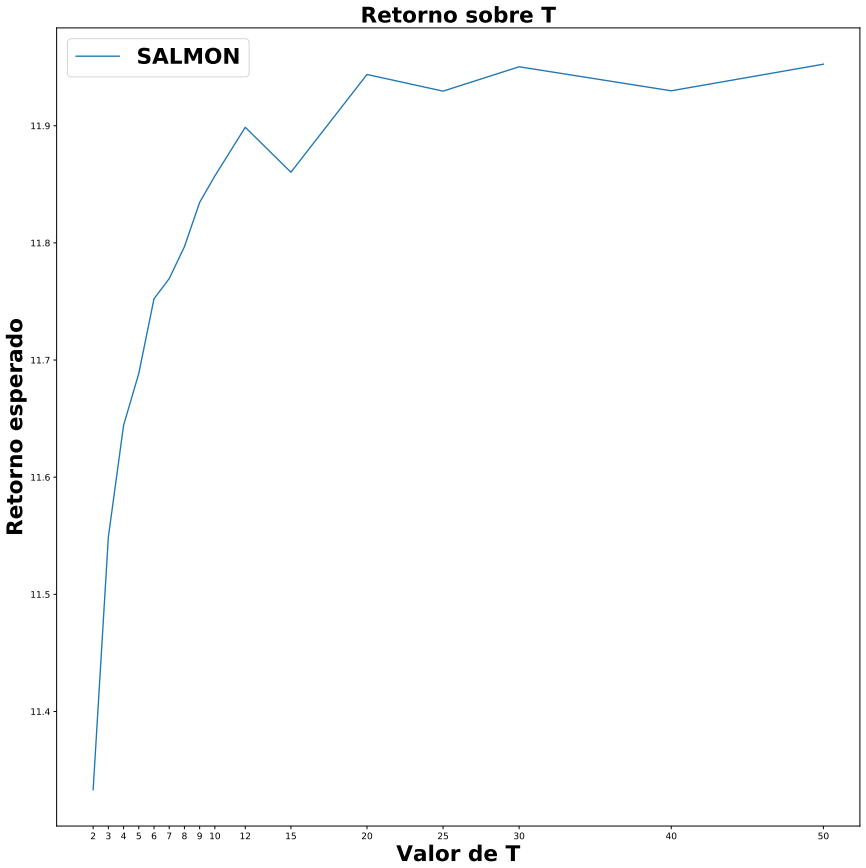


Figura 5.3: Escenario sintético número 2. Valor esperado para diferentes valores de T posibles. El mínimo se alcanza en $T = 2$.

pasan de realizar medidas con costo de 278 para el caso $c = [0{,}075, 0{,}025]$, a ≈ 400 para $c = [0{,}15, 0{,}05]$ y a ≈ 500 para $c = [0{,}3, 0{,}1]$.

Se continúa con la evaluación de los algoritmos para el último escenario sintético markoviano: dos rutas altamente variables. En este caso se realizarán más medidas en ambos algoritmos. Aún así no se logran alcanzar muy buenos resultados en cuanto a la elección de la ruta más rápida. Se emplean los siguientes valores para las matrices de transición:

$$P_1 = \begin{pmatrix} 0{,}8 & 0{,}2 \\ 0{,}3 & 0{,}7 \end{pmatrix}, \; P_2 = \begin{pmatrix} 0{,}7 & 0{,}3 \\ 0{,}3 & 0{,}7 \end{pmatrix}.$$

En la figura 5.5 se puede ver la evolución del retorno esperado en función de T, y en la grafica 5.6 la evolución de la *accuracy*. La elección de un T bajo confirma lo intuitivo y ya discutido: rutas variables exigen más medidas. En este caso, el

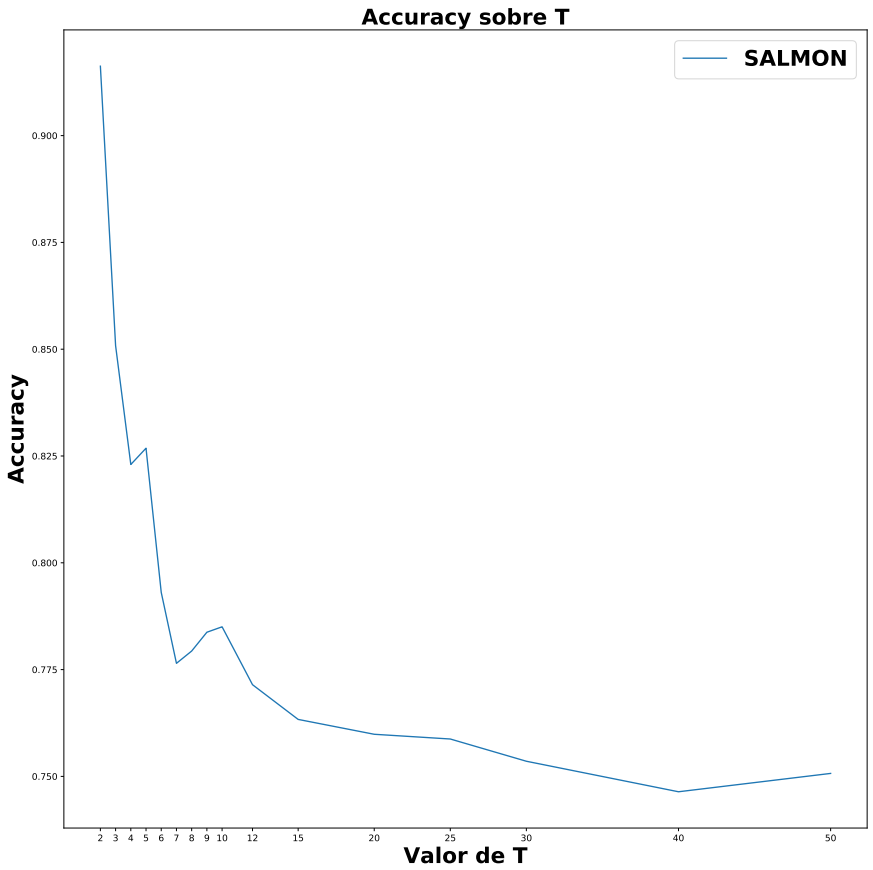


Figura 5.4: Escenario sintético número 2. Evolución de la *accuracy* según el valor de T.

costo relativamente alto impide que el SALMON elija un valor muy bajo de T, sin embargo el algoritmo de horizonte errante y el *policy iteration* realizan medidas en todas las épocas, lo que explica el resultado en cuanto a siempre elegir la ruta más rápida. Se probo bajando el costo de medida, con lo que se obtenía un rendimiento más pobre del SALMON con un 55 % del tiempo eligiendo la ruta más rápida, e igualmente aceptable de los algoritmos de horizonte errante y *policy*, con un 80 % del tiempo eligiendo la ruta más rápida.

Para todas las configuraciones, el bosque aleatorio se desempeña realmente cerca de los algoritmos de programación dinámica, y el de horizonte errante llega a la solución casi-óptima. Es interesante observar que la selección T tiene un comportamiento similar: la T óptima es tal que el costo no es demasiado alto (con una T baja, el costo puede tener un gran impacto en la recompensa esperada), y la medición de la ruta es suficiente para tomar decisiones de enrutamiento óptimas.

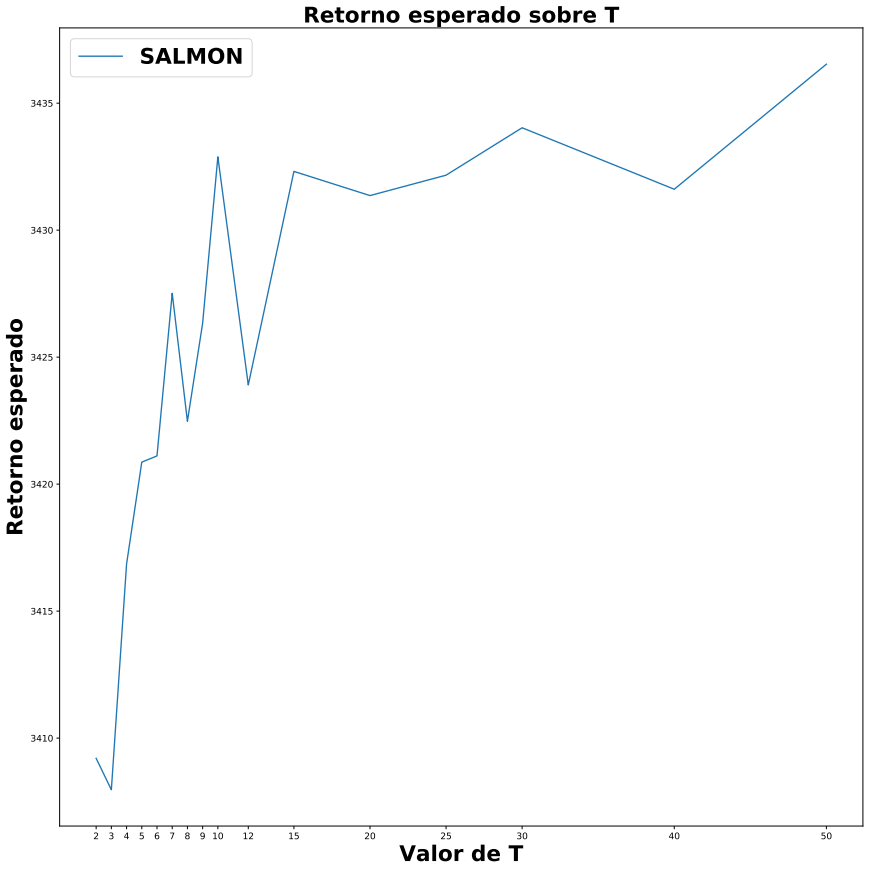


Figura 5.5: Escenario sintético número 3. Valor esperado del retorno para diferentes valores de T posibles. El mínimo se alcanza en $T = 4$. Luego se elegirá $(4, 4)$ como valor que minimiza el retorno esperado en un intervalo de T.

Con una configuración de bajo costo, una T más baja siempre será una opción preferida. Al mismo tiempo, una ruta muy cambiante exigirá un valor menor de T. Por lo general, hay un valor de T a partir del cual el retorno esperado es estable, ya que el costo de medición tiene poco impacto en la estimación del retorno, y el conocimiento sobre las rutas no es suficiente para tomar buenas decisiones de enrutamiento (el algoritmo preferirá elegir una misma ruta siempre). Esto es particularmente claro en la figura 5.3. En el escenario de rutas más variables, se utilizó un costo bajo inicialmente, lo que generó medidas permanentes para el *policy iteration* y el horizonte errante, y un valor bajo de T para el SALMON, y los tres algoritmos eligiendo la mejor ruta en todo momento. Al subir el costo y penalizar la medida se logra disminuir la cantidad de medidas realizadas, y se obtienen los resultados de las tablas 5.2-5.3.

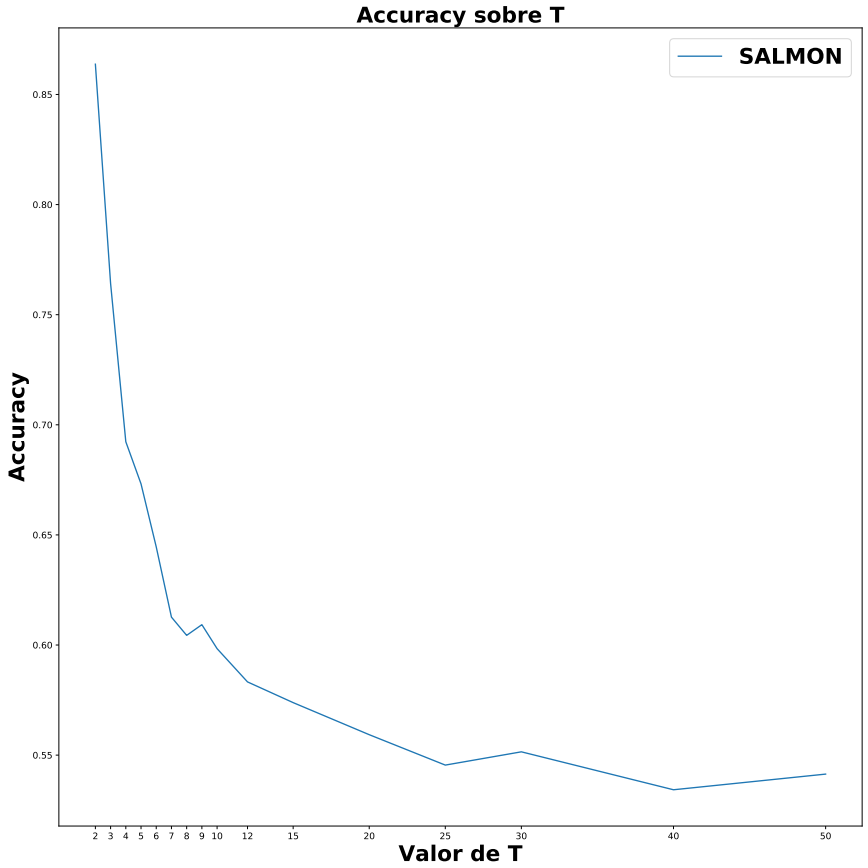


Figura 5.6: Escenario sintético número 3. *Accuracy* del clasificador de *random forest* usado en el SALMON según los valores de T. Observar que baja hasta niveles de error muy altos: una *accuracy* de poco más del 50 % es casi acertar la misma cantidad de veces que errar en la clasificación (que en este caso es binaria). Incluso con valores bajos de T el clasificador no llega al 90 % de *accuracy*.

Luego se ensayan nuestros algoritmos en una situación de 2 rutas y 2 niveles, pero ahora las rutas no siguen un proceso markoviano, sino que se construyen utilizando la distribución de Rayleigh. En este escenario, los algoritmos concuerdan en una política de baja medición: no miden nunca los de base markoviana (*policy* y horizont errante) y el SALMON selecciona una de las combinaciones más bajas de (T_1, T_2). Para el cálculo de las matrices de transiciones se realizó una aproximación frecuentista del modelo markoviano, y la matriz obtenida cuenta con valores muy particulares: todas las probabilidades de transición se encuentran cerca de 0.5, lo que representa un escenario de mucha incertidumbre (valen $P_{est} \approx \begin{bmatrix} 0{,}5 & 0{,}5 \\ 0{,}5 & 0{,}5 \end{bmatrix}$).

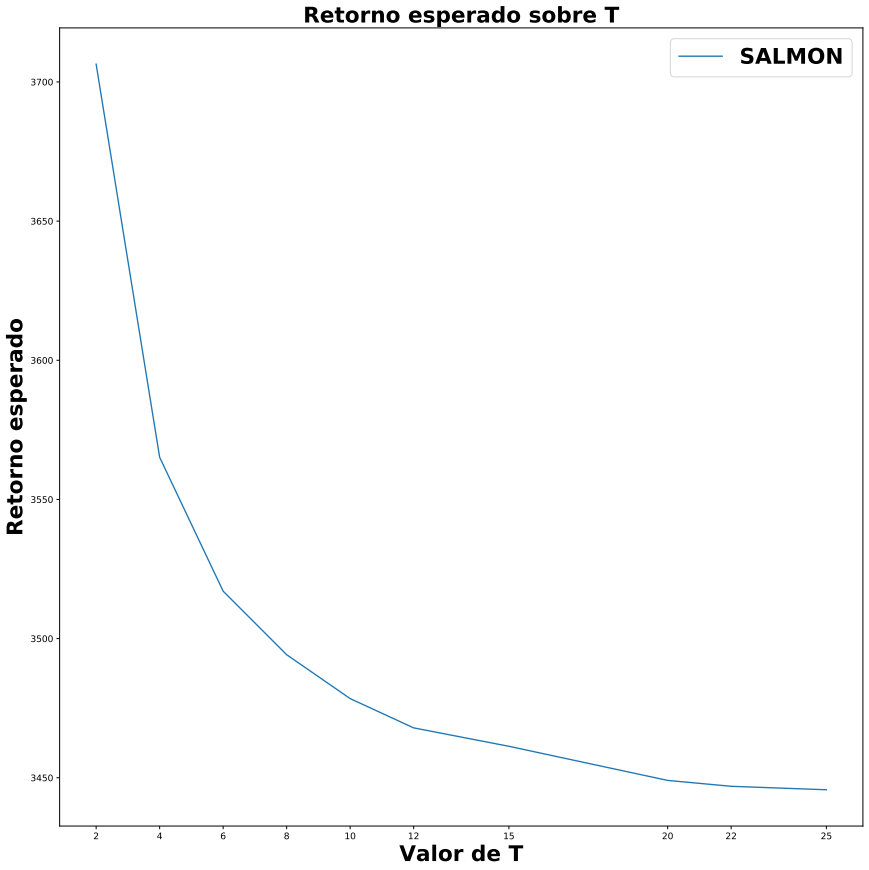


Figura 5.7: Escenario sintético número 4, trazas generadas según una distribución de Rayleigh. Valor esperado del retorno para diferentes valores de T posibles. El mínimo se alcanza en $T = 2$. Luego se elegirá $(1, 3)$ como valores óptimos en un intervalo de T.

Algo interesante es que ambos algoritmos realizan pocas o ninguna medida, y seleccionan una ruta con la que permanecen casi todo el tiempo. Sin embargo, los algoritmos markovianos seleccionan la ruta 2, mientra que el SALMON selecciona la ruta 1. En general, incluso si selecciona un "camino más corto", el costo total (una vez agregado el costo de medición) es mayor para el SALMON, pero todos están realmente cerca uno del otro. Los resultados del retorno esperado y la *accuracy* en función de T se pueden ver en las figuras 5.7-5.8.

Finalmente se generan trazas siguiendo la distribución de Pareto. Esta es una distribución con la característica de generar colas pesadas, que se alejan del escenario markoviano sin caer en algo tan errático como resultó ser Rayleigh. Los resultados fueron incorporados a las tablas 5.2-5.3, y se puede apreciar la evolución del retorno y la *accuracy* sobre T en las figuras 5.9-5.10. Nuevamente, no se reali-

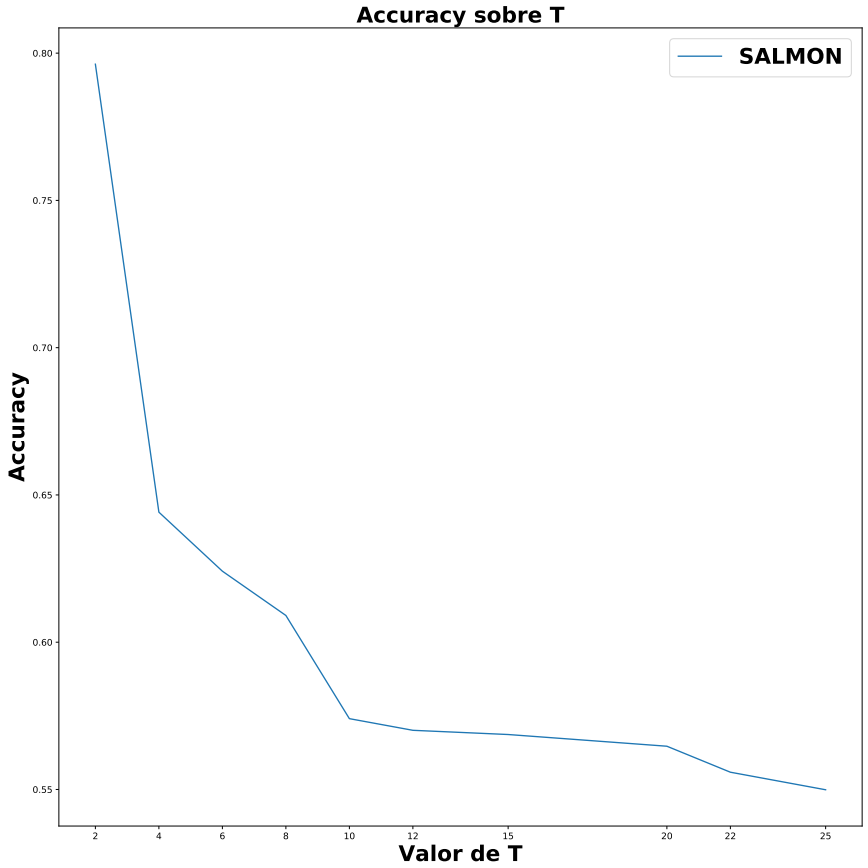


Figura 5.8: Escenario sintético número 4 (Rayleigh). *Accuracy* del clasificador de *random forest* usado en SALMON según los valores de T. Observar que la *accuracy* es de las más bajas, no llegando al 80 % aún en los valores bajos de T.

zan casi medidas de parte de los algoritmos. En este escenario, ambos algoritmos eligen la ruta con menor retardo esperado, y excepto el SALMON en contadas ocasiones, no cambian la elección. Esto se debe a que las matrices de transición halladas tienen mucho peso en el estado bajo de los estados posibles, y al elevarla a la potencia τ converge muy rápidamente a la matriz estacionaria, que también asigna gran probabilidad a estar en nivel bajo. Entonces prácticamente se terminan comparando los niveles bajos de las dos rutas. Las matrices halladas valen $P_{est} \approx \begin{bmatrix} 0{,}91 & 0{,}09 \\ 0{,}9 & 0{,}1 \end{bmatrix}$.

En estos dos últimos casos, se probó a disminuir el costo, para fomentar la medición por parte de los algoritmos. En el caso del escenario siguiendo una distribución de Rayleigh, esto generó que se midieran permanentemente todas las rutas,

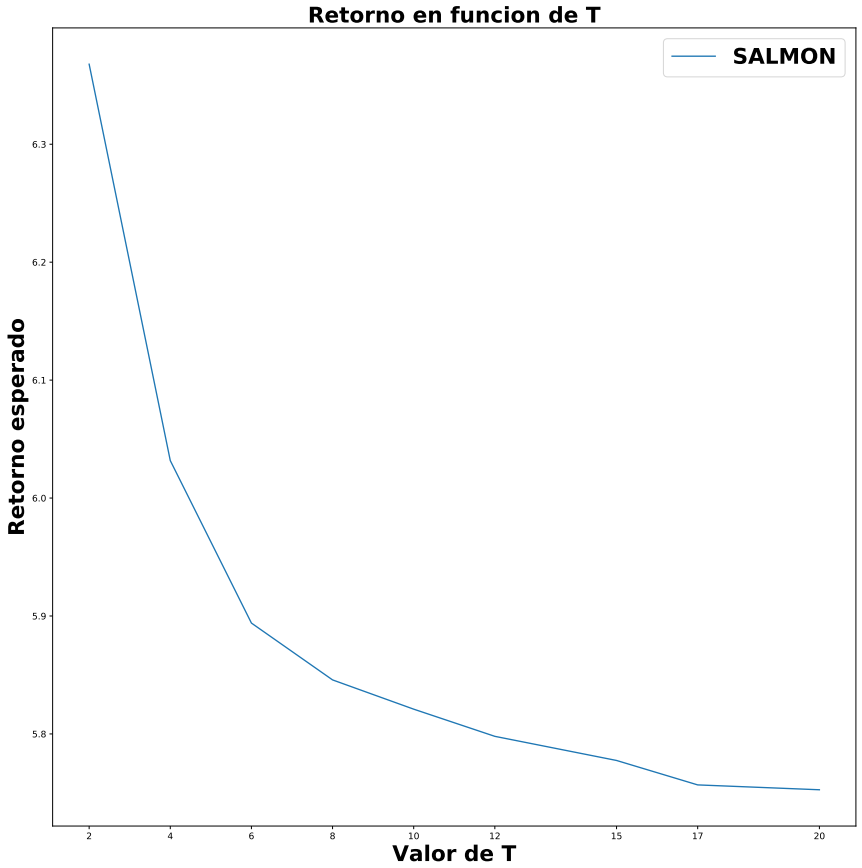


Figura 5.9: Escenario sintético número 5, trazas generadas según una distribución 80-20 (Pareto). Valor esperado del retorno para diferentes valores de T posibles. El mínimo se alcanza en $T = 2$. Luego se elegirá $(1, 3)$ como valores óptimos en un intervalo de T.

dado que hay una gran incertidumbre por tener las matrices de transición casi igual probabilidad de quedar en el mismo nivel como de cambiar de nivel. El valor de $(T_1, T_2) = (1, 3)$ para el SALMON coincide con el incremento de las mediciones. Los tres algoritmos logran elegir siempre la ruta ideal, pero al costo de mediciones casi-permanentes. Cuando se generaron trazas con la distribución de Pareto, esto también sucede pero en menor medida, dado que se sigue manteniendo la creencia de que ambas rutas se encuentren en el nivel bajo. En esta ocasión los algoritmos markovianos logran elegir la ruta ideal el 99 % de las veces con unas 125 mediciones (en 2500 épocas). El valor de $T = (16, 16)$ y el SALMON logra elegir la ruta correcta el 91 % de las veces realizando 312 mediciones.

En todos los casos el *policy iteration* obtiene un retorno esperado menor al resto, seguido muy de cerca o igualado al horizonte errante, y no muy lejos atrás

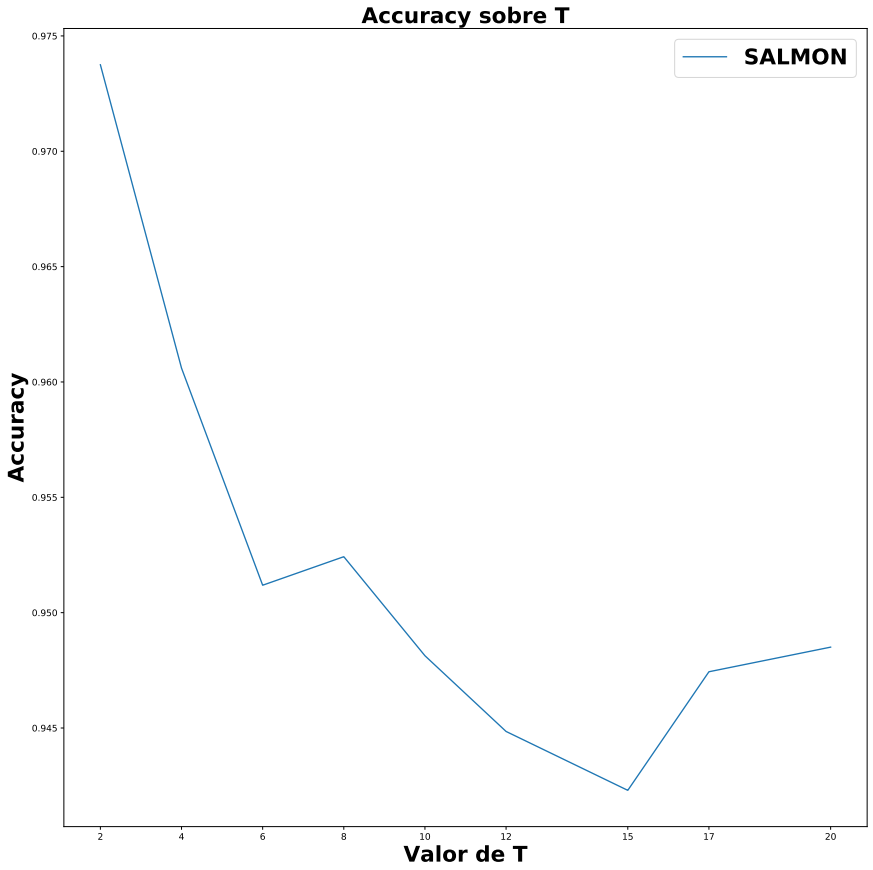


Figura 5.10: Escenario sintético número 5, trazas generadas con la distribución de Pareto. *Accuracy* según los valores de T.

	Costo de medida			**Suma de RTT ruta elegida**		**T**
Escenarios	**PI**	**RH**	**S**	**RH**	**S**	**S**
Márkov 1	7180	7180	**7140**	**872980**	878780	(7,7)
Márkov 2	**405**	435	500	**5066**	5092	(3,1)
Márkov 3	25000	25000	**12500**	**849200**	880650	(4,4)
Rayleigh	**0**	**0**	6160	711570	**709445**	(27,25)
Pareto	**0**	**0**	23.8	1481	**1477**	(21,21)
Márkov 2 (b)	**278**	**278**	500	5103.5	**4976.5**	(1,1)
Márkov 2 (c)	526	480	**143**	5497	6305	(14,14)

Tabla 5.2: Resultados de las decisiones de medida y ruteo para los escenarios sintéticos propuestos. Notar que no siempre el algoritmo con menor retorno esperado es el que elige la ruta más rápida, dado que juega el costo de medida en el retorno esperado.

	% del tiempo en mejor ruta			Retorno esperado		
Escenarios	**PI**	**RH**	**S**	**PI**	**RH**	**S**
Márkov 1	**93.48**	**93.84**	88.24	**3371**	**3371**	3392
Márkov 2	95.58	**95.76**	95.52	**10.52**	10.54	10.71
Márkov 3	**100**	**100**	67.8	**3349**	**3349**	3421
Rayleigh	45.8	45.8	**53.95**	**3407**	**3407**	3426
Pareto	94.32	94.32	**94.48**	**5.68**	**5.68**	5.75
Márkov 2 (b)	97.46	97.46	**100**	**10.30**	**10.30**	10.48
Márkov 2 (c)	**89.7**	89.32	74.3	**11.42**	11.44	12.35

Tabla 5.3: Resultados según las métricas elegidas para los escenarios propuestos. Observar que en muchos casos la política óptima y la política del horizonte errante coinciden completamente. Por otro lado, el *modified policy iteration* obtiene siempre el mejor retorno esperado, aunque no necesariamente elija en todo momento la ruta más rápida, ni realice la menor cantidad de medidas. Esto vale incluso para los escenarios no-markovianos.

el SALMON. Esto es esperado en los escenarios markovianos, aunque no necesariamente en los escenarios no-markovianos. En general la diferencia en el retorno esperado para los diferentes algoritmos es muy pequeña. Es interesante notar la cercanía en las decisiones de medición y ruteo que existe entre el *policy iteration* y el algoritmo de horizonte errante, a pesar de utilizar un paso de $H = 3$, que uno a priori pensaría bajo. Como vimos en el capítulo 3, el considerar solamente unas épocas hacia adelante, a pesar de contar con valores altos de ρ, ya ofrece una solución muy próxima de la óptima.

Al mismo tiempo, tanto el de horizonte errante como el *policy* funcionan muy bien en escenarios markovianos, pero tienen un funcionamiento muy variable en escenarios no-markovianos. El algoritmo de SALMON logra acercarse al óptimo en escenarios markovianos, y mantiene buenos resultados en los escenarios no-markovianos. De hecho, tampoco se despega de lo que le sucede al resto de los algoritmos: si bien su rendimiento es mejor en los escenarios en que estos no miden, sigue siendo pobre en el escenario de Rayleigh. Además, en escenarios no-markovianos y con bajo costo de medida, los algoritmos markovianos vuelven a aventajarlo (o igualarlo). Para el SALMON, la restricción de que T no puede crecer indefinidamente (por construcción de las muestras de entrenamiento del clasificador) impone la realización de medidas. Esto logra que el algoritmo obtenga mejores resultados en cuanto a la elección de rutas pero también un retorno esperado mayor (por sumarse el costo). No parece haber una conclusión determinante sobre una mayor robustez del algoritmo de SALMON, aunque si es notorio que en escenarios no-markovianos los algoritmos de horizonte errante y *policy iteration* funcionan mal, y podría resultar más beneficioso utilizarlo.

Sin embargo, así como no convence demasiado observar el gran desempeño de los algoritmos sobre escenarios markovianos artificiales, tampoco es muy interesante forzar una ruta que sigue una distribución de Rayleigh o de Pareto. La verdadera prueba es el ensayo de los algoritmos sobre trazas reales, a lo que nos

abocaremos en la sección siguiente, luego de comparar los resultados del algoritmo de horizonte errante con otro algoritmo que busca resolver el problema de medición y ruteo planteado.

5.2.1. Comparación con otra heurística que busca resolver el mismo problema

Presentación de la heurística alternativa

Ahora se explicará brevemente una heurística propuesta en [48] por O. Brun, M. Ségnéré y V. Prabhu que busca resolver el exacto mismo problema de ruteo y medición estudiado. Para esto, los autores buscan reducir la dimensión del espacio de estados sobre el que se realiza la iteración de la ecuación de Bellman (2.8). Como se vió, el principal inconveniente de (2.8) es que el espacio de estados crece exponencialmente en el número de caminos a explorar.

Lo que plantean es reducir el MDP a N MDP diferentes, uno por ruta. Integran al nuevo MDP solamente el costo de medir la ruta en cuestión, y resumen la información disponible de las otras rutas en una única ruta determinística que representa los demás caminos. Esta ruta "determinística" representa el mínimo del RTT esperado del resto de las rutas, ya sea que estas son medidas o no. Se busca de esta forma comparar cada ruta contra la ruta más rápida de las restantes. Con esto obtienen un algoritmo de complejidad $O(P(K\tau_{max})^2)$ en cada época de decisión, que resuelve para cada ruta un MDP de dos rutas en la que una es fija y no acepta acciones y la otra es la ruta de la que se busca la acción óptima.

Este algoritmo fue comparado con nuestra propuesta de horizonte errante (capítulo 3). Para esta comparación, y dado que en [41] se había observado que la política miope reportaba muy buenos resultados, primero se utilizan escenarios sintéticos en que se establezca la utilidad de contar con algoritmos mejores que el miope. Luego se compara un escenario de 3 rutas y 2 niveles, en que para distintos valores de ρ se halla el MRE para la política miope, la heurística recién presentada y el *receding horizon*. En estos ejemplos sintéticos no se utilizó el SALMON, siendo que no tiene sentido compararlo a nivel de MRE con una política MDP, y además corre con desventaja en un escenario generado completamente a partir de cadenas de Márkov.

Finalmente, se compara la utilización del horizonte errante y de la heurística alternativa sobre datos de trazas para rutas en que un algoritmo así presentaría ventajas (existen rutas paralelas y más rápidas entre origen y destino que la ruta por defecto).

Serán brevemente presentadas estas pruebas sintéticas, y luego se analizarán los resultados obtenidos sobre trazas reales. Lamentablemente, no se tiene acceso al código de Brun, Ségnéré y Prahbu, por lo que se utilizan resultados presentados en [48].

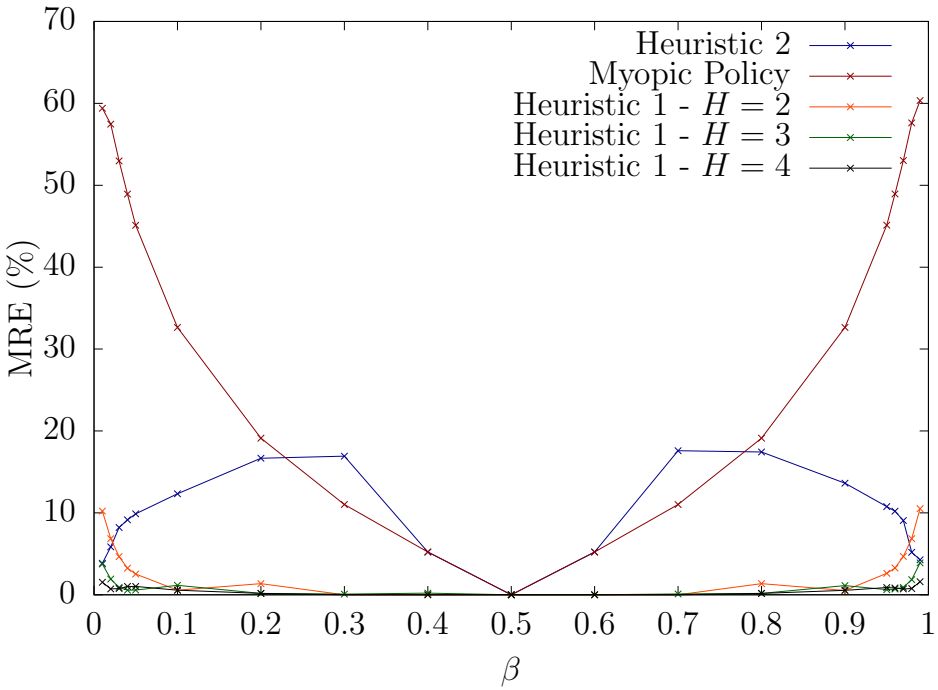


Figura 5.11: Ejemplo 1 - MRE en % de la política miope y de las heurísticas propuestas. La heurística 1 refiere al horizonte errante, con la profundidad indicada, mientras que heurística 2 es la propuesta por *Brun et al.* . Notar que la forma de la curva del horizonte errante se mantiene, a pesar de aplanarse el error al aumentar la profundidad. En la grafica se observa que para matrices muy estables (con mucho peso en las diagonales), es necesario un horizonte más grande que para rutas más cambiantes. Tomado de [48].

Escenarios sintéticos

El primer ejemplo permite mostrar que la política miope puede tener un desempeño relativamente malo en comparación con los algoritmos presentados. Hay dos caminos con dos estados cada uno, en que los retardos están dados por $\boldsymbol{\ell}_1 = \boldsymbol{\ell}_2 = (0{,}01, 100)$, y las matrices de transición son:

$$P_1 = P_2 = \begin{pmatrix} \beta & 1-\beta \\ 1-\beta & \beta \end{pmatrix}.$$

Utilizando $c = 25$, $\rho = 0{,}99$ y diferentes valores de β, se calcula la política de monitoreo óptima usando *modified policy iteration*. Las políticas halladas con los algoritmos propuestos se han evaluado utilizando el algoritmo *Policy Evaluation* [66]. El error relativo medio (MRE), definido como $\frac{1}{S}\sum_{\mathbf{s}} \frac{|V_\mu(\mathbf{s}) - V^*(\mathbf{s})|}{V^*(\mathbf{s})}$, se muestra en la Figura 5.11.

Se observa que la política miope funciona mal cuando las transiciones de las cadenas de Márkov ocurren con poca frecuencia y hay una gran diferencia entre los retardos de los dos estados.

En el segundo ejemplo, se busca mostrar que la política miope puede ser mejor o igual que ambas heurísticas en algunos casos. Considere el siguiente ejemplo con

ρ	c	MRE of Heurística (%)	MRE de Política miope (%)	MRE de Horizonte Errante (%)
0.99	0.5	9.19	0.60	0.08
0.99	0.25	6.68	0.88	0.20
0.99	0.125	7.21	0.10	0.10
0.99	0.0625	7.86	0.15	0.15
0.999	0.5	11.97	0.724	0.19
0.999	0.25	7.45	1.00	0.30
0.999	0.125	5.50	0.23	0.23
0.999	0.0625	6.97	0.27	0.27

Tabla 5.4: Ejemplo 2: Errores relativos medios sobre todos los estados de las heurísticas propuestas. Para el horizonte de retroceso se utiliza un horizonte de H = 3, sin la restricción de medir una sola ruta en cada paso.

tres caminos estocásticos con dos estados cada uno. Los retardos están dados por $\boldsymbol{\ell}_1 = (1, 3)$, $\boldsymbol{\ell}_2 = (\frac{1}{2}, 4)$ y $\boldsymbol{\ell}_3 = (\frac{1}{4}, \frac{15}{4})$, y las matrices de transición son las siguientes:

$$P_1 = \begin{pmatrix} 0{,}9 & 0{,}1 \\ 0{,}1 & 0{,}9 \end{pmatrix}, \; P_2 = \begin{pmatrix} 0{,}8 & 0{,}2 \\ 0{,}3 & 0{,}7 \end{pmatrix}, \; P_3 = \begin{pmatrix} 0{,}65 & 0{,}35 \\ 0{,}35 & 0{,}65 \end{pmatrix}.$$

Tenga en cuenta que los retardos esperados en el estado estacionario de las tres rutas están muy cerca (1,99, 1,9 y 2). Se considera que existe una aproximación razonable de la matriz de transiciones estacionaria para $\tau^1_{max} = 20$ para la ruta 1, mientras que se usa $\tau^2_{max} = \tau^3_{max} = 10$ para los otros dos caminos. Esto produce $S = 16,000$ estados del sistema. La tabla 5.4 proporciona el MRE para los algoritmos y para diferentes valores de ρ y c. Se desprende de los dos ejemplos anteriores que los algoritmos propuestos funcionan bien para todos los valores de β, siempre por debajo del 20 % del MRE, mientras que la política miope es mucho más variable, principalmente para rutas con valores muy cercanos a 0 o 1 en la diagonal (la simetría de la matriz se refleja en una simetría axial sobre $\beta = 0{,}5$ en la figura 5.11).

Este último ejemplo, si bien fue publicado en el artículo mencionado ([48]), se estima que cuenta con una desventaja importante. El asunto es que el valor de $\tau_{\text{máx}}$ utilizado es muy bajo como para que las matrices de transición se hayan acercado al valor estacionario en que $P^{\tau_{\text{máx}}} \approx P^{\tau_{\text{máx}}+1}$. Sin embargo, permite comparar los resultados de las tres políticas estudiadas, y siendo que las tres parten de un escenario markoviano, las limitaciones que surgen de la aproximación planteada afectarán a las tres (aunque difícilmente por igual). En particular para el resto de los ejemplos se han desechado los casos en que no se pueda realizar esta aproximación de manera razonable. Sin embargo, esto reduce las posibilidades a comparar sobre escenarios sintéticos de solamente dos rutas.

5.3. Trazas reales

Se ha visto que en escenarios sintéticos ambos algoritmos propuestos ofrecen buenos resultados. Sin embargo, la realidad siempre es más compleja aún que los modelos más complejos. En particular, para el algoritmo de horizonte errante se realiza una asunción muy fuerte sobre que los retardos de las rutas siguen un comportamiento markoviano. Si bien hay numerosos trabajos al respecto, esto no siempre se cumple igual de bien. Hay rutas más markovianas y rutas menos markovianas, si se quiere. O mejor planteado: es una aproximación válida para varios casos, pero ni generalizable a todos los casos; es una aproximación buena para muchos casos, pero no exacta para ninguno. Esto no implica necesariamente que los algoritmos derivados de estas asunciones funcionen mal, pero por ejemplo cambia el concepto de optimalidad.

El óptimo existe en relación a un problema definido, según el problema planteado originalmente la solución óptima sería siempre elegir la ruta correcta sin medir nunca. Si se pudiera predecir con exactitud el desarrollo de los retardos futuros, se podría considerar esta solución dentro del universo de soluciones posibles, y en ese caso incluso no sería necesario incluir el problema asociado al costo de medida. Esta podría ser una (arriesgada) línea de trabajo a futuro, donde indudablemente se realizarán aproximaciones, y donde una vez más no se estará en el escenario realmente ideal de nunca errar de ruta y nunca medir. En ese sentido, la incapacidad de predecir nuevos estados de la red dificulta el trabajo de algoritmos que no se (re)alimentan de la información que va apareciendo (a través de medidas, ¿cómo sino?).

Tanto el estudio de las rutas como MDP como la aproximación por aprendizaje supervisado exigen un período de entrenamiento, en el caso de SALMON para entrenar el clasificador y seleccionar el T^* y en el caso del MDP para conocer las matrices de transición P^i y los niveles K^i. El algoritmo de horizonte errante cuenta con la ventaja de trabajar en línea, pero utiliza los valores mencionados, realizando una abstracción de los RTT reales a los K niveles.

Se buscará entonces utilizar métricas que permitan afirmar "se ha ganado algo al usar este algoritmo". Las métricas serán:

- **Comparación con la ruta elegida por defecto:** el problema surge de la existencia de rutas paralelas y más rápidas. Encontrar y aprovechar esas rutas sin causar congestión por el costo de medida es entonces lo primero que se desea lograr.

- **Cantidad de medidas realizadas:** la selección de la ruta más rápida exige la realización de medidas, nos interesa en segundo lugar que estas medidas no generen congestión o retardos en la red.

- **Cercanía a la ruta más rápida:** no alcanza con disminuir la latencia, nos interesa disminuirla lo más posible. En este sentido se podría pensar en el "óptimo" del problema real, que sería simplemente la ruta más rápida en toda época y sin realizar mediciones. Sólo comparando el tiempo de ida y

Figura 5.12: Mapa de distribución de nodos (agrupados) de NLNOG Ring, obtenido de http://map.ring.nlnog.net/, consultada el 30/07/2020. Los nodos se encuentran.

vuelta y sin incluir el costo de medida se puede ver qué tan cerca se está de este "óptimo" inalcanzable.

- **Tiempo de ejecución:** la selección de la ruta por parte del algoritmo debe realizarse en un tiempo menor a una época.

- **Escalabilidad:** se evaluará hasta qué cantidad de rutas los algoritmos logran buenos resultados según las métricas anteriores.

Para aplicar nuestros algoritmos al mundo real, se considerará una serie de mediciones realizadas entre 20 nodos del *NLNog ring*, ver figura 5.12. Estas mediciones fueron recopiladas en 2014 realizando medidas cada dos minutos usando **ping** (del protocolo ICMP), durante 5 días y resultando en 3836 medidas (se pueden encontrar más detalles al respecto en [29]).

Las medidas fueron realizadas entre todos los nodos en topología de malla completa, lo que es conveniente para encontrar rutas paralelas. Estas serán simuladas utilizando el siguiente criterio: si entre los nodos A y B puedo ir a través del nodo C, entonces el RTT de A a B pasando por C será la suma de la medidas A-C y C-B. Para estimar los casos útiles, se consideran aquellos en que al menos el 10 % del tiempo existen rutas más rápidas pasando por algún nodo intermedio (o "proxy"). Este procedimiento fue el utilizado en [48] y [29], sobre las mismas rutas. Conviene una aclaración final sobre el uso a veces indistinto de RTT y latencia. En el estudio de las trazas y a lo largo del trabajo, se ha asumido que las rutas son lo suficientemente simétricas como para considerar (algo usual en muchos trabajos) que la latencia es la mitad del tiempo de ida y vuelta.

En particular, en [48] se consideran 4 pares origen-destino para los cuales se puede lograr una disminución significativa en el RTT seleccionando una ruta alternativa, y en que además esta ruta alternativa no es fija. Esto es importante pues permite ver la adaptación del algoritmo en la elección de diferentes rutas, si la ruta alternativa rápida fuera fija no se necesitaría realizar medidas. A nivel del

Par Origen-Destino	H	N	$\#\,\mathcal{S}$	Tiempo de procesamiento		
				Miope (ms)	HE (s)	S (ms)
Paris-Tokyo	3	2	$360,000$	1.56	0.06	2.2
Singapore-HongKong	3	3	$1{,}68 \times 10^8$	5.85	6.57	7.67
Haifa-Santiago	3	4	$2{,}88 \times 10^{10}$	6.95	30	12.46
Curitiba-Calgary	2	4	7×10^{10}	11.7	3.63	7.66
Paris-Santiago	3	4	9×10^{10}	-	35.8	-

Tabla 5.5: Número de caminos y de estados para los caminos seleccionados. Se aclara también el horizonte que fue utilizado durante las pruebas para los distintos caminos.

MDP sería como plantear que los K^i niveles de la ruta más rápida son menores a cualquiera de los K^j niveles de las otras rutas.

El modelo de Márkov para cada ruta se aprendió utilizando el enfoque HDP-HMM (Modelo de Márkov Oculto del Proceso Jerárquico de Dirichlet) descrito en [40]. A diferencia del HMM clásico, en que se debe conocer el número de estados en el momento del aprendizaje, el HDP-HMM trata el número de estados como desconocido y lo establece a partir de los datos, junto con los otros parámetros del modelo. Esto permite obtener modelos de markov para mediciones reales de demoras en Internet donde no se conoce la verdad sobre el número de estados de red. Este procesamiento de los datos fue realizado por el grupo de trabajo de IMT-Atlantique. A los efectos de las pruebas realizadas se simplificaron los modelos HMM a modelos markovianos definidos por la matriz de transición y los niveles, descuidando el ruido gaussiano del modelo HMM y considerando solo los valores medios. También vale la pena mencionar que, dado que no se observa directamente el estado de una ruta monitoreada, este estado se estimó como el estado más probable del HMM dadas las observaciones pasadas.

A los efectos del algoritmo de horizonte errante, este utilizará los niveles que toma cada ruta según el modelo markoviano. Por otra parte, el SALMON será alimentado directamente con las series temporales del RTT, con las que construirá las muestras de entrenamiento para el clasificador como fue explicado en el capítulo 4. Ya en esta diferencia al alimentar los algoritmos se observa que la aproximación markoviana puede introducir errores.

Los pares origen y destino seleccionados se pueden ver en la Tabla 5.5 y la figura 5.13. En la misma tabla se plantean los tiempos de procesamiento por época para cada algoritmo. Para las rutas y las implementaciones seleccionadas el tiempo de procesamiento no es una restricción. Claro que aumentando H, esto aumentará los tiempos del horizonte errante, pero póngase esto a un lado por ahora, dado que luego se estudiará el límite de este algoritmo para escenarios reales. Para tener como referencia, el cálculo de la solución óptima por *policy iteration* para el escenario con menos rutas y menos niveles (Paris-Tokyo) exige iterar sobre $(100 * 6)^2 = 360000$ estados. En este caso se considera $\tau_{\text{máx}} = 100$.

Los tiempos de procesamiento son aceptables para épocas que duran dos minutos. Sin embargo, se observa un incremento notorio tanto al crecer H como (y

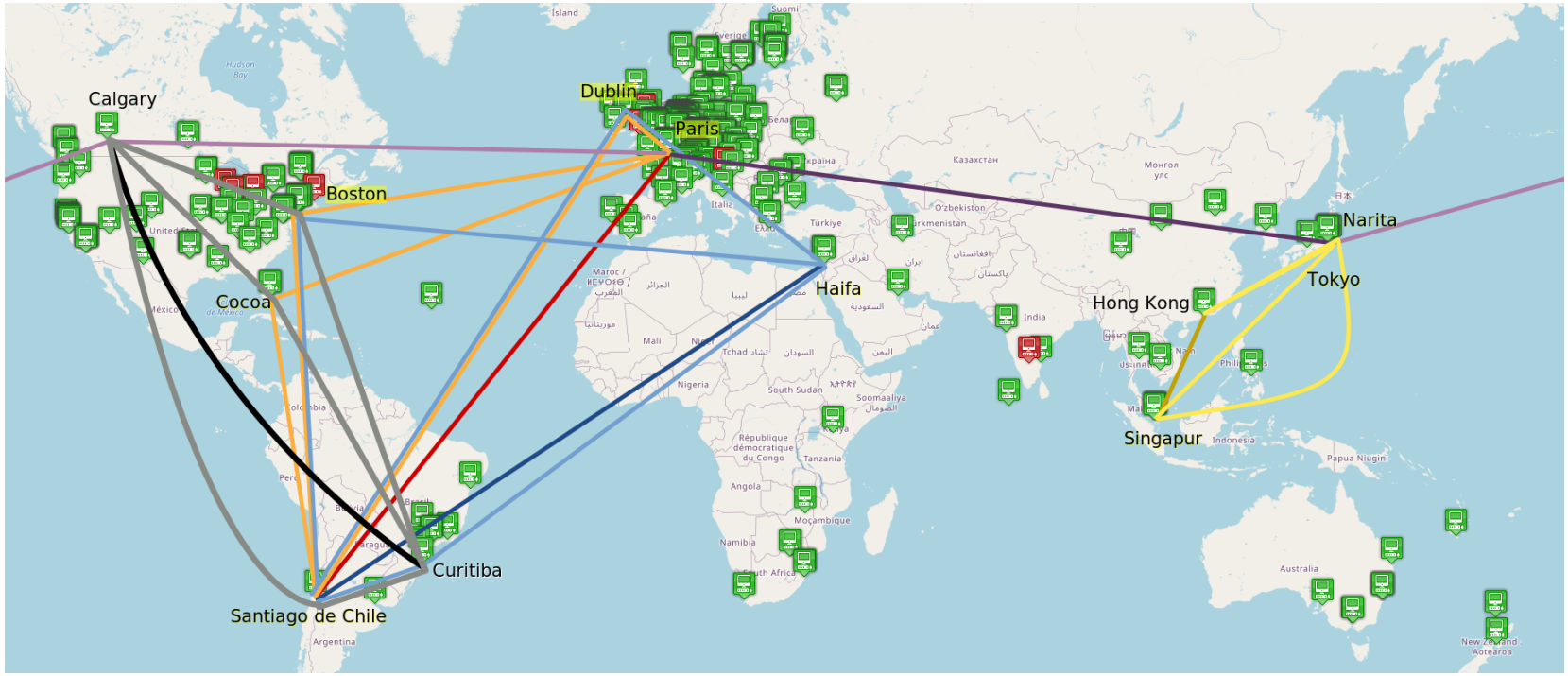


Figura 5.13: Mapa de nodos de NLNOG Ring, obtenido de http://map.ring.nlnog.net/, consultada el 30/07/2020, modificado por el autor para mostrar los pares O-D elegidos y las rutas alternativas. En azul oscuro está la ruta Haifa-Santiago, y en azul claro las alternativas (pasar por Boston, Curitiba, Paris). En rojo Paris-Santiago y en naranja las alternativas (Dublin, Boston, Cocoa), que se usaran para la comparación con la heurística alternativa. En violeta Paris-Tokyo y en lila pasar por Calgary. En dorado se aprecia Hong Kong-Singapur y en amarillo las alternativas Tokyo y Narita. Finalmente, en negro Curitiba-Calgary y en gris las alternativas via Boston, Santiago y Cocoa.

Par	Cantidad de medidas			Distancia a ruta ideal (en 10^{-3})			% del tiempo en ruta ideal		
O-D	M	HE	S	M	HE	S	M	HE	S
P-T	**0.002**	0.017	0.07	0.4	**0.2**	1	97.2	**98.3**	92.6
S-HK	**0.43**	0.56	0.75	**4**	**4**	14	**99.1**	**99.1**	98.3
H-S	**0.12**	0.25	0.80	0.9	**0.8**	14	94.5	**94.6**	78.6
C-C	**0.05**	0.00	0.13	**0.9**	1	2.5	**97.9**	97.2	94.4

Tabla 5.6: Resumen de los resultados logrados para los pares OD considerados, donde M es la política miope ($H = 1$), HE representa al horizonte errante, S el algoritmo de SALMON. Los promedios se calculan por intervalo de tiempo. Observe que no siempre que aumenta H se refleja en un resultado mejor para el conjunto de las métricas planteadas.

principalmente) la cantidad de rutas. El límite para aplicar un horizonte de $H = 3$ son $N \approx 4$ rutas. Al mismo tiempo, estos tiempos están vinculados a la cantidad de niveles que puede tomar cada ruta del sistema, así que es posible que algún escenario de 5 niveles se pueda realizar con $H = 3$. Por otro lado, depende (y mucho) de la capacidad de procesamiento y memoria de la computadora en que se corre el algoritmo. De contar con memoria y poder usar la versión de reutilización de memoria se ahorraría un tiempo importante. Sin embargo, al probar con grandes cantidades de rutas de muchos niveles y valores altos de $\tau_{\text{máx}}$ la utilización de memoria se vuelve prohibitiva. Finalmente, para $H = \{1, 2\}$ se puede usar el algoritmo de horizonte errante funcionando como máximo en el orden de los pocos segundos para todos los pares Origen-Destino con que se cuenta en la base de datos relevada de NLNOG.

Para los escenarios analizados se considera un costo $c = 0{,}5$ igual para todas las rutas, y un factor de descuento de $\rho = 0{,}9$. Recuerde que el costo nos permite formalizar el problema de optimización que minimiza el costo de medida, para tener una estimación realista del costo se debería conocer el impacto que tiene enviar paquetes de medida para cada ruta y en simultáneo (algo similar a lo realizado en [30]). Por otro lado no tendría porqué ser proporcional a la cantidad de medidas como se utiliza en esta Tesis, dado que se desea evitar causar congestión, y esto no sucede "proporcionalmente" al envío de paquetes de medida. Esta discusión excede el alcance de la Tesis, aunque no es un asunto menor en tanto un costo muy alto disminuirá la cantidad de medidas a realizar, mientras que uno muy bajo las fomentará. La formulación misma del problema de optimización va a regular la cantidad de medidas en función del costo, con lo que funciona para cualquier costo, pero no tiene mucho sentido pensar en la aplicación de estos algoritmos en escenarios en que no se puede medir porque el costo es prohibitivo, o en escenarios en que medir tiene costo 0.

Se comparan entonces los resultados de aplicar los algoritmos de horizonte errante y SALMON a las rutas seleccionadas. En las Tablas 5.6-5.7 se encuentra un resumen de algunas de las métricas que interesan: la distancia a la ruta de mínimo retardo (da un indicador de cuanto hay para mejorar), el tiempo medio, la cantidad de medidas, el valor de T elegido y la cantidad de árboles usados.

		RTT medio (ms)				**% de ganancia en el RTT**		
O-D	(T_1, T_2)	Estática	Miope	HE	S	Miope	HE	S
P-T	(13,15)	247.73	247.74	**247.59**	247.75	0	**0.05**	0
S-HK	(2,2)	42.64	42.21	**42.15**	42.16	0.98	**1.15**	1.13
H-S	(5,5)	307.08	302.93	**302.91**	306.87	1.35	**1.36**	0.07
C-C	(15,15)	240.89	**210.63**	210.77	210.80	**12.6**	12.5	12.5

Tabla 5.7: Resultados obtenidos sobre las trazas consideradas. Se aprecian el tiempo medio de la ruta elegida, la ganancia sobre la ruta por defecto (en porcentaje), y el valor seleccionado de T.

Es importante destacar la buena performance de ambos algoritmos. El SALMON obtiene resultados realmente cercanos al horizonte errante, excepto en el caso de la ruta Haifa-Santiago en que no logra mejorar prácticamente. Para entrenar al clasificador de *random forest* se utilizó un bosque de 10 árboles en el caso Paris - Tokyo, mientras que se utilizaron 100 árboles para el resto de los escenarios.

En las figuras 5.14–5.17 se pueden ver ver los resultados obtenidos con el horizonte errante y el SALMON. Se utilizó un horizonte de 3 para tener tiempos de procesamiento razonables, obteniendo buenos resultados, excepto en Curitiba-Calgary en que $H = 2$. La reducción del RTT es a menudo considerable, principalmente cuando los caminos son más factibles de ser modelados como un proceso markoviano. Sumado a la reducción en el tiempo, se aprecia que en el algoritmo de horizonte errante raramente se miden rutas: el peor de los casos se encuentra en el escenario Haifa-Santiago, donde aproximadamente se mide una ruta cada tres épocas. Para el SALMON no es tan trivial, dado que alcanza que algunas rutas sean muy cambiantes para que determinen un valor bajo de T. Los resultados del valor elegido de T así como de las características de los experimentos se pueden encontrar en la tabla 5.7.

En todos los casos se puede notar una reducción del tiempo de ida y vuelta. Sin embargo, se debe señalar que esto no siempre es posible, y depende del escenario. A veces no se logran tiempos muy inferiores por las propias características del escenario, y estar muy cerca todos los valores de RTT posible. En otras ocasiones, picos o mesetas en los retardos que pueden escapar a la aproximación markoviana pueden ser una causa de error para el horizonte errante. Por ejemplo, para horizontes más pequeños que 3 en la ruta Paris-Tokyo no se detecta el nivel bajo de la ruta alternativa.

Para el algoritmo de horizonte errante no se utilizó la restricción de medir una sóla ruta por época. Esta restricción permite obtener tiempos de procesamiento muy inferiores a la versión sin restricciones, pero al mismo tiempo anula las garantías de convergencia a la política óptima markoviana. En particular, para dos de los escenarios estudiados, al utilizar esta restricción empeoraron las decisiones de ruteo y medición, obteniendo resultados peores que con la política miope, y peores que con un horizonte más bajo y sin la restricción. Es que como vimos en el capítulo 3 destinado a explicar el algoritmo de *receding horizon*, ya con una

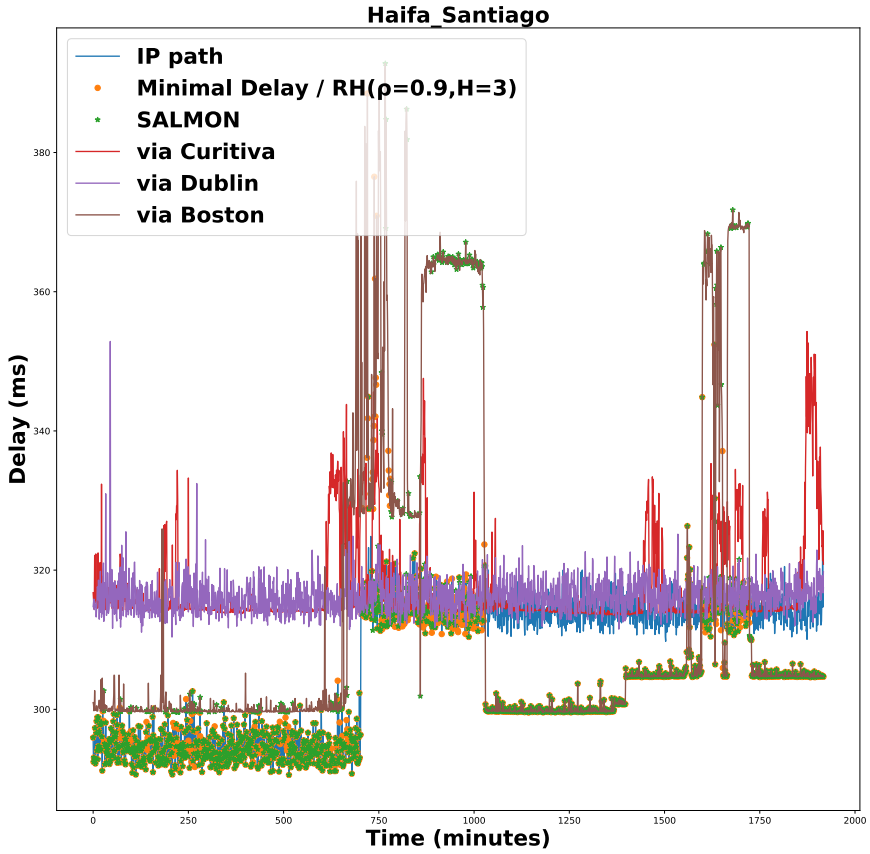


Figura 5.14: Demora de extremo a extremo entre Haifa y Santiago. Este ejemplo no es muy bueno en cuanto a la adaptación de los algoritmos a distintas rutas, puesto que el horizonte errante con $H = 3$ se mantiene fijo en la ruta de menor retardo. Sin embargo, permite ver la ganancia que se obtiene con el algoritmo en los escenarios más ventajosos, en que muy pocas medidas son necesarias.

profundidad baja del horizonte se logran disminuir drásticamente las diferencias entre las políticas π^H y π^*.

En el caso del SALMON se utilizó la selección de un único T para todas las rutas, sin la mejora de revisar en un entorno de T, excepto para el escenario París-Tokio. En este se utilizó la modificación de búsqueda de T a través de relajar a valores distintos por ruta, y el resultado es un $(T_1, T_2) = (13, 15)$. La recompensa esperada para las combinaciones de alrededor de $(14, 14)$ se muestra en la figura 5.18, en relación con la recompensa mínima esperada (solo para visualización). La traza se puede observar en la figura 5.17, donde se compara el algoritmo de horizonte errante y el SALMON relajado.

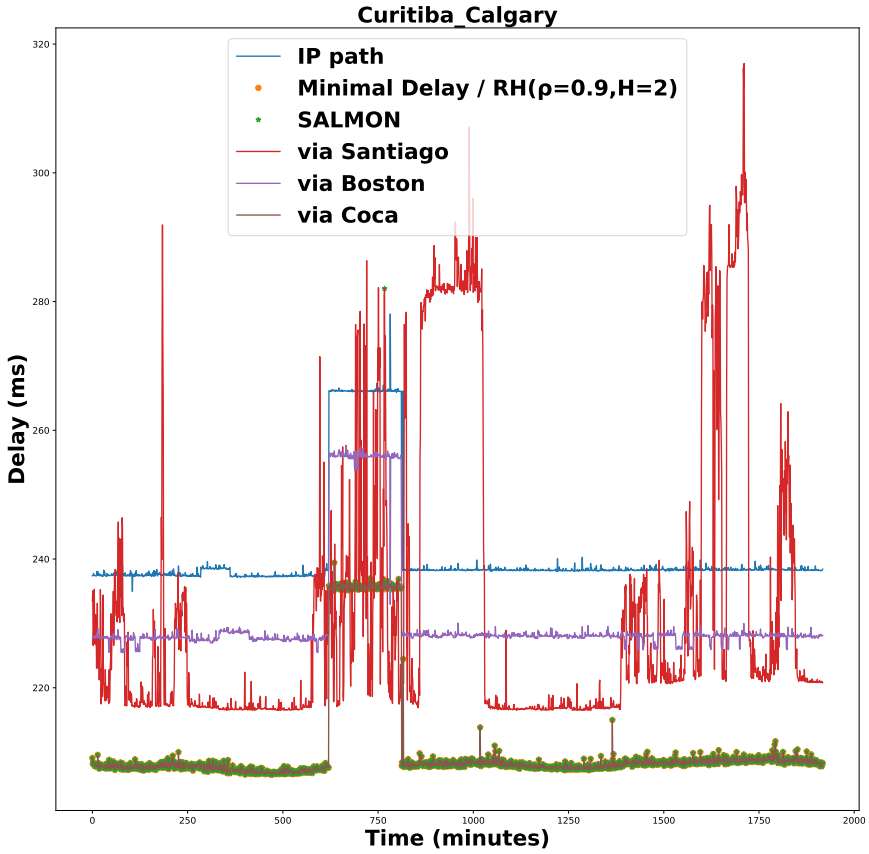


Figura 5.15: Demora de extremo a extremo entre Curitiba y Calgary. Este ejemplo no es muy bueno en cuanto a la adaptación de los algoritmos a distintas rutas, puesto que el horizonte errante con $H = 2$ se mantiene fijo en la ruta de menor retardo. Sin embargo, permite ver la ganancia que se obtiene con el algoritmo en los escenarios más ventajosos, en que muy pocas medidas son necesarias.

La combinación seleccionada ofrece una recompensa esperada más alta, pero no necesariamente implica un algoritmo de costo de medición menor, ya que para un camino aumentan las mediciones mientras se reduce este valor para el otro. Una de las ideas subyacentes a esta Tesis siempre es reducir el costo de medida. En este caso, siendo 0,5 el costo de medida, no se obtiene una reducción del costo de medida, que pasa de $c = \frac{L_{test}*1}{14}$ a $c = \frac{L_{test}*14}{13\times15}$, que es ligeramente superior. Sin embargo mejora la *accuracy* de la predicción, lo que hace que disminuya el retorno esperado y se opte por esta combinatoria.

Una observación interesante es que a medida que crece el número de caminos, el algoritmo SALMON parece tener un rendimiento más pobre. Esto puede

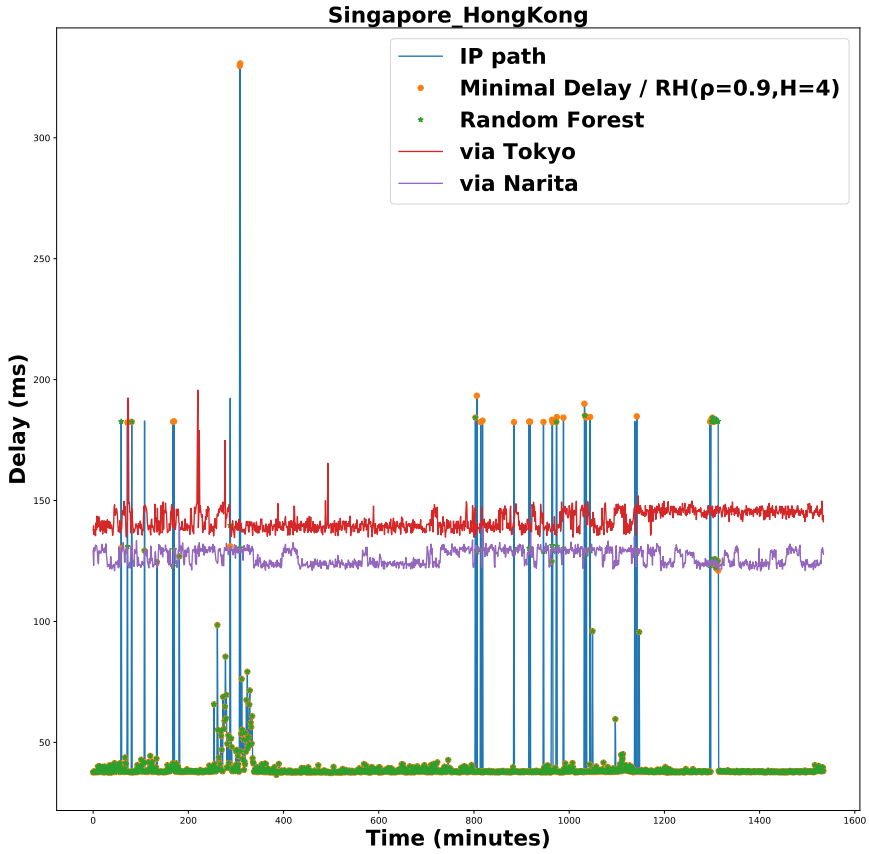


Figura 5.16: Demora de extremo a extremo entre Singapur y Hong Kong para los algoritmos de horizonte errante (H=3) y SALMON (T=2).

deberse a la necesidad de un conjunto de entrenamiento más grande para tener una predicción precisa. Es posible que el clasificador no alcanza a identificar una estructura razonable para el etiquetado del espacio de estados para las pruebas de Haifa-Santiago y Curitiba-Calgary, que cuentan con 4 rutas posibles, y por ende tienen mayor grado de complejidad. Para analizar esto, se utiliza el 50 % de las muestras para entrenamiento y validación, y se comparan los resultados del clasificador en dos escenarios diferentes. Para el caso Paris-Tokyo de dos rutas y con los valores de $T \in [2, 20]$, la *accuracy* siempre está por encima del 95 %. El escenario Haifa-Santiago consta de 4 rutas, y la accuracy para los mismos valores posibles de T se ubica entre el 85 y el 90 % de aciertos. Es notoria la diferencia entre los resultados logrados por el clasificador de *random forest* en ambos casos.

Se buscó aumentar la cantidad de árboles utilizados para los escenarios más complejos, pensando que de esta forma podría mejorar la clasificación, pero esto

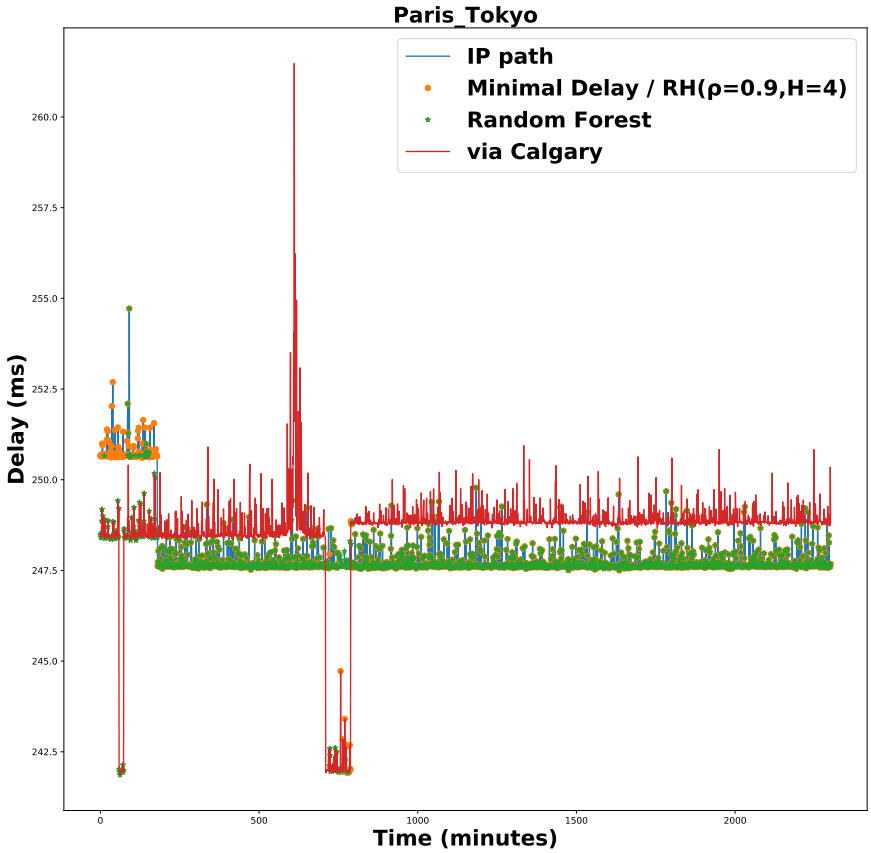


Figura 5.17: Retardos de las rutas Paris-Tokyo y Paris-Calgary-Tokyo, siguiendo las decisiones de ruteo de los algoritmos de SALMON y horizonte errante ($H = 3$). El retardo promedio de la ruta estática IP es de 247.74 ms, el logrado con el *receding horizon* es de 247.59 ms y el SALMON logra un resultado mediocre, con un retardo promedio de 247.75 ms.

tuvo un impacto muy fuerte en el tiempo de procesamiento, y un impacto casi nulo en la *accuracy*. Tampoco ampliar las series de entrenamiento permitió observar cambios notorios, y a su vez dificulta el análisis de las ventajas obtenidas, por ser muy pequeñas las series resultantes para test. Por otro lado tener una tasa alta de aciertos puede ser engañoso, como es el 90 % que se obtiene con el SALMON en Haifa-Santiago. Justamente, ese 10 % de las veces en que el clasificador erra es tal vez el 10 % de las muestras en que hay una ruta alternativa mejor, es decir que son las muestras en que se puede ganar algo. Si una ruta fuera la mejor el 90 % del tiempo, y el algoritmo aprende a elegir siempre esa ruta, tendríamos aciertos por ese 90 %, pero no estaríamos aprovechando el escenario para disminuir el RTT como nos interesa.

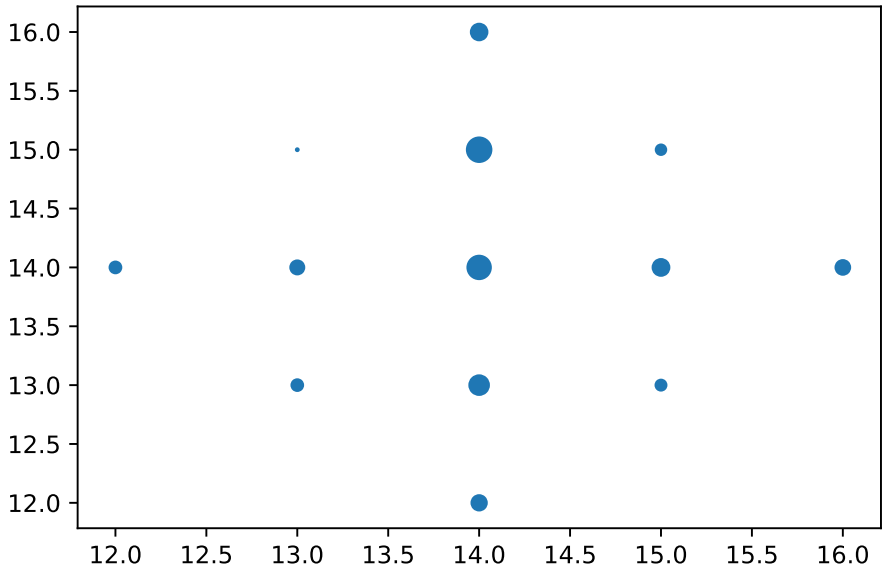


Figura 5.18: Recompensas esperadas para el algoritmo SALMON entrenado con diferentes valores de (T_1, T_2) alrededor de $(14, 14)$. Para la visualización, las recompensas esperadas se trazan en relación con el mínimo alcanzado, en este caso, por $(13, 15)$.

Un aspecto no menor del algoritmo SALMON es que exige una cantidad relativamente importante de muestras de entrenamiento y validación. Esto es porque el clasificador debe ser entrenado, y luego validado para la elección de T. Ya de por sí se tienen solamente 3836 muestras, que tendrán que separarse en entrenamiento y validación. Según que tanto permiten las muestras de entrenamiento capturar el comportamiento de las rutas se obtienen mejores o peores clasificadores. Se utilizó para los resultados el 50 % de las muestras para entrenamiento y validación, y el restante 50 % para test. Para la validación se utiliza *5-fold cross-validation*, lo que permite afinar el lápiz para la selección del T y evitar casos en que una partición poco representativa de las muestras sesgue el clasificador. Al mismo tiempo, puede ser que el tramo reservado para test, que debe ser contiguo, no tenga cambios de niveles en las rutas que permiten apreciar el funcionamiento de los algoritmos. Por este motivo se utilizó la ruta Curitiba-Calgary en vez de la ruta Paris-Santiago (que se utiliza en el artículo [48]), dado que carecía de interés en la región de test.

Recuérdese que para el cálculo del retorno esperado durante la validación y el test se debe contar con bloques contiguos de muestras, lo que también limita la posibilidad de hacer validación cruzada con un factor más alto. Usando el 50 % de las muestras para entrenamiento, de las cuales 12,5 % para validación, son unas ≈ 500 muestras. Si el valor de T es de 10, se pasa 50 veces por cada estado, lo cual es razonable para promediar el retorno esperado desde ese estado. Sin embargo, en la búsqueda de T se llegan a realizar medidas hasta cada $T = 20$ épocas (según

el valor máximo de T, en este caso 20). En este último caso (el más extremo) hay solamente 25 pasadas para la estimación del retorno esperado. Esto es una limitante para la elección del valor máximo posible de T. Incluso más, en la confección de las series de entrenamiento cuando $T = (T_1, T_2)$ estas se generan a partir de desplazar $T_1 \times T_2$ muestras a través de los $L_{ent} - T_1 \times T_2$ posibles inicios de medición que generan estados diferentes. Este último número, $L_{ent} - T_1 \times T_2$ puede resultar muy chico con sólo el 50 % de las muestras.

Claramente la cantidad de muestras con que se cuenta es una limitante para el entrenamiento de algoritmos de aprendizaje supervisado, aunque probablemente otros métodos que no son el *random forest* puedan obtener mejores resultados.

Se realiza a continuación un breve análisis de lo que se conoce como *feature importance*, que permite interpretar la lógica que sigue el clasificador. Es decir buscar cuales son las características de la entrada que tienen mayor peso para decidir sobre la etiqueta (clase) correspondiente. En todos los casos, la entrada menos importante fue el tiempo: cuanto tiempo atrás se realizó la medida. Esta característica tiene un peso muy bajo, del orden de 0.1 o menor (cuanto más cerca de 1 es más importante esa característica). Es decir que el clasificador de *random forest* considera prácticamente sólo cúal es el nivel medido de las rutas. De estas, dependiendo de T y entonces de lo que podía observar, siempre asigna un peso superior a la ruta más cambiante. En el caso de contar con una ruta que cambia mucho y otra que es bastante estable, la ruta variable tendrá un impacto mayor en la construcción de los árboles de decisiones.

Esto es razonable, dado que se realiza el *split* en torno a diferentes características. Si hay una característica que varía mucho, como el RTT medido en la ruta cambiante, y que tiene gran impacto en la etiqueta asociada (la ruta cambiante con valores bajos de RTT será la clase elegida, con valores altos de RTT ya se sabe que se elegirá una ruta alternativa), entonces se le asignará un peso importante a esa característica. Además de darle mayor importancia a la ruta más variable, se la da a la ruta más rápida. En los escenarios elegidos suelen coincidir, dado que si la ruta más rápida no tuviera un cierto grado de varianza, no tendría demasiado sentido evaluar los algoritmos, que justamente deben adaptarse a encontrar la ruta más rápida. Finalmente, En la figura 5.19 se aprecia un ejemplo de la importancia de las características para el caso de Paris-Tokyo, y en la 5.20 se observa un ejemplo para el escenario Haifa-Santiago, que cuenta con 4 rutas.

En el escenario Paris-Tokyo tiene un notorio mayor peso la ruta directa. Esto se da pues durante el período de entrenamiento la ruta directa tiene básicamente dos niveles muy definidos: uno bajo en que siempre es la ruta más rápida y uno alto en que siempre es la ruta más lenta. En este caso el tiempo entre medidas casi no pesa, lo que tiene que ver también con la elección de un valor alto de T: un valor bajo de T implica que la información que aporta realizar medidas es sustancial, mientras que para un valor alto se está buscando reducir el costo de medida antes que explorar y obtener información.

Al analizar la importancia de las características del escenario Haifa-Santiago, vemos lo que cambia tener una ruta más variable que otra, aún teniendo retardo medio similar. En particular, para las muestras de entrenamiento consideradas,

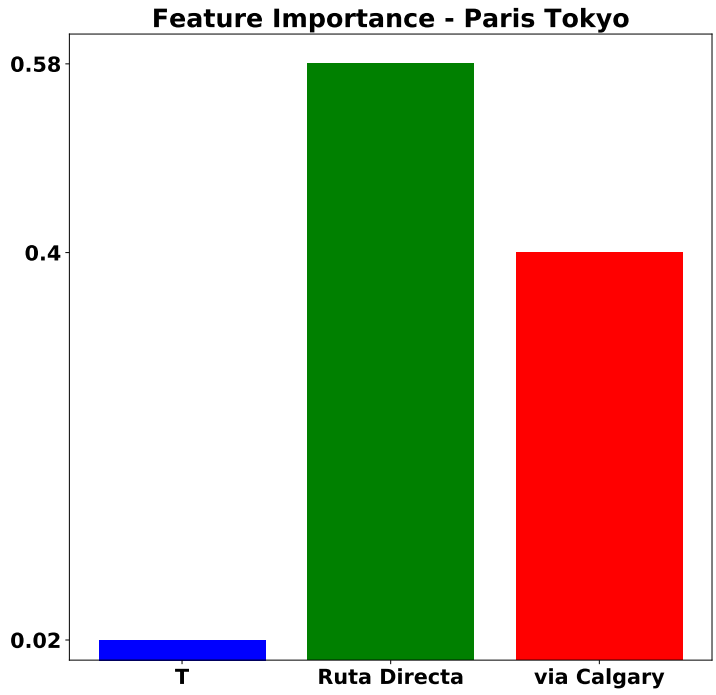


Figura 5.19: Análisis de la importancia de las características para el clasificador de *random forest* en el caso de la ruta Paris-Tokio. Se muestra el valor para el T seleccionado ($T = (13, 15)$), y habiendo realizado *5-fold cross validation*. Notar la importancia que tiene la ruta "por defecto", y el valor bajo que tiene el tiempo entre medidas.

los retardos medios ordenados valen (en ms): Ruta Directa – 296.8; via Dublin – 298.5; via Boston – 299.8; via Curitiba – 304.1. Sin embargo, Boston tiene mayor peso que Dublin, aún teniendo un retardo similar, por tener mayor variabilidad, y ser la ruta elegida durante buena parte del tiempo. Curitiba, al tener un retardo medio alto, es poco considerada, lo cual luego (en lo que resta de la serie) no se corresponderá con la realidad, dado que los retardos aumentan y Curitiba pasa a ser una opción interesante. Finalmente, la ruta directa no tiene muchas variaciones en el tramo de entrenamiento, y no es casi nunca la ruta elegida, pero posee un retardo medio muy bajo, por lo que mantiene un peso relativamente importante.

Comparación con la heurística alternativa

Como vimos en la sección de pruebas sintéticas, en [48] se proponen y comparan dos métodos para la resolución de nuestro problema de interés. Uno es el algoritmo de horizonte errante presentado, la heurística que se llamará "alternativa" fue desarrollada por Brun, Prahbu y Ségnéré. Al no tener acceso al código utilizado para repetir el experimento sobre otros escenarios, en esta Tesis se utilizaron los

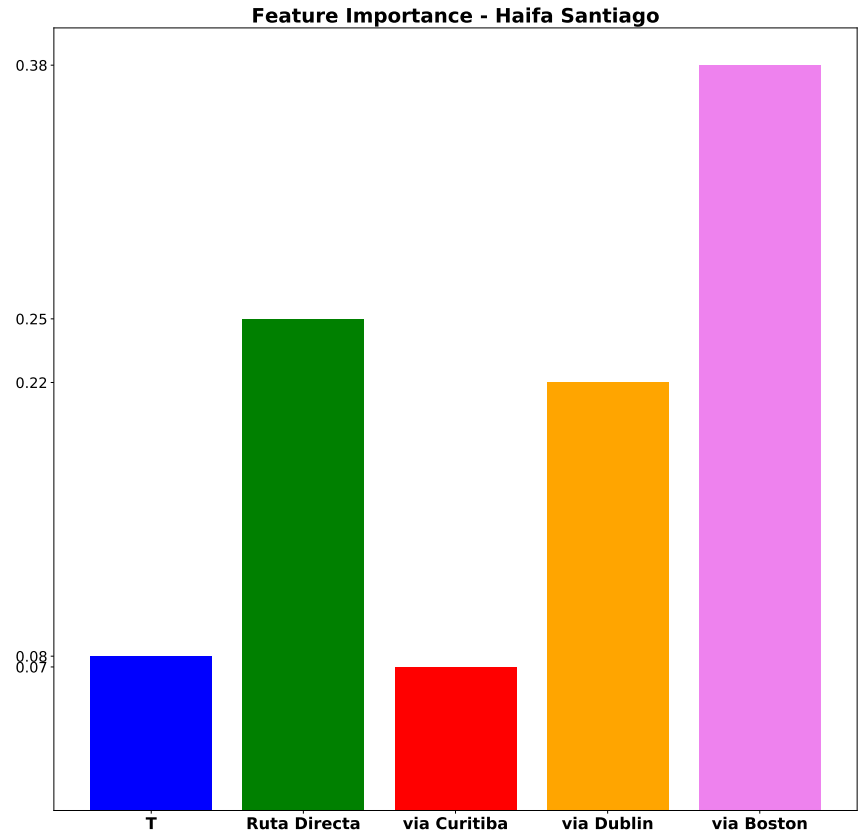


Figura 5.20: Análisis de la importancia de las características para el clasificador de *random forest* en el caso de la ruta Haifa-Santiago. Se muestra el valor para el T seleccionado ($T = 5$), y habiendo realizado *5-fold cross validation*. Notar como la importancia de las rutas se corresponde con cuales son más rápidas y más variables. El tiempo entre medidas, menor que para el caso de Paris-Tokyo, también asume mayor importancia.

resultados compartidos para la publicación del artículo, con lo que se obtiene la tabla 5.8. Estos resultados son si se quiere más interesantes que los presentados en la sección anterior para el algoritmo de horizonte errante, visto que se realizaron las pruebas sobre la traza completa.

Lamentablemente no se pudieron comparar los resultados de esta heurística alternativa directamente con el SALMON, dado que el SALMON exige un período de entrenamiento, mientras que los otros dos algoritmos no lo necesitan. En las figuras 5.21-5.22 se puede visualizar gráficamente las decisiones de ruteo para los casos de 4 rutas, en que vemos que la ruta elegida termina bordeando desde abajo al conjunto de rutas posibles (en casi toda época).

Par O-D	Número medio de medidas			% del tiempo mínimo retardo			diferencia con min retardo (%)		
	Alt	$H=3$	$H=1$	*Alt*	$H=3$	$H=1$	*Alt*	$H=3$	$H=1$
H-S	0.70	0.29	**0.12**	74	**87**	84	1.6	**0.2**	0.3
P-S	0.28	0.11	**0.05**	88	**97**	93	0.7	**0.1**	0.2
P-T	0.0	0.02	**0.002**	79	**88**	84	0.2	**0.1**	0.2
S-HK	1.06	0.56	**0.4**	**99**	98	98	**0.3**	0.8	0.8

Tabla 5.8: Resumen de los resultados logrados para los 4 pares OD considerados, donde Alt representa la heurística alternativa, $H=3$ el horizonte errante con horizonte 3 y $H=1$ representa la política miope.

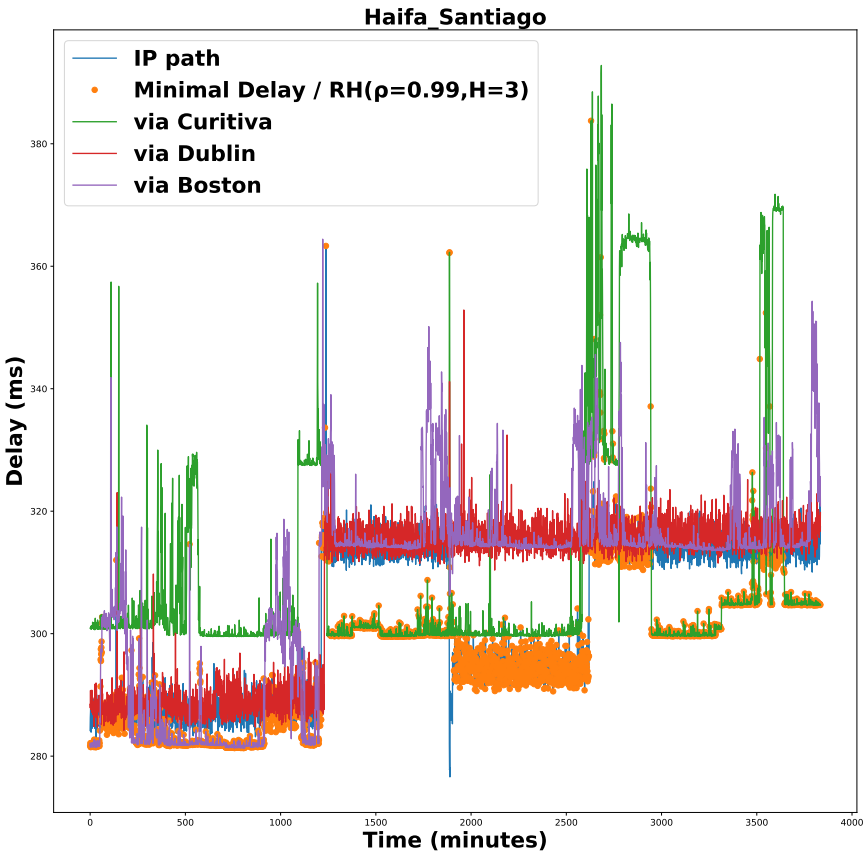


Figura 5.21: Demora de extremo a extremo entre Haifa y Santiago de Chile. En promedio, el horizonte errante monitorea solo $0{,}29$ rutas por paso de tiempo y, sin embargo, proporciona el retardo mínimo $87{,}4\,\%$ del tiempo. El retardo promedio de extremo a extremo es de $296{,}8$ ms con el horizonte errante, en lugar de 302 ms con la ruta IP directa.

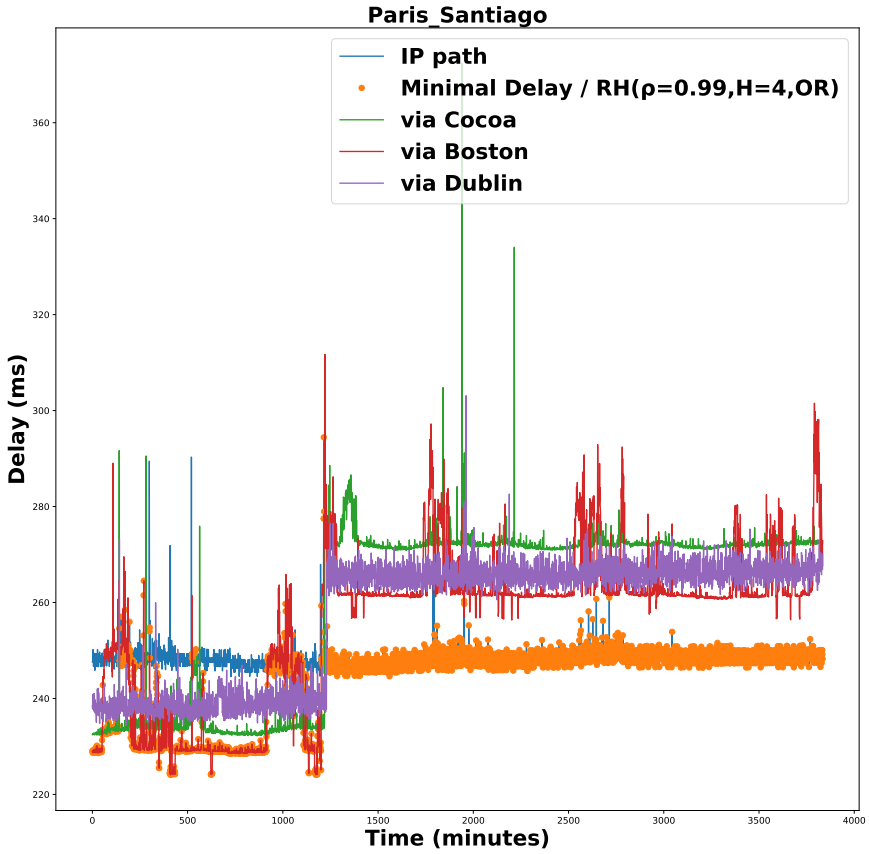


Figura 5.22: Retardo entre Paris y Santiago de Chile. En promedio, el horizonte errante monitorea sólo 0,07 rutas por época, obteniendo el retardo mínimo el 96,7 % del tiempo. El RTT medio baja de 248,4 ms con la ruta por defecto a 244,1 con el horizonte errante. Para esta imagen se utilizó un horizonte de $H = 4$ y la restricción de medir máximo una sola ruta por época de decisión.

En la figura 5.23 se aprecian las decisiones de ruteo obtenidas con la heurística alternativa, donde se observa que funciona bien, detectando los cambios de la ruta más baja. Vale la pena mencionar que en este escenario particular (Singapur - Hong Kong) los tres algoritmos obtienen muy buenos resultados, y que es en el escenario en que esta heurística mejor funciona. Como agregado, en su momento se probó a aumentar el horizonte a $H = 4$ y con esto se lograba alcanzar el 99 % del tiempo en la ruta ideal, es decir se empataba a la heurística alternativa. Esta es una virtud del algoritmo de horizonte errante: la posibilidad de seguir aumentando la profundidad; a pesar de que siempre existe el límite de escalabilidad.

Se observa que los resultados obtenidos con los algoritmos propuestos en esta

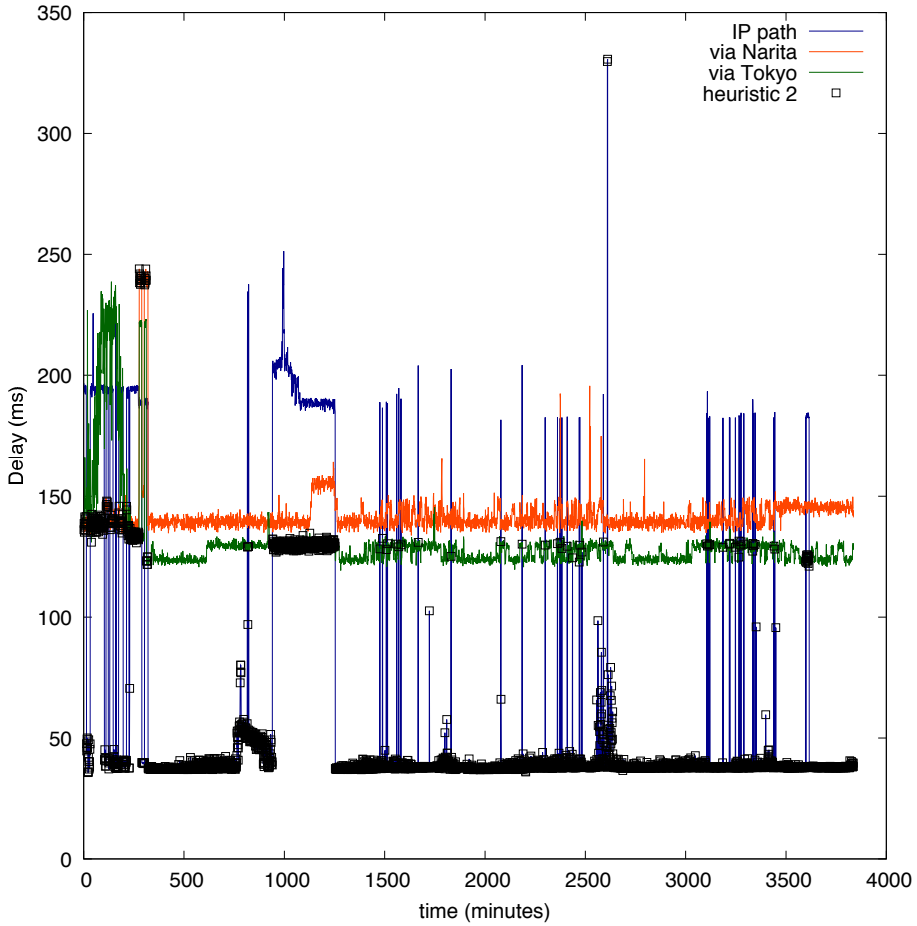


Figura 5.23: Retardo entre Singapur y Hong Kong. En promedio, la heurística alternativa monitorea 1,06 rutas por época, obteniendo el retardo mínimo el 99,2 % del tiempo.

Tesis son más alentadores que los de la heurística alternativa. Si bien en algunas rutas son similares, en varios casos la heurística alternativa no llega a igualar a la política miope, que en ningún caso es mejor que el horizonte de $H = 3$.

5.4. Análisis de resultados

Se realizarán a continuación algunos comentarios sobre los resultados obtenidos.

En primer lugar estos son alentadores en cuanto confirman el buen desempeño del algoritmo de horizonte errante en escenarios de la vida real. Es un algoritmo que realiza muy pocas medidas (como máximo medir una ruta cada cuatro minutos), que elije muy frecuentemente la mejor ruta (94.5 % – 99.1 %), y que funciona en línea lo cual es una virtud importante. Por otro lado, tiene problemas de escalabilidad con respecto a la cantidad de rutas involucradas. Esto podría mitigarse utilizando versiones con menor profundidad de horizonte, dado que alcanza mirar

unos pocos pasos hacia el futuro para que la política obtenida se acerque mucho a la política óptima. De hecho, los resultados obtenidos con la política miope son sorprendentemente buenos para algún escenario en que supera incluso un horizonte más profundo (Curitiba-Calgary).

Una observación interesante es que parece aumentar la cantidad de mediciones al aumentar la profundidad del horizonte. Esto no siempre se dió en los escenarios sintéticos, pero sí se da para todas las trazas. De alguna forma, contemplar los posibles estados y decisiones a futuro exige tomar mayor acción en el presente. Esta es una idea interesante en tanto midiendo más se logra una mejor política de ruteo; sin embargo sería importante acotar la cantidad máxima de medidas a realizar. No solamente buscar minimizar este valor, sino restringirlo a estar por debajo de un umbral, superado el cual podemos generar congestión, o deteriorar el estado de la red. Sin caer en problemas de budget fijo, uno podría buscar fomentar las medidas cuando estas son pocas, y restringirlas cuando son más frecuentes, por ejemplo utilizando pesos o funciones del costo en vez de mantener un costo fijo para todas las medidas.

Hay varios factores que pueden explicar algunas particularidades vistas en los escenarios. Por un lado hay mucha dependencia de lo que se puede ganar en disminución del RTT (que suele ser poco), hay casos en que quedan dudas de qué tan buenas son las aproximaciones markovianas para los comportamientos de las rutas. A su vez depende de qué tan cerca están las rutas paralelas entre sí (pudiendo los algoritmos errar en la decisión de ruteo sin perder mucho). El valor del costo incentivando o restringiendo las medidas también juega un rol importante, y finalmente el tramo elegido para test tendrá un impacto muy fuerte en los resultados obtenidos para las métricas elegidas (al comparar con el SALMON la reducción de muestras generó dificultades para encontrar tramos interesantes de observar).

Las cuantificaciones descritas han sido útiles para validar los algoritmos, que toman decisiones razonables para cambiar a las rutas más rápidas, a la vez que realizando una baja cantidad de medidas. Sin embargo, no permiten adelantar disminuciones del RTT posibles con los algoritmos: eso dependerá de las rutas en el momento en que se estén usando. Lo que si podemos asegurar es que los algoritmos mejorarán, y que el horizonte errante estará muy cerca de elegir casi-siempre la ruta correcta. Del SALMON es más difícil realizar esta afirmación.

Los resultados obtenidos consolidan la idea de incorporar herramientas de aprendizaje supervisado como solución al problema de ruteo, lo que nos libera de asumir modelos markovianos de los retardos. En este caso se utilizó un clasificador bien conocido y sencillo, que permitiera validar la dirección elegida. El algoritmo implementado alcanzó e incluso superó en un caso al de horizonte errante, lo que permite suponer que con otras herramientas se obtendrían resultados aún mejores. Se detectaron limitantes importantes para el entrenamiento: el algoritmo no logra muy buenos resultados con varias rutas. Esto es posiblemente por la necesidad de contar con mayor cantidad de muestras para el entrenamiento, que permitan capturar la complejidad del escenario considerado.

Utilizando cualquiera de los algoritmos propuestos se obtiene una disminución efectiva del tiempo de ida y vuelta. Una base de datos más nutrida permitiría

apreciar más significativamente el beneficio de utilizar los algoritmos, pero ya se vió en la Introducción en estudios concretamente enfocados al estudio de BGP y las rutas paralelas que se pueden obtener disminuciones en los tiempos de hasta el 50 % en un 25 % de los caminos [22]. Es decir: la ventaja existe, es cuestión de encontrar esas rutas y de utilizar los algoritmos propuestos para aprovecharlo.

Una última comparación importante es frente a la política de "siempre medir", que es la que se aplica en [11], [29], y tantos otros trabajos. En los algoritmos propuestos se está realmente muy cerca de elegir la mejor ruta en todo momento pero a su vez realizando muy pocas medidas. La restricción de medir todas las rutas con mismo período y en simultáneo no ofrece mucho aire para la disminución del costo en los algoritmos de aprendizaje supervisado, en ese sentido el algoritmo de horizonte errante tiene un ajuste más suave de la cantidad de medidas al costo de medida.

Capítulo 6

Conclusiones y trabajo a futuro.

6.1. Conclusiones

En esta Tesis se busca resolver el problema de elegir la ruta más rápida en una red sobrepuesta minimizando el costo de medida. La selección inteligente de rutas es un problema con vigencia. Esto sucede porque el ruteo entre sistemas autónomos utilizando BGP, que se ha instalado como protocolo por defecto, puede ser subóptimo y además propagar fallas o demorar en la recuperación ante las mismas.

Al formular este problema como un problema de optimización, se desea minimizar la esperanza de la suma del tiempo de ida y vuelta más el costo de las medidas realizadas. Es decir: minimizar la cantidad de medidas realizadas y el tiempo de ida y vuelta de la ruta elegida, en cada época de decisión (cada algunos minutos).

En un primer acercamiento basado en trabajos previos [41], se utilizan cadenas de markov para modelar el retardo en las rutas, lo que permite su formulación como problema de decisión markoviano (MDP), y la aplicación de técnicas de programación dinámica bien conocidas para encontrar la política óptima de medición y ruteo.

Se emplea para esto el algoritmo de *modified policy iteration*, principalmente como base comparativa para los algoritmos de elaboración propia que fueron introducidos en esta Tesis. La solución que ofrece el *policy iteration* es óptima en escenarios markovianos. Sin embargo, estas técnicas no son computacionalmente factibles para problemas de más de dos rutas, lo que impide su utilización en la práctica. Se proponen entonces dos algoritmos que permiten la resolución del problema de manera escalable y con buenos resultados.

El primero se monta sobre el enfoque del problema como un MDP, utilizando una técnica de ventana deslizante proveniente de la teoría del control predictivo por modelo. Esta es una combinación novedosa, que permite la utilización de las ecuaciones de Bellman con complejidad limitada, dado que se utiliza un horizonte fijo en una cantidad de pasos.

Este algoritmo, conocido como *receding horizon* o de horizonte errante, permi-

te obtener soluciones casi-óptimas con un costo computacional moderado. Tiene como ventaja que corre en línea, a diferencia de los algoritmos de programación dinámica. Esta solución es más escalable, aunque no deja de caer en el *curse of dimensionality*: el incremento en la cantidad de rutas es exponencial en la cantidad de estados a visitar.

Por otro lado, cada vez es más cuestionada la asunción de modelos, y se han multiplicado técnicas de aprendizaje supervisado que permiten escapar de hipótesis tan fuertes. Se realiza un modelado del problema para la aplicación de técnicas de aprendizaje supervisado para clasificación. El objetivo es clasificar el estado de la red con el fin de elegir la ruta más rápida en todo momento. Para esto, se establecen dos restricciones: todas las rutas serán medidas en simultáneo, y cada cierto período T. El estado del sistema consiste en el conjunto del RTT medido para cada ruta, y el tiempo transcurrido desde esa medición.

Se utiliza una técnica bien conocida llamada *random forest* para la elección de la ruta más rápida, Se formula un problema de optimización para estimar el valor T^* que ofrece el mínimo valor esperado del "costo de medida más tiempo de ida y vuelta", utilizando validación cruzada.

Ambos algoritmos fueron validados y comparados en escenarios sintéticos, Las trazas son generadas a partir del estudio de rutas de [33], y entre otras pruebas se usan ejemplos de la literatura ([48], [41]). Los resultados obtenidos son alentadores: el algoritmo de horizonte errante con paso $H = 1$, también llamado miope, obtiene ya muy buenos resultados en varias ocasiones. Sumado a esto, sólo incrementando unos pocos pasos la profundidad del horizonte se obtienen resultados casi-óptimos. La heurística de aprendizaje supervisado propuesta demostró ser muy cercana al algoritmo de horizonte errante, con más escalabilidad y ejecutándose en menor tiempo, sobre algunas trazas reales obteniendo incluso mejores resultados. Esto permite pensar que ante escenarios cuya aproximación markoviana no es buena, otras técnicas de aprendizaje libres de estas asunciones pueden tomar protagonismo.

También se comparó al algoritmo de horizonte errante y al clasificador de *random forest* contra otro algoritmo que busca resolver el mismo problema, utilizando trazas reales obtenidas de *NLNOG ring*, y con resultados muy alentadores. Esta heurística fue presentada en [48] y propuesta por O. Brun, M. Ségnéré y B. Prahbu, de CNRS–LAAAS Toulouse. Además, ambas heurísticas han funcionado correctamente en escenarios no-markovianos reales y sintéticos. Por último, se propusieron una serie de modificaciones a los algoritmos para su mejor funcionamiento.

6.2. Del trabajo pasado al trabajo a futuro

En un trabajo de horizonte finito, como una tesis de Maestría, quedan muchas ideas por el camino, máxime siendo que uno está más preparado para enfrentar el problema planteado al cierre del trabajo. En las próximas líneas se resumen algunas de ellas. En primer lugar, se comentan esfuerzos que no prosperaron, pero que pueden revisarse a la luz de los buenos resultados obtenidos con los algoritmos propuestos y pensando en combinaciones de estos métodos.

Utilizando el enfoque ya conocido del problema como MDP, se puede buscar la utilización de técnicas de aprendizaje por refuerzo. Esto se probó aunque sin suerte, dado que el sistema markoviano hace muy difícil la exploración para algunos estados: aquellos que se logran visitar únicamente gracias a la concatenación sucesivas de la acción "no-medir". Sin embargo, se puede pensar en utilizar métodos de aproximación para las zonas de difícil recorrido. Hubo cierto esfuerzo en este sentido, que finalmente no prosperó. Una idea es utilizar la aproximación del algoritmo de horizonte errante para entrenar un algoritmo de aprendizaje profundo por refuerzo que aprenda a partir del retorno real obtenido en la exploración con el retorno aproximado.

Otro enfoque estudiado es la utilización de teoría de juegos para tomar la mejor decisión de ruteo. Sin embargo esta formulación exige un mínimo modelado estadístico de los retardos de las rutas, lo que no nos libera demasiado de la suposición markoviana. Si bien se obtuvieron resultados interesantes, por partir de modelados markovianos y ser peores que los obtenidos con el horizonte errante, no se profundizó en esta línea de trabajo. Una posibilidad que permite escalar el algoritmo de horizonte errante es considerar solamente las K mejores rutas en cada instante, en vez de todas las rutas, para su aplicación. Esta restricción garantiza la factibilidad del problema, y K puede hallarse en función de la profundidad de horizonte deseada; ser fijado en un valor determinado; o a través de recorrer valores combinando H y K, imponiendo restricciones de tiempos y buscando minimizar la función de valor $V^{KH}(s)$, $\forall s \in \mathcal{S}$.

Existe gran variedad de técnicas de aprendizaje supervisado, y en este trabajo se probó la validez de utilizar clasificadores escalables, sencillos y robustos como *CART*. Se puede pensar en utilizar técnicas más complejas de aprendizaje automático, como son técnicas de aprendizaje profundo: se piensa en redes neuronales recurrentes, dado que se considera que sería importante estudiar la correlación temporal de los datos. En [56] se proponen varias técnicas que se pueden aplicar para estimar la ruta más rápida. Un aspecto importante a tener en cuenta en esta línea es que estos algoritmos suelen exigir grandes cantidades de datos, algo de lo que no se disponía a partir de las trazas estudiadas.

Por otro lado, sería importante relajar la restricción de usar un período fijo de medida simultánea obteniendo una solución escalable y de adaptación dinámica al estado del sistema. Una propuesta es explorar el aprendizaje del valor de un T fijo para cada ruta, o directamente los valores que T debe tomar en cada momento. Una primera forma en que se puede avanzar es en por lo menos encontrar valores alrededores de los cuales buscar T para cada ruta, de alguna forma vinculados a la estabilidad-variabilidad de los retardos, y poder mejorar aún más los resultados obtenidos. Otra falencia que tiene la formulación realizada es la medición simultánea de las rutas. Esto podría pensarse de otra forma: aún manteniendo un mismo período de medida por ruta, pero distribuyendo las medidas sobre diferentes épocas, evitando la realización de múltiples medidas en una misma época.

Es con alegría que se resume: fueron presentadas y es posible seguir desarrollando soluciones inteligentes y casi-óptimas para los problemas de ruteo y minimización del costo de medición en redes sobrepuestas.

Esta página ha sido intencionalmente dejada en blanco.

Apéndice A

Solución exacta del problema miope

Esta sección fue realizada como parte de trabajo final de Teoría y Algoritmia de Optimización[1].

A.1. Generalización al problema con N rutas, con K posibles retardos

Ampliaremos ahora el problema a uno con N rutas, cada ruta con K posibles retardos. Lo plantearemos de la siguiente forma:

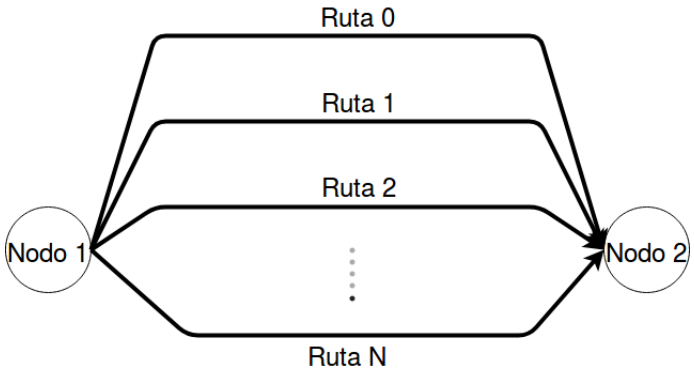


Figura A.1: Problema general: N rutas con hasta K posibles retardos (estados) cada una.

Cada ruta r_i tiene K posibles retardos, costo de medición c_i y acciones posibles $M_i \in \{0, 1\}$. La variable $a_{ij}(t)$ representa el caso en que mido en ciertas i rutas y no mido en las restantes j rutas. Se puede ver como un vector de largo 2^N cuyos valores son 1 en las rutas a medir y 0 sino.

Definimos al conjunto $\mathcal{C}_{a_{ij}}(t) = \{\overline{L_i}/M_i = 0\} \bigcup \{l_j^k/M_i = 1\}$, donde l_j^k es el nivel más bajo en las j rutas medidas: $l_j^k < l_j^i \forall i \neq k$.

[1] https://eva.fing.edu.uy/course/view.php?id=963

De entre todos los retardos de las j rutas medidas y retardo medios de las i rutas estimadas, hay un valor que es el menor. Llamaremos l^* a este valor, y observamos que $l^*(t) = \text{mín}_{a_{ij}} \mathcal{C}_{a_{ij}}(t)$.
Al mismo tiempo, este valor l^* tiene probabilidad de ocurrencia dada por el $x = P(l^*)$ si la ruta es medida, y tiene probabilidad 1 si l^* es un retardo medio, por lo que a l^* asociamos la probabilidad P^*:

$$l^* \to P^* = \begin{cases} 1 & \text{si } M = 0 \\ P(l^*) & \text{si } M = 1 \end{cases}$$

De igual manera, llamamos a l^{2*} la segunda ruta de menor costo (ya sea medido o medio); y definimos a P^{2*} como la probabilidad de ocurrencia de dicho retardo como ya se hizo para P^*. Continuamos así hasta la N-ésima ruta. Obsérvese que podemos escribir a l^{2*} como $l^{2*} = \text{mín}\{\mathcal{C} - l^*\}$.

Ahora, para cada combinación posible de 'medir o no-medir' a_{ij} consideramos $D_{a_{ij}}$ como el valor mínimo esperado del retardo para el conjunto de las rutas dado que medimos o no según el vector a_{ij}. No estamos considerando el costo de medir, sino simplemente el retardo.

En cada instante t, tendremos dos casos:

- O bien la ruta de menor retardo es medida y tiene probabilidad de ocurrencia $P^* \neq 1$, y entonces tengo que elegir con probabilidad $(1 - P^*)$ la segunda ruta de menor retardo
- O bien la ruta de menor retardo es un valor medio l^* y tiene probabilidad $P^* = 1$.

En el segundo caso el problema se simplifica bastante, ya que no tengo que preocuparme por la segunda ruta de menor retardo. En el primer caso, luego de preocuparme por la segunda ruta de menor retardo, debo realizar el mismo razonamiento con la ruta de tercer menor retardo, y así sucesivamente.

Planteamos entonces $D_{a_{ij}}(t)$ como la esperanza de tener el menor retardo posible para una combinación a_{ij} de medidas y no medidas. Es decir:

$$D_{a_{ij}}(t) = P^* l^* + (1 - P^*)[P^{2*} l^{2*} + (1 - P^{2*})[P^{3*} l^{3*} + [(1 - P^{3*})[P^{4*} l^{4*} + [\cdots]]]]]$$

Donde incorporamos las n probabilidades de retardos más bajos de las posibles rutas. Podemos reescribir esto como:

$$D_{a_{ij}} = \sum_{k=1}^{N} d_k p_k \prod_{j=1}^{k-1} (1 - p_j) + d_N \prod_{i=1}^{N-1} (1 - p_i)$$

Luego podemos plantear el problema general y relajado para N caminos con K rutas posibles:

$$a^* = \underset{a_{ij}}{\text{argmín}}\, E\{J(t)|a_{ij}\} = \underset{a_{ij}}{\text{argmín}}\, E\{D_{a_{ij}} + C_{a_{ij}}\}$$

$$\text{sujeto a, } \forall a_{ij} : \sum a_{ij} = 1$$

$$a_{ij}[k] \in [0,1] \forall k \in [0, 2^N]$$

Obtenemos un problema lineal sobre nuestra variable a^*_{ij}, que además tiene como espacio factible un simplex. Teniendo un problema lineal sobre un poliedro (o un simplex), podemos ver que las soluciones exactas al problema se encuentran en el conjunto factible, y más precisamente sobre las aristas del problema, es decir con valores de los diferentes $a_{ij} \in \{0, 1\}$. Esto es porque estamos dentro de las hipótesis del Teorema Fundamental de Programación Lineal [2].

A.2. Resolución del problema generalizado

Podemos reescribir el problema de manera más sencilla:

$$A^* = \underset{A}{\text{argmín}}\, A^T(D + C)$$

$$\text{sujeto a, } \sum a[i] = 1$$

$$a[i] \in [0,1] \forall i$$

Donde A, D y C son vectores de 2^N términos. Consideremos que los vectores D y C están ordenados para corresponder sus términos a la combinatoria de $a[i]$ respectiva. Es decir, el valor de $C[0] = 0$, ya que sería no medir en ninguna ruta, y el valor de $D[0]$ sería el mínimo de los valores medios de las rutas. En este caso, al hallar el valor óptimo de A^*, tendremos un vector con 0 en todos sus valores excepto en uno, que será la combinación óptima.

Si bien una solución posible del problema lineal puede ser simplemente tomar el mínimo valor del vector $D + C$, esta es una solución discreta, que no contempla la convexidad del problema relajado. Esta solución es más sencilla de visualizar, si se quiere, pero se aleja del contenido de la asignatura, por lo que decidimos realizar un descenso por el gradiente, proyectándolo cuando necesario (Projected Gradient Descent). Incluso podría resultar que la forma más rápida de encontrar el resultado no sea recorriendo un vector (que puede ser grande), sino iterando de alguna forma más astuta (por ejemplo con el método del punto interior).

[2] https://en.wikipedia.org/wiki/Fundamental_theorem_of_linear_programming

Esta página ha sido intencionalmente dejada en blanco.

Apéndice B

Una propuesta de mejora para el algoritmo de Horizonte Errante

Como se discutió anteriormente, el número de estados del sistema aumenta exponencialmente con el número de rutas N. Sin embargo, si se restringe la cantidad de rutas que se puede medir en cada paso a una ruta como máximo (teniendo en cuenta solo un subconjunto del espacio de acciones), entonces la cantidad de estados se reduce considerablemente.

Para ver esto, considere un estado inicial $s = (s^i)_{i=1...N}$. El número de diferentes estados de información en el nivel H viene dado por los diferentes valores que puede tomar cada componente de dicho vector (es decir, s_i). Como se vió, los valores que s^i puede tomar son (L_{Last}, τ^i), donde L_{Last} es cualquiera de los K^i niveles de la ruta i y τ^i es hace cuanto se midió esa ruta. Para simplificar, asumamos que cada ruta tiene la misma cantidad de niveles.

Para analizar los estados posibles H pasos después se discriminará según cuantas rutas se midieron. Recordemos que el estado de una ruta en una época de decisión está completamente caracterizado por un nivel medido en algún momento de esa ruta y el tiempo que pasó desde esa medida. Veamos entonces cuales son los posibles estados finales que puede tomar nuestro problema: esa es la cantidad final de estados en el árbol al aplicar el algoritmo con horizonte H. Si después de los H pasos del algoritmo (es decir en H épocas) se miden H rutas:

- cuantos posibles estados surgen de que H rutas son medidas entre las N posibles se puede ver como la combinatoria de H en N sin repetición: $\binom{N}{H}$.
- falta incluir los posibles ordenamientos de esas H medidas en las H épocas. Esto es la permutación de H en H: $P(H, H)$, de cuantas formas puedo ordenar y ubicar esas H medidas en H épocas.
- Cada ruta medida puede estar en alguno de los K niveles posibles, esto es K^H posibilidades para las H rutas.
- Las restantes N-H rutas no aportan posibles ramificaciones, dado que al no ser medidas evolucionarán solamente incrementando su valor de $\tau^i \to \tau^i + H$.

Si se mide H-1 rutas:

- Se debe seleccionar H-1 elementos de los N posibles: nuevamente obtenemos la combinatoria de H-1 en N sin repetición: $\binom{N}{H-1}$.
- Hay que ubicar las H-1 medidas entre las H épocas posibles, pudiendo reordenarse H-1: la permutación de H-1 en H, $P(H, H-1)$.
- Cada ruta medida aporta los posibles K^{H-1} niveles.
- Las restantes rutas evolucionan aumentando su *lag* de manera fija a $\tau^i + H$.

Así continuamos, hasta que se mide 1 ruta:

- Se debe seleccionar 1 elemento de los N posibles: nuevamente obtenemos la combinatoria de 1 en N sin repetición.
- Hay que ubicar esta medida entre las H épocas posibles.
- La ruta medida aporta los posibles K niveles.
- Las restantes rutas evolucionan aumentando su *lag* a $\tau^i + H$.

Cuando se mide una ruta, sabemos que hay solamente K estados posibles. Cuando no se mide ninguna ruta, hay 1 sólo estado posible. Si llamamos h a la cantidad de rutas medidas, podemos trasladar este razonamiento a su forma matemática en la ecuación (B.1)

$$\sum_{h=0}^{H} P(H,h)\binom{N}{h}K^h = \sum_{h=0}^{H} \frac{H!}{(H-h)!}\frac{N!}{(N-h)!h!}K^h, \tag{B.1}$$

donde $P(H,h) = \frac{H!}{(H-h)!}$ representa el número de $h-$permutaciones de H.

La intuición detrás de (3.4) es la siguiente. Desde la raíz hasta el nivel H, considere h diferentes rutas que se miden, esto lleva a K^h diferentes estados en el nivel H. Además, estas h rutas pueden ser cualquiera de las N rutas disponibles, tenemos por lo tanto, combinaciones de h rutas a elegir en N. Además, el orden en que se miden las rutas proporciona diferentes estados alcanzables. Tenemos h-permutaciones de H épocas de medida. Finalmente, h toma todos los valores de 0 a H. Tenga en cuenta que si $H > N$, (3.4) sigue siendo válida ya que los términos desde $N + 1$ en adelante son iguales a 0.

Esta versión del algoritmo se implementó. En la figura 3.5 se aprecia el tiempo de medida para el ejemplo visto, en el caso de utilizar el algoritmo en su versión original y con la restricción. Sin embargo, esta restricción nos está privando de visitar estados y de considerar acciones.

¿Cuál es entonces el razonamiento detrás de esta modificación? Que la reducción en el espacio de estados permite escalabilidad, buscando un impacto leve en el algoritmo: si necesitamos medir dos rutas, alcanza con que se mida una ruta en una época y la otra en la siguiente. Sin embargo, a pesar de que sí se logra disminuir el tiempo de procesamiento de manera considerable, esto nos aleja de

la política óptima. A su vez, disminuir el espacio de acciones y de estados de esta forma no logra romper con el crecimiento exponencial sobre la cantidad de caminos. Es decir: funciona más rápido, pero en escenarios en que el algoritmo original también era aplicable, y obteniendo peores resultados.

Otra idea que se implementó fue realizar *pruning* para aquellas ramas del árbol en que la probabilidad de visita fuera muy baja. *Pruning* es evitar considerar parte del espacio de estados y acciones que no aportan información relevante, disminuyendo así la complejidad del modelo. Al estar el retorno asociado a visitar esos estados multiplicado por esa probabilidad de visita, esto permitiría bajar la cantidad de ramificaciones pero sin alterar la función de valor estimada al tomar ciertas acciones. Esto no dió muy buenos resultados, aunque si permitió podar el árbol y acelerar la ejecución del algoritmo. El problema es que el árbol se simplificaba en exceso, y entonces se perdía información valiosa para dirimir entre diferentes acciones cuyas funciones de valor estuvieran relativamente cerca, lo que nuevamente nos alejaba de la política óptima.

Esta página ha sido intencionalmente dejada en blanco.

Referencias

[1] V. Paxson, "End-to-end routing behavior in the internet," *IEEE/ACM transactions on Networking*, vol. 5, no. 5, pp. 601–615, 1997.

[2] F. Jahanian, C. Labovitz, and A. Ahuja, "Experimental study of internet stability and wide-area backbone failures," *U. of Michigan, Tech. Rep. CSE-TR-382-98*, 1998.

[3] B. Chandra, M. Dahlin, L. Gao, and A. Nayate, "End-to-end wan service availability.," in *USITS*, pp. 97–108, Citeseer, 2001.

[4] S. Savage, A. Collins, E. Hoffman, J. Snell, and T. Anderson, "The end-to-end effects of internet path selection," in *Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication*, pp. 289–299, 1999.

[5] C. Labovitz, G. R. Malan, and F. Jahanian, "Internet routing instability," *IEEE/ACM transactions on Networking*, vol. 6, no. 5, pp. 515–528, 1998.

[6] C. Labovitz, A. Ahuja, A. Bose, and F. Jahanian, "Delayed internet routing convergence," *ACM SIGCOMM Computer Communication Review*, vol. 30, no. 4, pp. 175–187, 2000.

[7] A. Feldmann, O. Maennel, Z. M. Mao, A. Berger, and B. Maggs, "Locating internet routing instabilities," *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 4, pp. 205–218, 2004.

[8] H. Pucha, Y. Zhang, Z. M. Mao, and Y. C. Hu, "Understanding network delay changes caused by routing events," *ACM SIGMETRICS performance evaluation review*, vol. 35, no. 1, pp. 73–84, 2007.

[9] Y. Schwartz, Y. Shavitt, and U. Weinsberg, "A measurement study of the origins of end-to-end delay variations," in *International Conference on Passive and Active Network Measurement*, pp. 21–30, Springer, 2010.

[10] M. Rimondini, C. Squarcella, and G. Di Battista, "From bgp to rtt and beyond: Matching bgp routing changes and network delay variations with an eye on traceroute paths," *arXiv preprint arXiv:1309.0632*, 2013.

[11] D. Andersen, H. Balakrishnan, F. Kaashoek, and R. Morris, "Resilient overlay networks," in *Proceedings of the eighteenth ACM symposium on Operating systems principles*, pp. 131–145, 2001.

[12] A. Collins, *The Detour framework for packet rerouting.* PhD thesis, Master's thesis, University of Washington, 1998.

[13] S. Savage, T. Anderson, A. Aggarwal, D. Becker, N. Cardwell, A. Collins, E. Hoffman, J. Snell, A. Vahdat, G. Voelker, *et al.*, "Detour: Informed internet routing and transport," *Ieee Micro*, vol. 19, no. 1, pp. 50–59, 1999.

[14] Y. Chen, D. Bindel, H. Song, and R. H. Katz, "An algebraic approach to practical and scalable overlay network monitoring," in *Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, pp. 55–66, 2004.

[15] T. Anderson, L. Peterson, S. Shenker, and J. Turner, "Overcoming the internet impasse through virtualization," *Computer*, vol. 38, no. 4, pp. 34–41, 2005.

[16] J. Touch, Y.-S. Wang, L. Eggert, and G. Finn, "A virtual internet architecture," *ISI Technical Report ISI-TR-2003-570*, pp. 73–80, 2003.

[17] N. Feamster, H. Balakrishnan, J. Rexford, A. Shaikh, and J. Van Der Merwe, "The case for separating routing from routers," in *Proceedings of the ACM SIGCOMM workshop on Future directions in network architecture*, pp. 5–12, 2004.

[18] M. Beck, T. Moore, and J. S. Plank, "An end-to-end approach to globally scalable programmable networking," in *Proceedings of the ACM SIGCOMM workshop on Future directions in network architecture*, pp. 328–339, 2003.

[19] L. Qiu, Y. R. Yang, Y. Zhang, and S. Shenker, "On selfish routing in internet-like environments," in *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pp. 151–162, 2003.

[20] J. Han and F. Jahanian, "Impact of path diversity on multi-homed and overlay networks," in *International Conference on Dependable Systems and Networks, 2004*, pp. 29–38, IEEE, 2004.

[21] P. K. Gummadi, H. V. Madhyastha, S. D. Gribble, H. M. Levy, D. Wetherall, *et al.*, "Improving the reliability of internet paths with one-hop source routing.," in *OSDI*, vol. 4, pp. 13–13, 2004.

[22] H. Rahul, M. Kasbekar, R. Sitaraman, and A. Berger, "Towards realizing the performance and availability benefits of a global overlay network," 2005.

[23] P. Belzarena, G. G. Sena, I. Amigo, and S. Vaton, "Sdn-based overlay networks for qos-aware routing," in *Proceedings of the 2016 workshop on Fostering Latin-American Research in Data Communication Networks*, pp. 19–21, 2016.

[24] V. Kotronis, X. Dimitropoulos, and B. Ager, "Outsourcing the routing control logic: better internet routing based on sdn principles," in *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*, pp. 55–60, 2012.

[25] P. R. Torres-Jr, A. García-Martínez, M. Bagnulo, and E. P. Ribeiro, "Bartolomeu: An sdn rebalancing system across multiple interdomain paths," *Computer Networks*, vol. 169, p. 107117, 2020.

[26] R. W. Thomas, D. H. Friend, L. A. DaSilva, and A. B. MacKenzie, "Cognitive networks," in *Cognitive radio, software defined radio, and adaptive wireless systems*, pp. 17–41, Springer, 2007.

[27] E. Gelenbe and Z. Kazhmaganbetova, "Cognitive packet network for bilateral asymmetric connections," *IEEE Transactions on Industrial Informatics*, vol. 10, no. 3, pp. 1717–1725, 2014.

[28] O. Brun, H. Hassan, and J. Vallet, "Scalable, self-healing, and self-optimizing routing overlays," in *2016 IFIP Networking Conference (IFIP Networking) and Workshops*, pp. 64–72, IEEE, 2016.

[29] O. Brun, L. Wang, and E. Gelenbe, "Big data for autonomic intercontinental overlays," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 3, pp. 575–583, 2016.

[30] P. Belzarena and L. Aspirot, "End-to-end quality of service seen by applications: A statistical learning approach," *Computer Networks*, vol. 54, no. 17, pp. 3123–3143, 2010.

[31] C. Thraves-Caro, J. Doncel, and O. Brun, "Optimal path discovery problem with homogeneous knowledge," 2015.

[32] A. Nakao, L. Peterson, and A. Bavier, "Scalable routing overlay networks," *ACM SIGOPS Operating Systems Review*, vol. 40, no. 1, pp. 49–61, 2006.

[33] "RIPE Atlas." `https://atlas.ripe.net/`. Accessed: 2019-11-26.

[34] S. Belhaj and M. Tagina, "Modeling and prediction of the internet end-to-end delay using recurrent neural networks.," *J. Networks*, vol. 4, no. 6, pp. 528–535, 2009.

[35] A. Coates, A. O. Hero III, R. Nowak, and B. Yu, "Internet tomography," *IEEE Signal processing magazine*, vol. 19, no. 3, pp. 47–65, 2002.

[36] Y. Vardi, "Network tomography: Estimating source-destination traffic intensities from link data," *Journal of the American Statistical Association*, vol. 91, no. 433, pp. 365–377, 1996.

[37] N. Etemadi Rad, Y. Ephraim, and B. L. Mark, "Delay network tomography using a partially observable bivariate markov chain," *IEEE/ACM Transactions on Networking*, vol. 25, no. 1, pp. 126–138, 2017.

[38] J. D. Horton and A. López-Ortiz, "On the number of distributed measurement points for network tomography," in *Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement*, pp. 204–209, 2003.

[39] C. Szepesvári, "Shortest path discovery problems: A framework, algorithms and experimental results," in *AAAI*, pp. 550–555, 2004.

[40] M. Mouchet, S. Vaton, and T. Chonavel, "Statistical characterization of round-trip times with nonparametric hidden markov models," in *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pp. 43–48, IEEE, 2019.

[41] S. Vaton, O. Brun, M. Mouchet, P. Belzarena, I. Amigo, B. J. Prabhu, and T. Chonavel, "Joint minimization of monitoring cost and delay in overlay networks: optimal policies with a markovian approach," *Journal of Network and Systems Management*, vol. 27, no. 1, pp. 188–232, 2019.

[42] Y. Teh, M. Jordan, M. Beal, and D. Blei, "Hierarchical dirichlet processes (technical report 653)," *UC Berkeley Statistics*, 2004.

[43] Y. W. Teh, M. I. Jordan, M. J. Beal, and D. M. Blei, "Sharing clusters among related groups: Hierarchical dirichlet processes," in *Advances in neural information processing systems*, pp. 1385–1392, 2005.

[44] M. J. Beal, Z. Ghahramani, and C. E. Rasmussen, "The infinite hidden markov model," in *Advances in neural information processing systems*, pp. 577–584, 2002.

[45] M. Larrañaga, M. Assaad, A. Destounis, and G. S. Paschos, "Asymptotically optimal pilot allocation over markovian fading channels," *IEEE Transactions on Information Theory*, vol. 64, no. 7, pp. 5395–5418, 2017.

[46] S. Liu, M. Fardad, E. Masazade, and P. K. Varshney, "Optimal periodic sensor scheduling in networks of dynamical systems," *IEEE Transactions on Signal Processing*, vol. 62, no. 12, pp. 3055–3068, 2014.

[47] R. S. Sutton and A. G. Barto, "Reinforcement learning: An introduction," 2011.

[48] M. Mouchet, M. Randall, M. Ségneré, I. Amigo, P. Belzarena, O. Brun, B. Prabhu, and S. Vaton, "Scalable monitoring heuristics for improving network latency," in *NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium*, pp. 1–9, IEEE, 2020.

[49] G. P. Basharin, A. N. Langville, and V. A. Naumov, "The life and work of a. a. markov."

[50] C. M. Grinstead and J. L. Snell, *Introduction to probability*. American Mathematical Soc., 2012.

[51] M. L. Puterman, *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[52] M. L. Puterman and M. C. Shin, "Modified policy iteration algorithms for discounted markov decision problems," *Management Science*, vol. 24, no. 11, pp. 1127–1137, 1978.

[53] U. Meister and U. Holzbaur, "A polynomial time bound for howard's policy improvement algorithm," *Operations-Research-Spektrum*, vol. 8, no. 1, pp. 37–40, 1986.

[54] G. C. Goodwin, M. Seron, J. De Dona, *et al.*, "Constrained control and estimation: and optimization approach/graham c. goodwin, maría m. seron, and josé de doná.," 2005.

[55] G. Van Rossum and F. L. Drake Jr, *Python tutorial*. Centrum voor Wiskunde en Informatica Amsterdam, The Netherlands, 1995.

[56] S. Wassermann, P. Casas, T. Cuvelier, and B. Donnet, "Netperftrace: Predicting internet path dynamics and performance with machine learning," in *Proceedings of the Workshop on Big Data Analytics and Machine Learning for Data Communication Networks*, pp. 31–36, 2017.

[57] P. Casas, "Two decades of ai4nets-ai/ml for data networks: Challenges & research directions," in *NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium*, pp. 1–6, IEEE, 2020.

[58] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[59] L. Breiman *et al.*, "Statistical modeling: The two cultures (with comments and a rejoinder by the author)," *Statistical science*, vol. 16, no. 3, pp. 199–231, 2001.

[60] T. Hastie, R. Tibshirani, and J. Friedman, *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media, 2009.

[61] L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen, *Classification and regression trees*. CRC press, 1984.

[62] W.-Y. Loh, "Fifty years of classification and regression trees," *International Statistical Review*, vol. 82, no. 3, pp. 329–348, 2014.

[63] L. Breiman, "Bias, variance, and arcing classifiers," tech. rep., Tech. Rep. 460, Statistics Department, University of California, Berkeley . . . , 1996.

[64] L. Breiman, "Bagging predictors," *Machine learning*, vol. 24, no. 2, pp. 123–140, 1996.

[65] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[66] V. Krishnamurthy, *Partially Observed Markov Decision Processes: From Filtering to Controlled Sensing*. Cambridge University Press, 2016.

Índice de tablas

Esta página ha sido intencionalmente dejada en blanco.

Índice de figuras

Índice de figuras

Esta es la última página.
Compilado el martes 15 septiembre, 2020.
http://iie.fing.edu.uy/