TESIS DEFENDIDA POR

León Felipe Dozal García

Y aprobada por el siguiente comité:

Dr. Gustavo Olague Caballero

Director del Comité

M.C. José Luis Briseño Cervantes

Miembro del Comité

Dr. Rafael Kelly Martínez

Miembro del Comité

Dr. José Luis Medina Monroy

Miembro del Comité

Dr. Pedro Gilberto López Mariscal

Coordinador del Programa en Ciencias de la Computación

Dr. Raúl Ramón Castro Escamilla

Director de Estudios de Posgrado

11 de septiembre del 2006

CENTRO DE INVESTIGACIÓN CIENTÍFICA Y DE EDUCACIÓN
SUPERIOR DE ENSENADA

POSGRADO EN CIENCIAS
EN CIENCIAS DE LA COMPUTACIÓN

Evolución de comportamientos básicos en un robot Pioneer P2-AT

TESIS

que para cubrir parcialmente los requisitos necesarios para obtener el grado de

MAESTRO EN CIENCIAS

Presenta:

León Felipe Dozal García

Ensenada, Baja California, México. septiembre del 2006.

RESUMEN de la tesis de **León Felipe Dozal García**, presentada como requisito parcial para obtener el grado de MAESTRO EN CIENCIAS en CIENCIAS DE LA COMPUTACIÓN. Ensenada, Baja California. septiembre del 2006.

Evolución de comportamientos básicos en un robot Pioneer P2-AT

Resumen aprobado por:

Dr. Gustavo Olague Caballero

Director de Tesis

Esta tesis describe una metodología para evolucionar comportamientos en un robot móvil Pioneer P2-AT. Una estrategia comúnmente empleada en programación de robots móviles es a través de comportamientos. Los comportamientos básicos que se evolucionaron en esta tesis son los siguientes: navegación básica y navegación dinámica de recarga de batería mediante visión. La navegación es escencial para que un robot móvil se desplace y realice tareas más complejas. Nuestra meta es que el robot aprenda de manera automática los comportamientos para que los realize de una forma autónoma. Para este fin utilizamos una red neuronal artificial como neuro-controlador del robot y un algoritmo evolutivo para llevar a cabo el aprendizaje de la red neuronal. Esta estrategia es conocida bajo el nombre de robótica evolutiva. Nuestra implementación se desarrolló bajo un sistema denominado "EvoPioneer". Nuestro trabajo experimental demuestra como los neuro-controladores propuestos, los cuales fueron probados en el robot real Pioneer P2-AT, realizan exitosamente las tareas en simulación y en la práctica. En resumen el robot fue capaz de desarrollar los comportamientos en un ambiente real.

Palabras clave: Algoritmos Evolutivos, Robot Móvil, Robótica Evolutiva..

ABSTRACT of the thesis presented by **León Felipe Dozal García**, as a partial requirement to obtain the MASTER SCIENCE degree in COMPUTER SCIENCES. Ensenada, Baja California. september 2006.

Evolution of basic behaviors on a robot Pioneer P2-AT

Abstract approved by:

Dr. Gustavo Olague Caballero

Thesis director

This thesis describe a metodology for behavior evolution of a mobile robot Pioneer P2-AT. An strategy usually used in mobile robots programing is through behaviors. The basic behaviors that have been evolved in this thesis are the following: basic navigation and navigation with dynamic battery recharge through vision. The navigation is essential in mobile robotics and basic to realize more complicated activities. Our goal is that the robot could be able to learn in an automatic way the behaviors to it self in a autonomous way. To reach this goal we used an artificial neural network as the robot's neuro-controler and an evolutionart algorithm to accomplish the learning of the neural network. This strategy is known as evolutionary robotics. Our implementation was developed on a system called "EvoPioneer". Our experimental work demostrate how the proposed neuro-controlers, which were tested on the real robot Pioneer P2-AT, can realize successfully the activities on simulation and on practice. In summary the robot was capable to develope succesful behaviors on a real environment.

Keywords: Evolutionary Algorithms, Mobile Robot, Evolutionary Robotics..

Dedicado a mis padres

Clara Luz García Sagarnaga
y
Hugo Dozal Aguayo

Agradecimientos

A mis padres, a mi hermana y cuñado
Tania Dozal García y
Juan Carlos Mendoza Lujan.
y muy especialmente a la nueva luz en nuestras vidas
María Elisa Mendoza Dozal.

A mi asesor, Dr. Gustavo Olague Caballero.

A mi comité de tesis, M.C. José Luis Briseño Cervantes
Dr. Rafael Kelly Martínez. y
Dr. José Luis Medina Monroy.

Al Centro de Investigación Científica y de Educación
Superior de Ensenada.

Al Consejo Nacional de Ciencia y Tecnología.

Ensenada, México
11 de septiembre del 2006.

León Felipe Dozal García

Tabla de Contenido

Tabla de Contenido (Continuación)

Tabla de Contenido (Continuación)

Lista de Figuras

Lista de Figuras (Continuación)

Lista de Figuras (Continuación)

Lista de Figuras (Continuación)

Lista de Tablas

Capítulo I

Introducción

En la actualidad el progreso del mundo se desarrolla a velocidades vertiginosas por lo que el hombre ha intentado mantener dicha carrera, lo cual no ha sido tarea fácil. Para ello es necesario entrar en un mundo competitivo donde el tiempo es un factor importante y no se permiten tropiezos. La seguridad en este mundo es otro factor primordial, así como el bienestar y la comodidad los cuales se suman a las demandas del mundo moderno. Hoy en día si un producto, ya sea de investigación, comercial, militar, etc. no cumple con los requerimientos anteriormente nombrados, no es factible utilizarlo.

Hoy en día existe un área de investigación que recientemente ha demostrado resultados excelentes en lo que se refiere a estos factores básicos. La robótica le ha permitido al hombre soñar con el futuro, no existe una concepción del futuro sin robots, ahora es tiempo de alcanzar al futuro. Los avances en robótica son sorprendentes aunque aún están lejos de parecerse a los de la ciencia-ficción, pero lo mejor, es que existe la creencia de que pueden desarrollarse robots con esas capacidades sorprendentes, es decir, pasar de la ficción a la realidad, tarea por demás ardua pero nunca imposible.

El estado actual de la robótica es fácilmente observable. ¿Quién no ha oído del robot enviado a Marte para realizar tareas de reconocimiento o de los robots submarinos o de cualquier otro?

La necesidad inminente es desarrollar sistemas de control para robots basados en información obtenida de sensores conectados al robot. El problema se vuelve más difícil cuando se trata de robots móviles. El desplazamiento del sensor de un lugar a otro

agrega dificultad al problema y su solución no es trivial.

La robótica móvil es un campo multidisciplinario que envuelve las ciencias computacionales y la ingeniería. La robótica móvil está dirigida al diseño de sistemas autónomos, y puede situarse en la intersección de la inteligencia artificial, visión por computadora y robótica, Dudek y Jenkin , 2000.

La robótica evolutiva es un nuevo enfoque el cual mira a los robots como organismos artificiales autónomos que desarrollan sus propias habilidades en una estrecha interacción con el ambiente, y sin intervención humana, Nolfi y Floreano , 2000.

Uno de los problemas para un robót móvil autónomo es transladarse de un lugar a otro sin ninguna intervención humana. Existen multiples aplicaciones para robots móviles si dicha problemática se resuelve: robots paseándose por un edificio, haciendo la limpieza, o autos que se conducen solos en medio de la ciudad, sólo por mencionar algunas.

Los robots presentan limitaciones debido a los sensores que usan. Los sensores visuales, las cámaras por ejemplo, tienen un gran potencial ya que emulan el sentido que más información brinda a los seres vivos a cerca de su medio ambiente, es ahí donde radica su gran potencial. Es por eso que la integración de la información arrojada por los sensores visuales a la realización de tareas robóticas ha tomado mucha importancia.

La visión por computadora es la disciplina cuyo objetivo es proveer del sentido de la vista a robots para que estos puedan interactuar de forma más eficiente en ambientes complejos, Branch y Olague , 2001. El reto ahora es reproducir el proceso de la visión en robots. La visión por computadora porta la estafeta en esta tarea. Esta rama de la inteligencia artificial hace uso de múltiples paradigmas para realizar visión. Como una nueva línea de investigación encontramos a la computación evolutiva la cual ha arrojado resultados alentadores cuando se utiliza para resolver problemas de visión. Así pues la unión de estas dos disciplinas es muy prometedora.

I.1 Descripción del problema

Consideremos un robot en un ambiente delimitado por cuatro paredes y se desea que este robot navegue por la habitación sin chocar en ninguna de las cuatro paredes. Existen varias preguntas que contestar para resolver este problema:

- ¿Cómo debería estar representado el ambiente?
- ¿Cómo debería estar representado el robot?
- ¿Cómo relacionar el conjunto de mediciones del robot con la habilidad de generar comportamientos?

Quizá esta última pregunta es la más importante y la más difícil de contestar. El presente trabajo intenta avanzar más hacia la respuesta de esta pregunta proponiendo una solución con enfoque evolutivo, es decir, utilizando técnicas de robótica evolutiva.

La robótica evolutiva está inspirada en el principio Darwiniano de la reproducción selectiva del más apto, y es muy cercana a las ciencias naturales tales como la biología y la etología. La robótica evolutiva hace uso de herramientas como las redes neuronales artificiales, algoritmos genéticos, sistemas dinámicos e ingeniería biomórfica.

La investigación que nos interesa consiste en encontrar de manera automática los controladores, en este caso son neuro-controladores, que ayuden al robot a realizar una tarea satisfactoriamente. El neuro-controlador es el encargado de transformar las mediciones de los sensores del robot a acciones que el robot ejecuta. Estas acciones deben llevar al robot, en un momento dado, a tomar decisiones correctas y adecuadas a la tarea que está realizando.

El método mediante el cual se evoluciona el neuro-controlador no requiere saber detalles de la configuración física del robot ni tampoco del ambiente en el que se

desenvuelve. Esto permite una gran automatización del proceso, alcanzando en gran medida uno de los objetivos planteados por la robótica evolutiva.

I.2 Investigación previa relevante

El campo de estudio que nos atrae es llamado robótica evolutiva (del ingles evolutionary robotics, ER). Este es un término que ha sido utilizado desde comienzos de la década de los 90's para referirse al estudio y aplicación de un análogo artificial de la evolución Darwiniana para el diseño de robots o agentes simulados. Esta no es una idea nueva, 50 años antes, Alan Turing habló del diseño de redes como las del cerebro a través de "búsqueda genética", Harvey *et al.* , 2005.

Hablar de ER es hablar de computación evolutiva (del ingles evolutionary computing, EC). En los 80's hubo un resurgimiento del interés por las redes neuronales artificiales y el campo de los algoritmos evolutivos comienza a recibir mayor atención. En esa misma década Brooks , 1991a, diseña robots inspirados de los insectos con un comportamiento que mejora y está modelado en el proceso de evolución natural. Un robot estaba contruido con sensores, motores y el mínimo de conexiones nerviosas artificiales necesarias para realizar los comportamientos más simples de movimiento y evasión de obstáculos. Después de que un comportamiento es probado y depurado en un robot, se le incorpora un nuevo comportamiento que interactúa con el ambiente y con los comportamientos preexistentes. El énfasis está en probar, depurar y modificar al robot; el cual a través de este proceso imita el proceso evolutivo (filogenia) de creaturas reales.

Por otro lado, más recientemente en 1996 se desarrolla la arquitectura Saphira que es un sistema integrado de control y sondeo para aplicaciones en robots móviles, Konolige y Myers , 1996. Esta arquitectura permite el control de los comportamientos por medio

del software a diferencia de la metodología de Brooks, donde las adaptaciones se hacen a mano.

Los robots móviles, desempeñan tareas útiles y llegan a ser aceptables en ambientes abiertos, deben ser autónomos: capaces de adquirir información y desarrollar tareas sin intervención. La autonomía tiene muchos aspectos centrales diferentes: la habilidad de atender a otro agente, tomar mediciones del ambiente y realizar tareas que se le asignen. Los tres aspectos involucran complejas operaciones de reconocimiento y planeación por parte del robot, Konolige y Myers , 1996.

El objetivo de la arquitectura Saphira es brindar una base para el desarrollo de comportamientos y realización de tareas predefinidas por parte del robot, comenzando por tareas sencillas y mezclándolas para crear otras más complicadas.

Existe una amplia investigación en robótica evolutiva hecha por Nolfi y Floreano , 2000, que brinda buenos resultados y una buena guía al respecto de lo que se ha hecho en robótica evolutiva. Ellos trabajan en el Instituto de Ciencia Cognitiva y Tecnologías de Laussanne, Suiza, en las áreas de robótica evolutiva, vida artificial, computación evolutiva y redes neuronales.

Una prueba contundente del potencial de la robótica evolutiva es dada a continuación. Recientemente en la universidad de Parma, Italia, investigadores aplican un enfoque evolutivo para controlar la navegación de un robot móvil. En este caso un camión dotado con una cámara a trevés de la cual adquiere imágenes del camino con la intención de avanzar sobre él. Broggi y Cattani , 0, presentan al vehículo automáticamente guiado TerraMax en la primer competencia DARPA Grand Challenge 2004 y son el único grupo que utiliza computación evolutiva para implementar la navegación del vehículo.

El enfoque está inspirado en la optimización de colonias de hormigas (ACO). Este enfoque se ha utilizado con éxito para resolver problemas de optimización como el TSP

(del inglés Traveling Salesman Problem) entre otros y ahora se ha aplicado con éxito al control de la navegación en caminos de tierra difíciles de detectar, incluso para el ojo humano.

I.3 Relevancia del trabajo

La robótica tiene un gran impacto sobre el mundo científico, la seguridad, la medicina y muchos otros ámbitos de la vida del hombre. Es un área que actualmente recibe mucha atención debido a sus grandes aplicaciones y es aquí donde radica la principal importancia de esta área de investigación.

En particular la presente investigación se propone en la robótica móvil y evolutiva. La robótica móvil es el área de la róbotica que presenta más retos, y cualquier aportación es bienvenida. Existen robots móviles que limpian desechos radioactivos, que cargan bombas, que exploran zonas inhóspitas, sólo por nombrar algunos. Tareas que para un hombre son altamente peligrosas y gracias a los robots ahora es posible realizarlas.

La robótica evolutiva ha obtenido buenos resultados, tanto que es considerada como una metodología para la vida artificial. La robótica evolutiva hace énfasis en el conocimiento y en como reproducirlo en agentes simulados o en robots emulando el complicado proceso del pensamiento en criaturas inteligentes.

La navegación es una tarea básica para un robot móvil. Esta le permitirá realizar tareas más complejas como entregar paquetes, seguir a una persona y cosas por el estilo. La autonomía del robot es importante, eso lo convierte en un ente independiente, característica primordial de los seres vivos.

Esta investigación pretende impactar positivamente a la sociedad, aunque en

primera instancia el impacto será dentro de la comunidad científica, no pasará mucho tiempo antes de que la sociedad disfrute de los beneficios.

Si el problema de navegación deja de serlo, las futuras investigaciones podrían enfocarse en otros problemas, esa es la pretensión de esta investigación, aportar una solución nueva y mejorada al problema de navegación, impactando así a la comunidad científica relacionada e interesada en el tema.

I.4 Objetivos del trabajo

El objetivo general de la investigación es:

> Generar comportamientos complejos automáticamente en un robot móvil Pioneer P2-AT de tal forma que los realice de manera autónoma mediante el uso de robótica evolutiva y visión por computadora.

Los objetivos particulares del presente trabajo de investigación son:

1. Implementar un método evolutivo para el desarrollo automático de comportamientos en un robot móvil Pioneer P2-AT.
2. Obtener comportamientos autónomos en un robot móvil Pioneer P2-AT.
3. Demostrar la efectividad de los comportamientos obtenidos en simulación y en la práctica.

I.5 Importancia de la investigación

En visión por computadora el enfoque evolutivo ha obtenido grandes avances los cuales son aplicados a la robótica. La robótica móvil es un área de la robótica que plantea

problemas difíciles de resolver, por ejmplo la navegación. La navegación es una tarea básica para un robot móvil, de ahí la importancia de obtener una solución a este problema.

En la presente tesis se resuelven dos problemas de gran interés combinando estos tres campos de investigación. Los problemas a resolver son:

- Navegación básica.
- Navegación y dinámica de recarga mediante visión.

I.6 Organización de la tesis

Esta tesis está estructurada de la siguiente manera:

- **Capítulo II.** Se introducen los conceptos básicos de robótica evolutiva, acompañados de antecedentes en el área, necesarios para el entendimiento de la investigación aquí descrita.
- **Capítulo III.** Se presentan los conceptos de redes neuronales artificiales y algunas implementaciones. Además de los diferentes algoritmos de aprendizaje relacionados con estas redes.
- **Capítulo IV.** Se muestra la arquitectura Saphira para robots móviles autónomos, la cuál es utilizada en nuestro robot Pioneer P2-AT.
- **Capítulo V.** Se introduce al lector en el tema de los algoritmos genéticos, se tratan conceptos básicos importantes para el entendimiento del enfoque evolutivo.
- **Capítulo VI y VII.** Se describen los fundamentos y detalles del primer y segundo experimento, respectivamente, realizados durante la investigación. Así como los resultados obtenidos durante las pruebas.

- **Capítulo VIII.** Se da una explicación del funcionamiento y uso del sistema "EvoPioneer" para la evolución de comportamientos.

- **Capítulo IX.** Finalmente se plantea la discución de conluciones, aportaciones y trabajo futuro sobre esta investigación.

Capítulo II

Robótica evolutiva

II.1 Introducción

En esta sección se presenta un primer acercamiento al concepto de robótica evolutiva. Mediante un ejemplo sencillo se muestra como es que funciona, los alcances que tiene y sus limitaciones. La robótica evolutiva es un enfoque el cual mira a los robots como organismos artificiales autónomos que desarrollan sus propias habilidades en una estrecha interacción con el ambiente, y sin intervención humana, ver,Nolfi y Floreano , 2000.

II.2 Robótica basada en el comportamiento

La robótica basada en el comportamiento está fundada sobre la ídea de proveer al robot con una colección de comportamientos básicos simples. El comportamiento global del robot emerge a través de la interacción entre esos comportamientos básicos y el ambiente en el cual el robot se encuentra (Brooks , 1986; Arkin , 1998). Los comportamientos básicos están implementados en subpartes separadas del sistema de control y un mecanismo de coordinación es responsable de determinar la fuerza relativa de cada comportamiento en un momento particular. La coordinación puede ser llevada a cabo por medio de métodos competitivos o cooperativos. En los métodos competitivos sólo un comportamiento afecta la salida del motor del robot en un momento particular. En los métodos cooperativos diferentes comportamientos pueden contribuir a una acción

sencilla del motor aunque con diferente fuerza.

En robótica basada en el comportamiento, como en robótica evolutiva, el ambiente juega un papel central en la determinación del rol de cada comportamiento básico en cualquier tiempo dado. Además, estos sistemas son usualmente diseñados a través de un proceso de prueba y error en el cual el diseñador modifica los presentes comportamientos e incrementa el número de comportamientos básicos mientras prueban el resultado del comportamiento global en el ambiente. No obstante, la robótica evolutiva, mediante la realización en un proceso evolutivo automático, usualmente hace un uso más grande del proceso de prueba y error descrito anteriormente. Además, mientras en el enfoque basado en el comportamiento la separación del comportamiento deseado en comportamientos básicos más simples se lleva a cabo intuitivamente por el diseñador, en robótica evolutiva esto es a menudo el resultado de un proceso de auto-organización. Verdaderamente, la organización entera del sistema de evolución, incluyendo su organización en subcomponentes, es el resultado de un proceso de adaptación que usualmente involucra un gran número de evaluaciones de las interacciones entre el sistema y el ambiente.

II.3 Perspectiva desde la ingeniería

En la actualidad nadie discute la dificultad de diseñar sistemas de comportamiento tal como robots móviles autónomos. Como se puede ver cada día, existen programas de computadora eficientes que pueden jugar ajedrez o resolver problemas formales pero no hay robots móviles inteligentes en nuestros hogares o ciudades. La razón principal de porque los robots móviles son difíciles de diseñar, es que su comportamiento debe ser una propiedad emergente de la interacción de su motor con el ambiente. El robot y el ambiente pueden ser descritos como un sistema dinámico porque el estado del

sensor del robot en cualquier tiempo dado es una función de ambas: el ambiente y las acciones previas del robot. El hecho de que el comportamiento sea una propiedad emergente de la interacción entre el robot y el ambiente tiene la consecuencia de que robots simples pueden producir comportamientos complejos. Sin embargo, también tiene la consecuencia de que, como en todos los sistemas dinámicos, las propiedades del comportamiento emergente no pueden ser fácilmente predichas o inferidas de un conocimiento o de las reglas que gobiernan las interacciones. Lo inverso es verdad también: es difícil predecir cuales reglas producirán un comportamiento dado, ya que el comportamiento es el resultado emergente de la interacción dinámica entre el robot y el ambiente.

La estrategia convencional seguida para evitar estas dificultades ha sido divide y vencerás: En donde se encuentra una solución, dividiendo el problema en una lista optimista de problemas más simples. Los enfoques clásicos de robótica han asumido a menudo una ruptura primaria entre Percepción, Planeación y Acción. Esta forma de dividir el problema ha producido resultados limitados y ha sido criticado por un número de investigadores. Brooks , 1986, propone un enfoque radicalmente diferente en el cual la división se lleva a cabo al nivel del comportamiento. El comportamiento deseado se separa en un conjunto de comportamientos básicos más simples, los cuales son modulados a través de un mecanismo de coordinación. En este último enfoque el sistema de control es construido incrementalmente nivel por nivel, donde cada nivel es responsable de un solo comportamiento básico ligando directamente sensores a motores. Los comportamientos básicos simples se implementan primero, entonces nuevos niveles que implementan otros comportamientos básicos son adheridos uno a la vez después de intensas pruebas y depuraciones. Este enfoque ha probado ser más exitoso que el enfoque clásico. Además a mostrado que ambos niveles (módulos) responsables de los comportamientos básicos simples y el mecanismo de coordinación puede ser obtenido

a través de un proceso de auto-organización más que por un diseño explícito.

Los enfoques de descomposición basados en el comportamiento, dejan al diseñador la decisión de cómo romper el comportamiento deseado en comportamientos básicos simples. Desafortunadamente, no es claro cómo un comportamiento debería ser descompuesto y es muy difícil llevar a cabo tal descomposición a mano. Incluso investigadores que adoptaron exitosamente la descomposición de comportamientos e integración sienten que este es un problema crucial. Rodney Brooks, por ejemplo, anota: *"En cambio, dados muchos comportamientos presentes en un sistema basado en el comportamiento, y sus dinámicas individuales de interacción con el mundo, es a menudo difícil decir que una serie particular de acciones fue producida por un comportamiento particular. Algunas veces muchos comportamientos estan operando simultáneamente, o están alternando rápidamente"* (Brooks , 1991b, pp.584-585). Colombetti *et al.* , 1996 anotan al final de su artículo: *"Las técnicas de aprendizaje deben ser extendidas a otros aspectos del desarrollo del robot, como la arquitectura del controlador. Esto significa que la estructura de los módulos de comportamiento deberían emerger del proceso de aprendizaje, en lugar de ser prediseñada"*.

Para entender mejor por qué es difícil dividir un comportamiento global en un conjunto de comportamientos básicos más simples, tenemos que distinguir dos formas de describir comportamientos: una descripción desde el punto de vista del observador y una descripción desde el punto de vista del robot. La descripción *distante* es una descripción desde el punto de vista del observador en la cual términos de alto nivel tal como alcanzar o discriminar son usados para describir el resultado de una secuencia de ciclos sensor-motor. La descripción próxima es una descripción desde el punto de vista del sistema agente del sensor-motor que describe cómo el agente reacciona en diferentes situaciones sensoriales.

Debería notarse que la descripción distante es el resultado no sólo de la descripción

próxima, sino también del ambiente. Más precisamente, la descripción distante es el resultado de una interacción dinámica entre el agente y el ambiente. De hecho, los patrones sensoriales que el ambiente provee al agente, determinan parcialmente las reacciones del motor del agente. Estas reacciones del motor, modifican el ambiente o la posición relativa del agente en el ambiente y, por lo tanto, determina parcialmente el tipo de patrones sensoriales que el agente recibirá del ambiente.

Esta interacción dinámica puede explicar por qué es difícil dividir un comportamiento en un conjunto de comportamientos básicos que son simples desde el punto de vista de la descripción próxima. En general, la división se lleva a cabo intuitivamente por el investigador en la base de una descripción distante del comportamiento global. Sin embargo, ya que el comportamiento deseado es el resultado emergente de la interacción dinámica entre el sistema de control del agente y el ambiente, es difícil predecir cual tipo de comportamiento será producido por un sistema de control dado. Lo opuesto también es verdad - es difícil predecir cual sistema de control producirá un comportamiento deseado.

Si no hay una transformación uno a uno entre subcomponentes del sistema de control del agente, y subcomponentes del correspondiente comportamiento desde el punto de vista de la descripción distante, es cuestionable cuando una descripción distante puede usarse efectivamente para determinar la manera en que el sistema de control del agente debería ser estructurado. En otras palabras, el hecho de que un comportamiento global puede dividirse en un conjunto de comportamientos básicos más simples desde el punto de vista de la descripción distante no implica que tales comportamientos básicos del sistema de control del agente pueden ser implementados en niveles (módulos) separados y relativamente simples.

La robótica evolutiva, dependiendo de una evaluación del sistema como un todo y de su comportamiento global, libera al diseñador de su carga de decidir cómo dividir

el comportamiento deseado en comportamientos básicos simples.

II.4 Perspectiva etológica

Moléculas, células, órganos, organismos y ecosistemas son todos entidades a diferentes niveles que son potencialmente relevantes para el estudio del comportamiento. Como una consecuencia, el estudio del comportamiento está siendo conducido dentro de diferentes disciplinas. En particular dos grupos de disciplinas pueden ser identificadas las cuales están separadas por una total heterogeneidad de los conceptos que usan para describir, analizar y explicar su objeto de estudio: biología molecular, biología celular, biología de desarrollo, genética y neurociencia, por un lado; psicología, ecología y biología evolutiva por el otro lado. Por una parte las neurociencias, por ejemplo, usan conceptos que son apropiados para describir un objeto físico (i.e. el sistema nervioso). Por otro lado, la psicología emplea conceptos que normalmente no se refieren a estructuras físicas o procesos. Los conceptos utilizados por los psicoanalistas para hablar acerca del comportamiento y cognición son derivados de la filosofía y de los conceptos que usamos cada día de la vida para describir, predecir y explicar nuestro propio comportamiento y el comportamiento de otras criaturas vivientes.

Dada la existencia de dos diferentes vocabularios, sólo es posible mirar a posteriori por correlaciones entre fenómenos físicos y anátomo-fisiológicos por un lado y fenómenos psicológicos por el otro. Esta es la tarea de tales disciplinas como psicofísica y neuropsicología las cuales sirven como puentes. Sin embargo, es muy difícil seguir las observaciones concernientes al comportamiento de la estructura de una sola (e.g. el sistema nervioso) entidad que podría ser descrita y analizada usando un vocabulario teórico sencillo.

En la última década, nuevos campos de investigación los cuales tratan de vencer

esta brecha etimológica han sido desarrollados: conexionismo y cognición incorporada. El conexionismo propone redes neuronales como un marco teórico unificado para estudiar ambos: comportamiento y cognición, por un lado, y el sistema nervioso, por el otro. Por lo tanto, el conexionismo puede ser visto como un intento de deshacer el dualismo tradicional abrazado por la psicología. La cognición incorporada acentua la importancia de los aspectos físicos de un sistema (forma física, gravedad, fricción, inercia, idiosincracia de cada sensor y actor). Por otra parte, acentua la importancia de la interacción con el ambiente.

La percepción es a menudo vista como un proceso no incorporado en el cual el agente es pasivamente expuesto a una corriente continuamente cambiante de estimulación sensorial sin consideración de lo que el agente necesita hacer. En contraste, un fuerte acoplamiento entre acción y percepción es una tema central en cognición incorporada. Explotando su interacción con el ambiente (i.e. por coordinación sensor-motor), los agentes pueden modificar parcialmente el estímulo que ellos reciben del ambiente. Por lo tanto, el comportamiento no puede ser considerado un producto de un agente aislado del mundo, pero puede sólo emerger de un fuerte acoplamiento del agente con su propio ambiente. En otras palabras, un comportamiento dado no puede ser explicado sobre la base de mecanismos internos solamente. Similarmente, la estructura de un sistema nervioso produciendo un comportamiento dado no puede ser considerado en aislamiento del ambiente. La función de un sistema nervioso de un agente, de hecho, es la coordinación de percepción y acción para generar un comportamiento adaptativo.

En la mayoría de la invetigación en robótica, el poder de la interacción con el ambiente es ampliamente inexplorada. Algunas notables excepciones podrían ser mencionadas. Los vehículos de Braitenberg son probablemente la primer demostración clara de que un simple agente puede producir un comportamiento complejo explotando la

complejidad del ambiente, Braitenberg , 1984. Pfeifer y Scheier mostrarón que el problema de percepción podría ser simplificado enormemente si el movimiento del agente y su interacción con el ambiente se toma en cuenta, Scheier y Pfeifer , 1995. Ellos describen un robot Khepera que puede aprender a discriminar entre cilindros pequeños y grandes dependiendo de tres simples comportamientos básicos prediseñados (i.e. mover hacia delante, evadir objetos, voltear hacia un objeto). Esto es posible por que la coordinación sensor-motor hace al agente rodear objetos lo cual afecta significativamente el tipo de información sensorial que el robot recibe del ambiente.

La razón por la que el poder de la interacción con el ambiente está todavía inexplorada en robótica, como apuntamos anteriormente, es que el comportamiento adaptativo es difícil de obtener a través del diseño. En particular, es difícil diseñar sistemas que explotan la coordinación sensor-motor. Para agentes que interactúan con un ambiente externo, cada acción del motor tiene dos diferentes efectos: (a) determina parcialmente qué tan bien se desenvuelve el agente con respecto a una tarea dada; (b) determina parcialmente el siguiente patrón sensorial que el agente recibirá del ambiente, el cual a su vez puede determinar si el agente será apto para resolver su tarea o no. El problema es determinar cual acción del motor del agente debería realizar a cada momento tomando en cuenta (a) y (b), esto es muy difícil dado que: cada acción de motor puede tener consecuencias a largo término; el efecto de cada acción de motor es una función también de las acciones precedentes y sucesivas; el efecto de cada acción es una función de la interacción entre el agente y el ambiente.

La robótica evolutiva, por su larga dependencia en la auto-organización, no se afecta por estos problemas y, como una consecuencia, es un marco ideal para estudiar el comportamiento adaptativo. Efectivamente, muchos de los robots evolucionados explotan la interacción activa con el ambiente para maximizar su criterio de selección.

II.5 Perspectiva biológica

La robótica evolutiva y la biología comparten un interés en la siguiente pregunta fundamental: ¿Cuáles son las características clave de la evolución natural que la hacen tan exitosa en producir la extraordinaria variedad de formas de vida altamente adaptada presente en el planeta? Producir mejores respuestas a ésta pregunta puede incrementar significativamente nuestro entendimiento de los sistemas biológicos y nuestra habilidad para diseñar sistemas artificiales. Desde el punto de vista de robótica evolutiva dicha cuestión puede exponerse como sigue: ¿En cuáles condiciones es más probable que un proceso evolutivo artificial seleccione buenos individuos que desarrollen competencias complejas para adaptarse a su ambiente artificial? Como veremos, es posible, en principio, desarrollar formas complejas de comportamiento sin incrementar la cantidad de supervisión. Esto puede ser alcanzado de las siguentes formas: (a) generando evolución incremental a través de competencias entre especies o dentro de las especies; (b) dejando al sistema libre de decidir cómo extraer supervisión del ambiente; (c) incluyendo la tranformación genotipo-fenotipo dentro del proceso evolutivo.

II.6 Evolución Incremental

Desde el punto de vista de la robótica evolutiva, una pregunta clave es: ¿Cómo la evolución artificial puede seleccionar individuos que tengan capacidades para permitirles resolver tareas complejas? (e.g. navegar en un ambiente complejo). Un problema relacionado en biología es entender cómo y en cuáles circunstancias la evolución natural puede descubrir nuevas capacidades (e.g. la habilidad de volar o construir un nido). Para evitar la confusión, deberíamos clarificar aquí que los criterios de capacidades y selección son entidades diferentes. La selección de la evolución natural esta basada en un criterio simple y general: la habilidad de reproducirse. A pesar de esto, la evolución

natural ha sido capaz de producir individuos con capacidades sofisticadas tales como la habilidad de nadar, volar, y comunicarse a través del lenguaje natural.

Si se desea seleccionar individuos capaces de resolver una tarea que requiere una habilidad especifica a través de evolución artificial, lo más sencillo de hacer es seleccionar los individuos por su habilidad para resolver esa tarea especifica. Esto se reduce a diseñar un criterio de aptitud que califique a los individuos de acuerdo a su habilidad de resolver la tarea. Sin embargo, es fácil mostrar que esta simple estrategia puede trabajar sólo para tareas simples. Cuando la complejidad de la tarea se incrementa, la probabilidad de que algunos individuos de las primeras generaciones sean capaces de llevarla a cabo disminuye. La tarea es inversamente proporcional a la complejidad de la tarea misma. Para tareas complejas, todos los individuos de las primeras generaciones son calificados con el mismo valor nulo, y como consecuencia el proceso de selección no puede operar. Nos referiremos a este hecho como el *problema de autosuficiencia*.

Una posible solución a este problema es incrementar la cantidad de supervisión. Los elementos del experimento pueden ser utilizados para incluir en el criterio de selección recompensas para subpartes de la tarea deseada. Otra posibilidad es comenzar el proceso evolutivo con una versión simplificada de la tarea y entonces progresivamente incrementar su complejidad modificando el criterio de selección (la última técnica es usualmente llamada *evolución incremental*). Esta idea fue previamente propuesta en el contexto de sistemas clasificadores por Dorigo y Colombetti , 1994, quienes le llamaron *modelado*, tomando prestado el término de las técnicas de psicología experimental empleadas para entrenar animales para producir respuestas predefinidas. Estas técnicas, ya que requieren más supervisión, incrementan el riesgo de introducir restricciones inapropiadas. Sin embargo, desde una perspectiva ingenieril, es más fácil usar los elementos del experimento para modelar el criterio de selección que para diseñar el sistema de control en si.

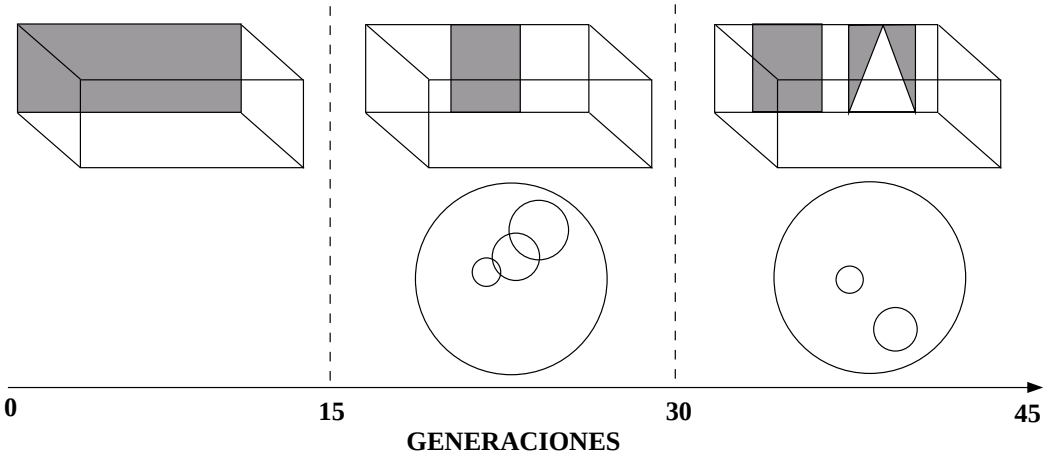


Figura 1: Modificación del ambiente a lo largo de las generaciones, incluyendo una representación esquemática de la posición y ancho de los campos receptivos de la unidades de visión usadas por los controladores mejor evolucionados.

Por ejemplo, Harvey *et al.* , 1994, han evolucionado incrementalmente comportamientos guiados visualmente y morfologías sensoriales para un robot móvil que se espera discrimine formas y navegue hacia un rectángulo mientras evade al triángulo (Figura 1). Los autores comenzaron con una versión simple de la tarea que selecciona controladores por su habilidad para navegar hacia una pared cubierta por un papel de color. Sucesivamente, los autores redujeron el área del papel de color en la pared y reanudaron la evolución desde la última generación evolucionada. Finalmente, la forma rectángular fue desplazada y una forma triángular fue puesta cerca de la primera. La función de aptitud fue modificada para fomentar la evasión del triángulo y alcanzar el rectángulo, y la población previamente evolucionada fue incrementalmente evolucionada en el nuevo ambiente. La morfología sensorial evolucionada al final utiliza solo dos píxeles para detectar cuando la forma percibida fue rectángular o triángular.

Este experimento muestra cómo manipular las experiencias de aprendizaje a través de generaciones e introduciendo más restricciones en el criterio de selección para resolver tareas que no podían ser resueltas de otra manera. Sin embargo, existe un peligro en el que restricciones adicionales pueden canalizar el proceso de evolución en

direcciones equivocadas.

Una solución más deseable al problema de autosuficiencia, sería un proceso auto-organizado capaz de producir evolución incremental que no requiera ninguna supervisión humana.

II.7 Extracción de supervisión del ambiente a través del aprendizaje en vida

Desde el punto de vista de un organismo natural o artificial el ambiente externo no provee ninguna pista directa sobre cómo el agente debería actuar para alcanzar una meta dada. Sin embargo, los agentes reciben una gran cantidad de información del ambiente a través de los sensores. Tal información, la cual es una función de la estructura ambiental y de las acciones del motor del agente puede ser utilizada no sólo para determinar cómo reaccionar en diferentes circunstancias ambientales, sino también para adaptarse al ambiente actual a través del aprendizaje en vida. Por ejemplo, un robot puede aprender las consecuencias de diferentes acciones en diferentes contextos ambientales.

Al principio, en una población en evolución, ninguna habilidad la cual puede ser adquirida a través del aprendizaje en vida puede también ser genéticamente adquirida a través de la evolución. No obstante estas dos maneras de adaptación al ambiente difieren en un aspecto importante: la adaptación ontogenética puede depender de una muy rica, aunque menos explícita, cantidad de supervisión. Desde el punto de vista de la adaptación filogenética, los individuos son evaluados solo una vez en la base de un solo valor, el cual codifica que tan bien estuvieron adaptados a su ambiente a lo largo de todo su tiempo de vida (i.e. el número de hijos en el caso de evolución natural y el valor de aptitud en el caso de evolución artificial). En cambio, desde el punto de

vista de la adaptación ontogenética, los individuos reciben información del ambiente a través de sus sensores a lo largo de toda su vida. Sin embargo, esta enorme cantidad de información codifica sólo muy indirectamente que tan bien lo hizo un individuo en diferentes momentos de su propia vida o cómo debería modificar el comportamiento para incrementar su propia aptitud. El problema es entonces el entender cómo tal información puede ser transformada en una indicación de lo que el agente debería hacer o que tan bien lo está haciendo.

Por los problemas que se discutieron en secciones anteriores, es probablemente difícil diseñar un sistema capaz de realizar una buena transformación. Por otro lado, podemos esperar que la evolución resuelva este tipo de problema produciendo subsistemas capaces de extraer autónomamente información de supervisión que puede ser usada para aprendizaje rápido en vida. Esto ha sido mostrado en dos experimentos computacionales llevados a cabo por Ackley y Littman , 1992, y Nolfi y Parisi , 1997. En ambos casos la arquitectura controladora fue dividida en dos submódulos de los cuales, el primero tiene la función de determinar cómo reaccionar al estado sensorial actual y el último tiene la función de generar una señal de enseñanza para el primero. En Ackley y Littman los estados de los sensores fueron transformados en señales de reforzamiento, mientras en Nolfi y Parisi los estados de los sensores fueron transformados en señales de enseñanza autogeneradas. Estos últimos sujetando los pesos de las dos subredes a un proceso evolutivo, los autores reportaron la evolución de individuos que aprenden durante su vida a adaptarse al ambiente a través de señales autogeneradas, transformando la información sensorial en señales de reforzamiento utiles o señales de enseñanza. Como mostraron Miller y Todd , 1990, un resultado similar puede ser obtenido evolucionando redes neuronales con topologías que pueden variar evolutivamente y que aprenden a lo largo de su vida con aprendizaje no supervisado. En estos casos las restricciones en la arquitectura canalizan los cambios manejados por

los estados de los sensores en las direcciones correctas.

Como se mencionó anteriormente, lo que puede ser obtenido con evolución y aprendizaje puede tambien ser obtenido sólo con evolución. En un nivel alto de descripción, por ejemplo, un individuo que nace con una estrategia general capaz de producir un comportamiento el cual es efectivo en un conjunto de diferentes ambientes, es equivalente a otro individuo capaz de adaptarse a cada ambiente a través del aprendizaje en vida. Por otro lado, en un nivel más bajo de descripción es claro que estos dos individuos están organizados en diferentes maneras. Individuos que no comienzan con una estrategia general pero se adaptan a lo largo de su vida deberían ser capaces de detectar el ambiente en el que están localizados, y asimismo deberían ser capaces de modificar su estrategia de manera acorde (llamaremos a estos individuos *maleables*). Por otro lado, los individuos que tienen una estrategia general ya lista para diferentes ambientes no necesitan cambiar (llamaremos a estos individuos *completos*). Desde este punto de vista los individuos completos serán más efectivos que los individuos maleables porque no tienen que sufrir un proceso de adaptación a través de su vida. Sin embargo, puede suceder que en ciertas condiciones los individuos completos no puedan ser seleccionados porque una estrategia completa no existe (o porque es demasiado complejo y por lo tanto la probabilidad de ser seleccionado es muy baja). Si este es el caso, una solución maleable es la única opción disponible.

II.8 Desarrollo y evolución de la evolucionalidad

Uno de los aspectos de la evolución natural que es más crudamente simplificada en robótica evolutiva es el desarrollo de la evolucionalidad. Esto puede explicar porque recientemente la importancia del desarrollo de la evolucionalidad es una de los temas más debatidos en robótica evolutiva. De manera interesante, la importancia y el rol

del desarrollo de la evolucionalidad en la evolución natural es un tema controversial en biología evolutiva también.

Desde una perspectiva de robótica evolutiva, este tema es usualmente referido como el problema de transformación *genotipo - fenotipo*. Como expusieron Wagner y Altenberg , 1996: "Para que la adaptación ocurra, estos sistemas deben poseer *evolucionalidad*, i.e. la habilidad de variaciones aleatorias para producir mejoras algunas veces. Se encontró que la evolucionalidad depende criticamente de la forma en que la variación genética se transforma a variación fenotípica, un tópico conocido como *problema de representación*".

La transformación más simple genotipo - fenotipo es una transformación uno a uno en la cual cada gen codifica para un solo carácter del robot. No obstante, en la mayoría de los experimentos conducidos en robótica evolutiva, la transformación es más compleja que eso. Puede involucrar: (a) algunos niveles de organización (e.g. genotipo, sistema nervioso, y comportamiento) los cuales están jerarquicamente organizados e involucran interacciones no lineales entre diferentes niveles; (b) instrucciones cada vez mayores que son recursivas en el sentido de que son aplicadas a sus propios resultados en una manera que se parece al proceso de duplicación de células y diferenciación; (c) los genotipos que varian en longitud.

La oportunidad para estudiar estas características del desarrollo de la evolucionalidad en aislamiento, también para manipular la forma en que son modeladas, y para generar enormes cantidades de datos fácilmente, puede permitir a la robótica evolutiva ayudar a entender la importancia y el rol del desarrollo de la evolucionalidad desde una perspectiva evolutiva.

Otra diferencia importante entre evolución natural y artificial es que en el primer caso la transformación está sujeta al proceso evolutivo mientras que en el último caso la transformación es generalmente diseñada por el experimentador (aunque es diseñada

tomando inspiración en cómo se lleva a cabo en organismos naturales). Desafortunadamente, también en este caso, es difícil diseñar una buena transformación genotipo - fenotipo. De manera similar, es difícil darle forma a la transformación sólo imitando a la naturaleza dado que no todo es conocido y que, por razones prácticas, sólo algunos aspectos del proceso natural pueden ser modelados.

Todos estos problemas y el hecho de que todavía no es claro cómo la transformación puede estar sujeta al proceso evolutivo, explican porque sólo resultados limitados han sido obtenidos en el intento de evolucionar comportamientos más complejos empleando genotipos más plausibles.

II.9 Cómo evolucionar robots

La evolución artificial de robots puede tomar un tiempo considerable para realizar las pruebas de aptitud requeridas por los individuos. Los resultados obtenidos son dignos de todo el tiempo que toman. Así con objeto de evolucionar los controladores de los robots, se deben tomar en consideración los siguientes tópicos:

- Robustez mecánica. Muchos individuos (principalmente en generaciones cercanas) pueden producir comportamientos que pueden estar dañando al robot físicamente, tales como colisiones a alta velocidad contra objetos, o fuertes cargas ininterrumpidas a los motores empujando hacia un muro.

- Suministro de energía. Los experimentos evolutivos son más largos que el promedio de duración de las baterías; es necesario encontrar una manera para proveer de energía o sobrepasar este problema en otras formas.

- Análisis. El sistema de control de robots evolucionados puede ser muy complejo.

Lo cual se traduce en la dificultad. Esto está intimamente relacionado al ambiente, por tal motivo es difícil de entender por simple observación. Una solución puede encontrarse en métodos inspirados en la neuro-etología que pueden proveer elementos útiles correlacionando el comportamiento y las dinámicas de controladores evolucionados.

- Tiempo de eficiencia. Una ejecución evolutiva puede durar horas, días, o incluso semanas. En algunas circunstancias este tiempo puede ser considerablemente reducido sin reducir necesariamente el número de pruebas de aptitud, como se podrá apreciar en este trabajo.

- Diseño de funciones de aptitud. El criterio de selección puede tener una mayor influencia sobre los resultados de una ejecución evolutiva, pero puede ser difícil el poder diseñar a priori una función efectiva para robots autónomos que operen en ambientes parcialmente desconocidos.

II.9.1 Evolución de robots físicos

El Pioneer P2-AT es un robot móvil de la familia de los robots Pioneer, cuya arquitectura fue desarrollada por Kurt Konolige, investigador de la SRI internacional. En la tabla I se muestran todas las caracteristicas físicas del Pioneer P2-AT.

Pioneer P2-AT viene con una variedad de programas computacionales para robots moviles:

- El desarrollo Saphira cliente provisto con Colbert.

- Simulador Pioneer.

- Aplicación de interface Pioneer.

Tabla I: Características del robot móvil Pioneer P2-AT.

Características físicas	**Pioneer P2-AT**
Largo	50 cm.
Ancho	49 cm.
Alto (cuerpo)	24 cm.
Peso	14 kg.
Carga útil	20 kg.
Poder	
Batería	3x12V
Carga	252 watt-hr
Duración	4-6 hrs.
Tiempo de recarga	8 hrs.
Movilidad	
Ruedas	4 neumáticos
Diámetro de la rueda	22 cm.
Ancho de la rueda	7.5 cm.
Dirección	Derrape
Radio de vuelta	0 cm.
Velocidad de translación max.	2 m/sec
Velocidad de rotación max.	360 grados/sec
Terrenos atravesables	Todo terreno
Sensores	
Ultrasonicos frontales	8
	1 a cada lado
	6 hacia delante a intervalos de 20°
Ultrasonicos posteriores	8
	1 a cada lado
	6 hacia atras a intervalos de 20°
Decodificadores de posición	9,850 señales por revolución
Electrónica	
Procesador	Siemens 8C166 (20 MHz)
RAM	256 KB
Interruptores de poder	1 principal; 2 auxiliares
RADIO	Radio modem o Ethernet

Saphira viene con un lenguaje de línea de mando interactiva, con Colbert, y con un programa de demostración que permite el control del manejo del Pioneer de manera manual (con el teclado o un joystick) y de manera automática. El programa también permite activar muchos comportamientos robóticos, incluyendo evasión de coliciones, reconocimiento de características, y navegación autónoma.

La aplicación de interface Pioneer es un desarrollo en lenguaje C que trabaja con Saphira. Permite a los desarrolladores ganar más control de los detalles de bajo nivel del servidor del Pioneer.

El tamaño del robot Pioneer puede traer problemas en los experimentos evolutivos. Por ejemplo, es difícil para el experimentador construir ambientes complejos en espacios limitados. Para un robot como el Pioneer P2-AT se necesitan espacios amplios del tamaño de una habitación grande, un corredor, etc. Un espacio pequeño de trabajo permite un monitoreo más fácil del robot. Las leyes fundamentales de la física le dan mayor robustez mecánica a los robots miniatura. Con el fin de entender este fenomeno físico, se compara un robot con un diámetro de 50 mm chocando contra un muro a una velocidad de 50 mm/seg con un robot como el Pioneer con un diámetro de 70 cms chocando contra la misma pared a una velocidad de 1 m/seg. El robot miniatura resistirá la colisión, mientras que el Pioneer probablemente reportará serios daños.

Los controladores evolucionados no pueden ser entendidos aislandolos del ambiente y observando su estructura porque su funcionamiento está intimamente relacionado a interacciones físicas entre el robot y el ambiente. Además los neuro-controladores evolucionados pueden tener comportamientos dinámicos complejos que aparecen sólo bajo un comportamiento autónomo normal del robot. Esta situación es similar a la que encaran los neuroetologistas intentando entender ¿cómo trabaja el cerebro de un animal? Estudios *in vitro* del cerebro pueden decir muy poco a cerca de la función de comportamiento de un circuito neural. Por lo tanto, la práctica común es implantar

electrodos en el cerebro y correlacionar la actividad neuronal con el comportamiento mientras el animal realiza algunas tareas. Desde esta perspectiva, un robot autónomo evolucionado después de generaciones de libre interacción con el ambiente es muy similar a la de un animal.

II.9.2 Evolución en simulación

Aunque la evolución en robots reales es factible, la evaluación en serie de individuos sobre un solo robot físico debería requerir un tiempo muy largo. El experimento de la evolución de navegación realizado por Floreano y Mondada , 1996, el cual es uno de los más complejos experimentos llevados a cabo en robots físicos, tomó 10 días. En principio, teniendo a disposición un gran número de robots y probando más individuos en paralelo podría reducir significativamente el tiempo requerido. Por ejemplo, probando la población entera en paralelo en 80 robots físicos, el mismo experimento habría tomado solo algunas horas. No obstante, uno debería tener en mente que robots diferentes, incluso si son idénticos desde el punto de vista electrónico y mecánico, podrían diferir significativamente entre ellos mismos. Para probar la población en paralelo la descendencia de los individuos seleccionados debería ser probada en diferentes cuerpos. Si los individuos seleccionados son incapaces de adaptarse a una estructura de cuerpo nueva durante su tiempo de vida, la evolución artificial podría ser incapaz de seleccionar mejores individuos. Watson *et al.* , 1999, por ejemplo, evolucionó individuos en paralelo usando 8 robots identicos. Sin embargo, para evitar el problema descrito anteriormente ellos desarrollaron un nuevo algoritmo en el cual los individuos exitosos no se reproducen pero transmiten sus genes a otros individuos de una población que nunca muere.

Una manera de evitar el problema de tiempo es evolucionar robots en simulación y entonces correr los más exitosos individuos en el robot físico. Esta estrategia es la que

se adoptó para la realización de este trabajo. Los robots simulados de hecho podrían correr más rápido que su contraparte real. Además, el poder de las computadoras en paralelo puede ser fácilmente explotado para correr más individuos al mismo tiempo. La ventaja en términos de tiempo, sin embargo, depende de la complejidad de la interacción simulada robot-ambiente. La visión simulada, por ejemplo, es generalmente un proceso que consume tiempo.

La simulación no obstante presenta otros problemas. Como fue declarado por Brooks , 1992:

(1) "Sin una validación regular en robots reales hay un gran peligro de que mucho esfuerzo se irá en resolver problemas que simplemente no se pueden realizar en el mundo real con un robot (o robots) físico"

(2) "Hay un peligro real (de hecho, una certidumbre) de que programas que trabajan bien en robots simulados fallarán completamente en robots reales por las diferencias de sensado y actuación en el mundo real –es muy difícil simular las dinámicas actuales del mundo real".

El simulador Pioneer con el que se cuenta es muy exacto. Conforme pasa el tiempo los simuladores se vuelven más apegados a la realidad, lo que se vuelve una ventaja.

Muchos experimentos evolutivos recientemente conducidos en simulaciones y validados exitosamente en robots reales demostraron que, al menos para una clase restringida de interacciones robot-ambiente, es posible desarrollar exitosamente individuos-robot en simulación que generen también comportamientos idénticos en el ambiente real. Por lo tanto, aunque el proceso evolutivo en el robot físico, cuando es factible, es altamente preferible (una simulación nunca será una copia exacta de la interacción robot-ambiente) y aunque no es claro si los métodos actuales pueden escalar significativamente a casos más complejos, las simulaciones validadas en robots reales pueden ser

una herramienta justificable bajo algunas circunstancias. El uso de simulación simplifica aspectos tales como abastecimiento de energía y restablecer el estado del ambiente al principio de cada época.

II.9.3 Métodos para modelar aproximadamente las interacciones robot-ambiente

Para modelar aproximadamente las interacciones robot-ambiente se deben encarar diferentes problemas. Un primer problema se debe al hecho de que *diferentes sensores y actuadores, incluso si son aparentemente identicos, se pueden desempeñar de manera diferente por las ligeras diferencias en su electrónica o mecánica.*

Por ejemplo, una inspección de los sensores infrarojos de diferentes robots Khepera muestra que las respuestas de los sensores varían por causa de las ligeras diferencias en el ambiente (e.g. el ajuste de las luces ambientales, color, y forma de los objetos). Otras importantes fuentes de variación de respuesta son las propiedades intrinsecas de cada sensor. En otras palabras, cada sensor responde de una manera muy diferente de otros sensores cuando se expone a los mismos estímulos externos. Considérese por ejemplo la activación de los 8 sensores infrarojos de un robot Khepera ubicado a diferentes ángulos y distancias con respecto a objetos cilíndricos. Diferentes sensores tienen además diferentes rangos de sensibilidad. El octavo sensor es sensitivo dentro de los rangos de ángulo y distancia que están cerca del doble de largo de los del cuarto sensor. Esto también implica que dos robots diferentes se pueden desempeñar de manera muy diferente en condiciones identicas por las diferencias en las caracteristicas sensoriales.

Como se sugirió en Miglino *et al.* , 1995, una manera de tomar en cuenta la idiosincrasia de cada sensor es muestrear empíricamente las diferentes clases de objetos en

el ambiente a través de los sensores del robot. Considérese por ejemplo el caso de un ambiente consistente en una arena rodeada por paredes las cuales contienen un cierto número de objetos cilíndricos del mismo tamaño. Este simple ambiente contiene dos clases de objetos: paredes y cilíndros. Se puede muestrear este ambiente posicionando el robot en diferentes ángulos y distancias con respecto a una pared y con respecto a un cilíndro, y almacenando el estado de activación correspondiente de los sensores infrarojos. En el caso de Miglino *et al.* , 1995, por ejemplo, los objetos fueron muestreados en 180 orientaciones y en 20 diferentes distancias. Las matrices de valores de activación resultantes pueden entonces utilizarse por el simulador para establecer los niveles de activación de los sensores infrarojos dependiendo de su posición actual en el ambiente simulado.

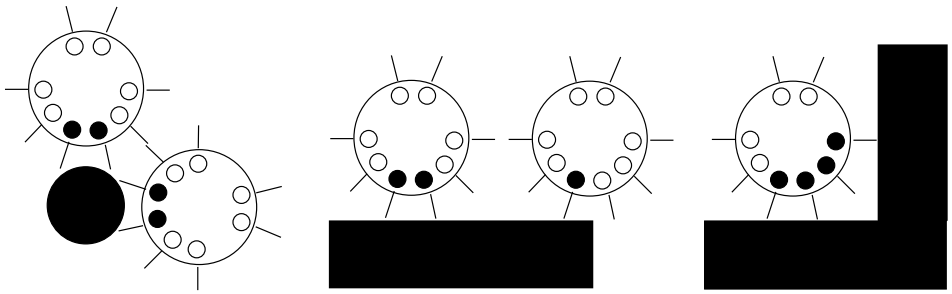


Figura 2: **Izquierda:** En el caso de un objeto cilíndrico, los sensores activos (puntos negros) son una función del ángulo y la distancia entre en robot y el objeto. **Centro:** En el caso de una caja rectangular, el patrón de sensores activos también depende de la porción del objeto percibido. **Derecha:** Los sensores podrían ser simulados para más de un objeto. En este ejemplo, están simulados para dos objetos de la misma clase.

Los individuos evolucionados en simulaciones basadas en esta técnica continuan su desempeño satisfactoriamente cuando se transfieren al ambiente real. Desafortunadamente, su aplicación a ambientes complejos puede volverse costosa porque cada clase de objetos debe también ser muestreada y objetos asimétricos deben ser muestreados desde diferentes puntos de vista. De hecho, mientras la percepción de objetos simetricos tales como cilíndros es sólo una función de la orientación y distancia del robot

con respecto al objeto (Figura 2, izquierda). Para otros objetos, tales como una caja rectangular, el estado perceptual es también una función de la forma local del objeto (Figura 2, centro). Un problema adicional surge cuando el robot simulado está cerca de dos diferentes objetos (por ejemplo en el caso de una esquina como en la Figura 2, derecha). Una solución consiste en sumar los vectores muestreados que el robot experimentaría en la proximidad de cada objeto tomado en aislamiento, pero ésto puede introducir discrepancias significantes entre la simulación y el ambiente real.

Cuando el uso de esta técnica de muestreo se vuelve muy costosa se puede depender de modelos matemáticos. También en este caso, los parámetros de las funciones que describen la interacción robot-ambiente puede ser determinada empiricamente empleando el robot real. Esta técnica fue adoptada por Jakobi *et al.* , 1995, quien desarrolló un simulador para un robot Khepera. Los autores modelaron las velocidades de las ruedas, los sensores infrarojos y de luces ambientales del robot con un conjunto de ecuaciones generales y entonces establecieron las constantes de tales ecuaciones almacenando los estados de activación de los sensores y la velocidad de las ruedas mientras el robot real estaba moviendose.

Un segundo problema que debería ser tomado en cuenta en la construcción de un simulador es que en robots reales *los sensores físicos entregan valores inciertos y los mandos a los actuadores tienen efectos inciertos.* Los sensores no regresan valores aproximados sino sólo una aproximación difusa de lo que están midiendo. De manera similar, los actuadores tienen efectos que deberían variar en el tiempo dependiendo de muchas causas impredecibles. Este problema puede ser aliviado introduciendo ruido a la simulación en todos los niveles. La adición de ruido a la simulación en los valores regresados por los sensores y en los efectos de los actuadores reduce el descenso en el desempeño observado cuando los individuos son transferidos desde la simulación al ambiente real. El ruido puede ser introducido agregando aleatoriamente valores

seleccionados dentro de un cierto rango a los valores calculados por los sensores o a los efectos de los actuadores. Esto puede llevarse a cabo en diferentes maneras: agregando valores aleatorios dentro de un cierto rango en cada ciclo de vida; alterando los valores de una manera aleatoria pero sistemática para un cierto número de ciclos de vida; variando aleatoriamente ciertos valores en diferentes épocas. Agregar ruido en cada ciclo de vida puede compensar las propiedades estocásticas de la interacción robot-ambiente, pero debe ser agregado en la cantidad apropiada. Se ha demostrado, de hecho, que los controladores neuronales evolucionados dependen del ruido. Como una consecuencia, si la cantidad de ruido en la simulación es más grande que la presente en la interacción real robot-ambiente, los individuos evolucionados se desempeñarán probremente cuando se transfieran al ambiente real por la falta de ruido extra.

Finalmente, ya que la evolución puede explotar características imprevistas de la interacción robot-ambiente, *el cuerpo del robot y las características del ambiente deberían ser reproducidas aproximadamente en simulación.* Esto significa, por ejemplo, que mundos cuadriculados, que a menudo son utilizados en simulaciones de vida artificial, tienen menos significado para el propósito de desarrollar controladores de robot porque son inexactos.

II.9.4 Simulaciones mínimas

Recientemente Jakobi , 1997, propuso un enfoque alternativo basado en el intento de modelar sólo aquellas características de la interacción robot-ambiente que son relevantes para que emerja un comportamiento deseado. Estas características, que son referidas como *conjunto base*, deberían estar aproximadamente modeladas en simulación. En suma a estas características, las simulaciones incluirán otros aspectos que no tendrán ninguna correspondencia en la realidad. Estos aspectos adicionales, los cuales son referidos como *aspectos de implementación*, deberían ser aleatoriamente

variados en diferentes épocas con el objetivo de asegurar que los individuos evolucionados dependerán de los aspectos del conjunto base solamente. Finalmente, el conjunto base debería también variarse de época en época para permitir que emerjan individuos robustos capaces de lidiar con cierta cantidad de variación en el ambiente. Como se reportó en Jakobi , 1997, la métodología completa consiste en los siguientes pasos:

1. *Un conjunto base de interacciones robot-ambiente (que son suficientes para subrayar el comportamiento que queremos evolucionar) debe ser identificado y hecho explicito. Una simulación desplegará aspectos del conjunto base, los cuales tienen su base en la realidad, y aspectos de implementación, que no tienen su base en la realidad.*

2. *Cada aspecto de implementación de la simulación debe ser aleatoriamente variado de prueba en prueba de modo que los controladores evolucionados que dependen de ellos sean poco fiables. En particular, una variación suficiente debe ser incluida para asegurar que los controladores evolucionados no puedan, en la práctica, ser confiablemente capaces si no son del conjunto base exclusivamente (i.e., Ignoran activamente cada aspecto de implementación enteramente).*

3. *Cada aspecto del conjunto base de la simulación debe ser variado aleatoriamente de prueba en prueba de modo que los controladores confiablemente capaces son forzados a ser robustos en el conjunto base. La extención y el carácter de ésta variación aleatoria deben ser suficientes para asegurar que los controladores confiablemente capaces sean aptos para lidiar con las inevitables diferencias entre los aspectos del conjunto base y la realidad, pero no tan grande para que los controladores confiablemente capaces fallen del todo al evolucionar.*

Como un ejemplo consideremos el caso reportado en Jakobi , 1997, de un robot Khepera que es posicionado en un ambiente en forma de “T” con dos fuentes de luz

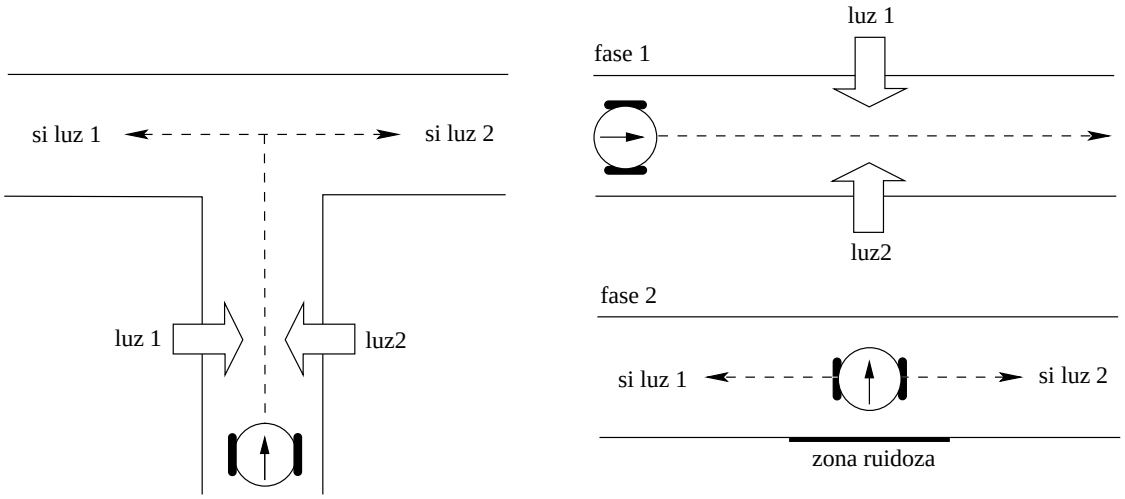


Figura 3: **Izquierda:** El ambiente con forma de "T". Las líneas rectas representan las paredes del ambiente, las flechas grandes representan las dos luces en los dos lados del corredor principal, el círculo representa al robot Khepera, las líneas punteadas representan las dos trayectorias esperadas dependiendo de cual luz se encienda. **Derecha:** La simulación minima del ambiente en "T" mostrado. La fase 1 y 2 representan las dos partes del ambiente, el corredor principal y los dos brazos, respectivamente. La zona ruidosa representa el area donde los valores de activación de los sensores infrarojos son variados aleatoriamente.

que se intercalan. Se espera que el robot voltee a la izquierda o derecha en la unión dependiendo del lado donde la luz esté encendida. Las características del conjunto base en éste experimento son los movimientos de las ruedas y las activaciones de los sensores de proximidad y luz ambiente mientras el robot está en el corredor principal, en donde estas características han sido modeladas aproximadamente en la simulación. En su lugar, la activación de los sensores de proximidad durante la negociación en la unión (i.e., la parte que es más difícil de simular aproximadamente) fue considerada como un aspecto de implementación. Como consecuencia, esta parte de la interacción robot-ambiente no fue modelada en absoluto en simulación.

En la práctica el ambiente "T" fue modelado como dos segmentos (figura 3). Cuando el robot simulado alcanza el final del corredor principal (fase 1), se mueve de allí, rotando 90°, y posicionado en medio del corredor horizontal (fase 2). En éste punto, una mayor discrepancia entre el ambiente simulado y el real surge (en ésta posición el robot real tiene una complicada unión detras de él mientras el robot simulado tiene un muro). Esto se debe al hecho de que, como se mencionó anteriormente, la negociación de la unión se trata como un aspecto de implementación y no está modelado en absoluto en la simulación. Sin embargo, para prevenir la posibilidad de que individuos evolucionados exploten el estado de los sensores infrarrojos orientados hacia esta área (i.e., la zona ruidoza en la figura 3 derecha), los valores de los sensores afectados por esta porción de la pared fueron aleatoriamente variados de prueba en prueba. Finalmente, para asegurar la evolución de individuos robustos, otros aspectos de la simulación tal como el lado con la luz encendida, el ancho del corredor, la orientación inicial del robot, el largo de los corredores, y el largo de la sección iluminada del corredor fue aleatoriamente variada en diferentes épocas a ciertos intervalos determinados.

Jakobi mostró que los controladores evolucionados en esta simulación mínima continuaron desempeñandose satisfactoriamente cuando se transfirieron a un ambiente real. En adición a la tarea del ambiente en "T", el autor fue capaz de resolver una tarea de discriminación visual. El problema con éste enfoque es que se deben conocer por anticipado las interacciones robot-ambiente que son cruciales para producir el comportamiento deseado (i.e., las características del conjunto base). En general, sin embargo, éste no es el caso. Considerando el caso de la tarea del ambiente en "T". Jakobi , 1997, hipotetizó que para resolver ésta tarea, los controladores "deben involucrar una dependencia en algún estado interno o estado externo (menos probable en éste caso) lo cual les permite recordar en cual lado estába la lámpara encendida para que puedan tomar la dirección correcta en la unión". Actualmente, los individuos evolucionados parecen depender de estados internos que mantienen una pista de los estados sensoriales previamente experimentados cuando decidieron a donde voltear en la unión. Sin embargo, se puede imaginar una estrategia más simple consistente en voltear hacia la luz encendida, alcanzando ese lado del corredor, y entonces realizando un comportamiento simple de seguimiento de pared. Esta estrategia puede ser efectiva en el ambiente real, pero no puede ser explotada en un ambiente simulado porque las simplificaciones del modelado introducidas en la unión "T" previenen la posibilidad de realizar un seguimiento de pared al rededor de ésta área. En otras palabras, las interacciones robot-ambiente que deberían ser incluidas en el conjunto base de las simulaciones mínimas dependen de la variedad de estrategias de comportamiento que podrían ser explotadas durante el proceso evolutivo el cual, en cambio, no puede ser conocido de antemano por el diseñador.

II.10 Espacio de aptitud

En la evolución artificial la función de aptitud se utiliza para evaluar el desempeño de los individuos y seleccionar los mejores. El resultado de una corrida evolutiva depende mucho de la forma de ésta función. Las funciones de aptitud para robots autónomos usualmente incluyen variables y restricciones que estiman el desempeño con respecto al comportamiento deseado, pero éstas variables y restricciones son difíciles de elegir porque el comportamiento evolucionado por el robot no es completamente conocido de antemano. Actualmente, el grado de conocimiento de un comportamiento esperado es inversamente proporcional al atractivo de usar evolución artificial. Incluso cuando se conocen de antemano las variables más relevantes y como combinarlas en una función, no está garantizado que el sistema será evolucionable, ya que las variables que miden objetivamente la aptitud de un comportamiento completamente funcional pueden ser incapaces de estimar los aspectos primordiales de ese comportamiento en etapas tempranas de la evolución. Por lo tanto, algunos aspectos del comportamiento del robot que son instrumentos para el desarrollo de mayores habilidades pueden ser subestimados; o, incluso peor, todos los individuos de generaciones tempranas reciben aptitud cero.

La función de aptitud juega un rol mayor en la evolución de un sistema. La falta de principios formales para el diseño de funciones de aptitud de sistemas evolutivos autónomos tiene dos consecuencias mayores: a) los resultados obtenidos incluso con pequeñas diferencias de aptitud son difícilmente comparables; b) la elección de una función apropiada frecuentemente procede de pruebas y errores, siendo un enfoque que consume tiempo cuando se evolucionan robots físicos.

Proponemos el espacio de aptitud como un marco objetivo para describir y comparar varias funciones de aptitud para sistemas autónomos (Figura 4). El espacio de aptitud

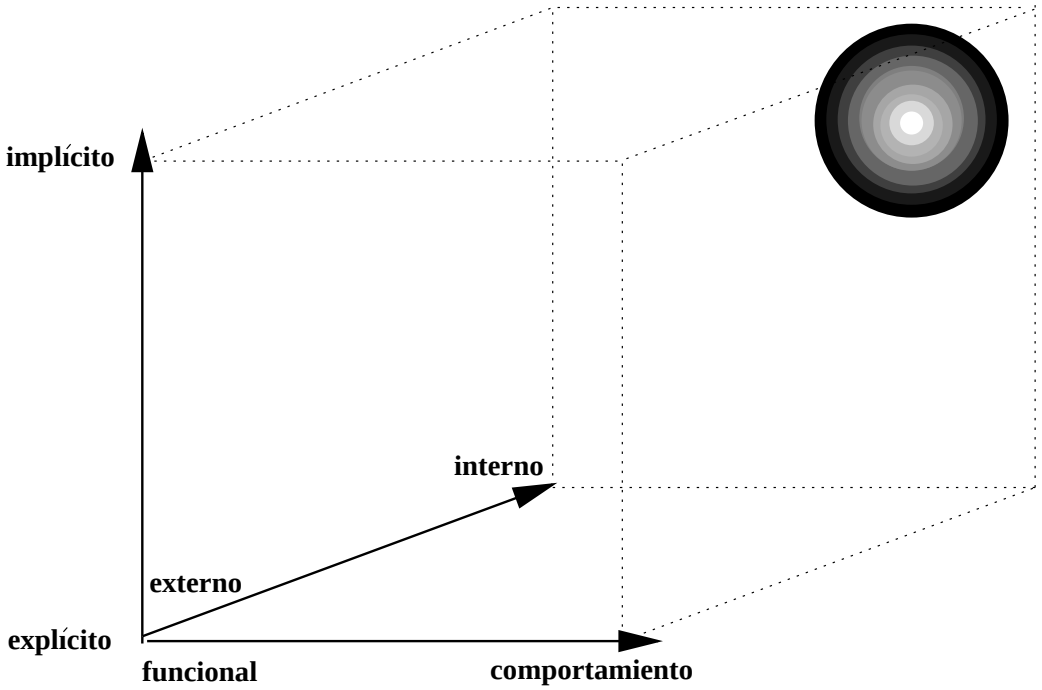


Figura 4: El espacio de aptitud provee un marco para describir y diseñar funciones de aptitud de sistemas autónomos. Las tres dimensiones definen un importante criterio de diseño de aptitud a lo largo de un espacio continuo. La aptitud para robots autónomos está localizada idealmente en la esquina superior derecha.

puede también ayudar a definir líneas generales para diseñar funciones de aptitud y valorar las implicaciones de ciertas elecciones. El espacio de aptitud está definido por tres dimensiones continuas; una función de aptitud representa un punto en este espacio 3D.

- La dimensión *funcional-comportamiento* indica cuando la función de aptitud estima modos especificos de funcionamiento del controlador o cuando estima elcomportamiento resultante del controlador. Por ejemplo, considerese la evolución de una caminata para un robot con piernas y equipado con una red de osciladores neurales. Una función de aptitud estimaría la frecuencia de oscilación de las neuronas en la red para evolucionar controladores cuyas dinámicas corresponden a patrones de caminata. En cambio, una función de comportamiento estimaría la distancia cubierta por el robot dentro de un cierto periodo de tiempo. Las funciones de aptitud pueden incluir una combinación de componentes funcionales y de comportamiento. La dimensión funcional-comportamiento es aplicable sólo cuando hay una diferencia entre el funcionamiento de un sistema evolutivo y su comportamiento, la cual es una característica de agentes autónomos.

- La dimensión *explícito-implícito* define la cantidad de variables y restricciones incluidas en la función. Por ejemplo, una función implícita seleccionaría solamente robots que no se terminaron su energía después de un cierto tiempo. En su lugar, una función explícita incluiría variables y restricciones que evaluan varios componentes del comportamiento deseado, tal como cuando el robot alcanza un área de abastecimiento de energía, la velocidad de viaje, la actividad de ciertas neuronas en el controlador, etc. La posición de una función de aptitud a lo largo de ésta dimensión es dada por el número total de variables, constantes, y restricciones (de tipo si-entonces, e.g.) incluidas en la función, no por su combinación y peso

relativo. Por ejemplo, la función $f = 2x + y$. Ocupa la misma posición que la función $f = x - 3y$. Las funciones de aptitud implícitas no tienen (o muy pocos) componentes.

- La dimensión *externa-interna* indica cuando las variables y restricciones incluidas en la función de aptitud se calculan utilizando información disponible para el agente evolutivo. Notece que incluso si ésta información está disponible para el agente, puede no ser utilizada necesariamente como una entrada al controlador del agente. Por ejemplo, la distancia absoluta entre un robot y una meta es una variable externa porque no puede ser calculada empleando la información que viene de los sensores del robot. En su lugar, las activaciones de los sensores de luz ambiental, o de la carga de batería, son variables internas porque son calculadas utilizando información disponible para el robot que se evoluciona. Si un experimento se lleva a cabo enteramente en simulación, la posición de la función de aptitud a lo largo de ésta dimensión no es muy relevante dado que todos los tipos de información están disponibles en software. Pero si se planea combinar simulación y mundo real, o llevar a cabo una ejecución evolutiva enteramente en un robot físico, las variables externas pueden no ser medibles en el mundo real y pueden requerir la realización de maquinarias complejas y procedimientos para calcularlas exactamente y en tiempo real.

Decidir donde se está en el espacio de aptitud depende de la meta de un corrida evolutiva. Si la meta es optimizar un conjunto de parámetros para un muy complejo - pero bien definido - problema de control en un ambiente bien controlado, entonces la función de aptitud debería ser localizada en la esquina inferior izquierda del espacio de aptitud. Las funciones FEE (Funcional, Explícita, Externa) describen como el controlador debería trabajar, estiman el sistema sobre la base de muchas variables y

restricciones, y emplean artefactos para tomar medidas externas exactas.

En su lugar, si la meta es evolucionar robots capaces de operar autónomamente en ambientes parcialmente desconocidos y impredecibles sin intervención humana, la función de aptitud debería tender hacia la esquina superior derecha del espacio de aptitud. Las funciones BII (Behavioral-de comportamiento, Implícita, Interna) estiman sólo el comportamiento resultante de un controlador evolutivo, el cual depende de algunas - o ninguna - variables y restricciones, y sus componentes pueden ser calculados a bordo. Las funciones BII son apropiadas para robots adaptivos autónomos. Estas funciones dejan al sistema evolutivo libre para elegir y adaptar su propia estrategia a las caracteristicas del ambiente, explotando lo que el ambiente le permite hacer. Ya que las funciones de aptitud BII dependen menos de restricciones impuestas externamente, son apropiadas para evolución abierta incremental donde los organismos evolucionados se adaptan a un ambiente siempre cambiante con el único proposito de sobrevivir.

La línea diagonal entre los extremos de las FEE y BII en el espacio de aptitud define una continuación entre optimización de ingeniería tradicional y síntesis de sistemas de vida artificial, respectivamente. Las posiciones relativas de las proyecciones de las funciones de aptitud en ésta diagonal dan una comparación tosca entre los niveles de autonomía de los sistemas evolucionados.

La *aptitud subjetiva* es el nombre empleado para el caso donde los observadores humanos consideran el desempeño de individuos evolucionados por medio de una inspección visual. Un famoso ejemplo de aptutid subjetiva son los *Biomorfos* descritos por Dawkins , 1986, donde un observador puede seleccionar algunos individuos de una pantalla llena de diferentes criaturas con formas estáticas; los individuos seleccionados, cuyo cromosoma define la forma de las criaturas, son reproducidos y mutados para crear una nueva población de criaturas. Lund *et al.* , 1998, han aplicado ésta estrategia para la evolución de sistemas de control de robot simulados cuyos comportamientos

son desplegados en la pantalla de una computadora. Los observadores humanos son cuestionados sobre cuales individuos son seleccionados para reproducción y mutación. La aptitud subjetiva está usualmente localizada en la base cerca de la esquina (BEE) del espacio de aptitud porque está basada en el comportamiento del robot, depende de restricciones explicitas (aunque no objetivamente o verbalmente expresadas), y las restricciones son externas al sistema evolucionado. La posición a lo largo de la dimensión explícita - implícita puede variar dependiendo de como un sujeto estima un robot. En algunos casos el sujeto es cuestionado sobre un valor numérico para un conjunto de habilidades predefinidas, mientras que en otros casos los sujetos son libres de decidir sobre cual criterio desean. Puede suceder que la cantidad y el tipo de restricciones varía de un sujeto a otro e incluso para el mismo sujeto durante el tiempo evolutivo. Las funciones de aptitud subjetivas pueden ser útiles cuando es difícil diseñar una función objetivo o en situaciones donde es deseable tener una máquina que se adapte interactivamente con un sujeto humano (por ejemplo en entretenimiento y servicio robótico). Sin embargo, si el sujeto es inexacto e inconsistente puede ser difícil obtener resultados significativos.

Capítulo III

Redes neuronales artificiales

III.1 Introducción

Las redes neuronales artificiales, o simplemente redes neuronales (RN), han sido estudiadas por más de tres decadas desde que Rosenblatt aplicó primero un perceptron de una sola capa al aprendizaje de clasificación de patrones en la decada de 1950, Rosenblatt , 1958. Sin embargo, debido a que Minsky y Papert , 1969, apuntaron que los sistemas de una sola capa eran limitados y expresarón pesimismo sobre los sistemas multicapa, el interés en las RN disminuyó en la decada de 1970. La reciente resurgencia de interés en el campo de RN ha sido inspirada por nuevos desarrollos en algoritmos de aprendizaje de RN, circuitos integrados analógos de muy grande escala, y técnicas en procesamiento paralelo.

Bastantes modelos de RN han sido propuestos e investigados en años recientes. Estos modelos de RN pueden ser clasificados de acuerdo a varios criterios, tales como sus métodos de aprendizaje (supervisado contra no supervisado), arquitecturas (hacia delante contra recursivas), tipos de salida (binario contra continuo), tipo de nodos (uniformes contra híbridos), implementaciones (software contra hardware), peso de las conexiones (ajustables contra conectados fuerte), operaciones (motivadas biologicamente contra motivadas sicológicamente), y más.

III.2 Perceptrones

III.2.1 Arquitectura y regla de aprendizaje

El *perceptron* representa uno de los primeros intentos de construir sistemas inteligentes y que aprendieran usando componentes simples. Fue obtenido de un modelo de una neurona del cerebro biológico introducido por McCulloch y Pitts , 1943. Más tarde, Rosenblatt , 1958 diseñó el perceptron con vistas a explicar y modelar las habilidades de reconocimiento de patrones de los sistemas visuales biológicos. Aunque la meta es ambiciosa, el paradigma de la red es simple. La figura 5 es un montaje de un perceptron típico para aplicaciones de reconocimiento de patrones, en el cual los patrones visuales están representados como matrices de elementos entre 0 y 1. La primer capa del perceptron actúa como un conjunto de "detectores de características" que están conectados directamente a las señales de entrada para detectar características específicas. La segunda capa toma las salidas de los "detectores de características" en la primer capa y clasifica el patrón de entrada dado. El aprendizaje es iniciado haciendo ajustes a las fuerzas de las conexiones relevantes (i.e., los pesos w_i) y un valor umbral θ. Para un problema de dos clases (por ejemplo, determinar cuando el patrón dado en la figura 5 es "P" o no), la capa de salida usualmente tiene sólo un nodo. Para un problema de n-clases con n mayor que, o igual a 3, la capa de salida usualmente tiene n nodos, donde cada uno corresponde a una clase, y el nodo de salida con el más largo valor indica a cual clase pertenece el vector de entrada.

Cada función g_i en la capa 1 es una función específica que tiene que ser determinada *a priori*: transforma todo o parte del patrón de entrada en un valor binario $x_i \in$ {-1, 1} o un valor bipolar $x_i \in \{0, 1\}$. Al término x_i se le refiere como *activo* o *excitatorio* si su valor es 1, *inactivo* si su valor es 0, e *inhibitorio* si su valor es -1. La unidad de salida es un *elemento umbral líneal* con un valor umbral θ:

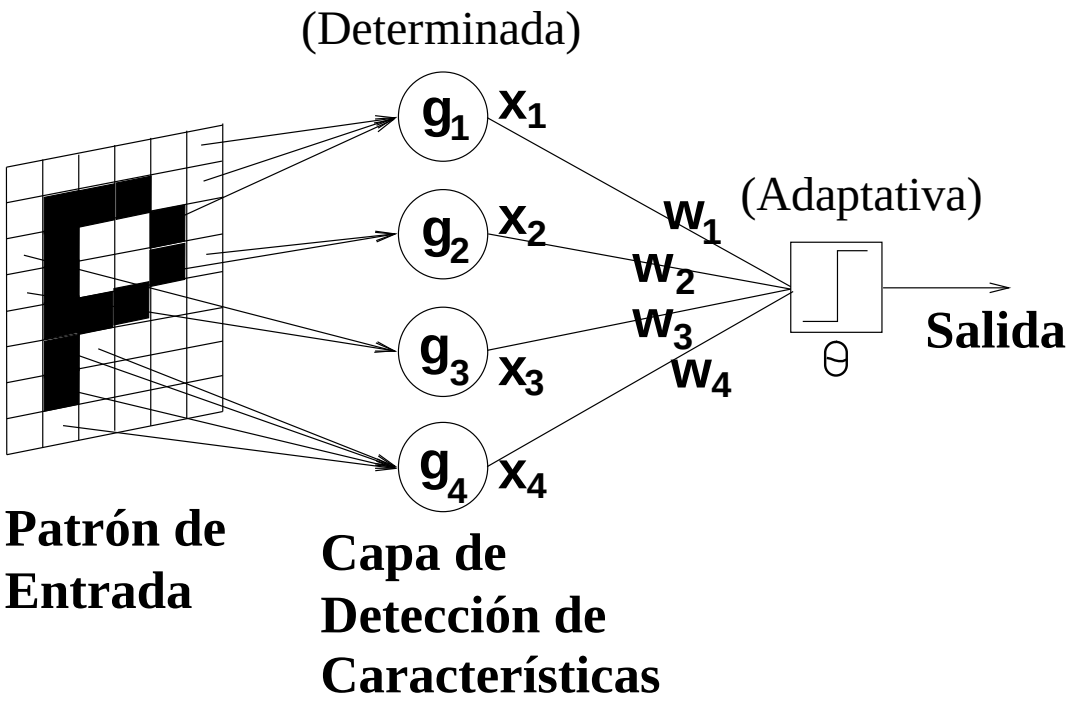


Figura 5: El perceptron.

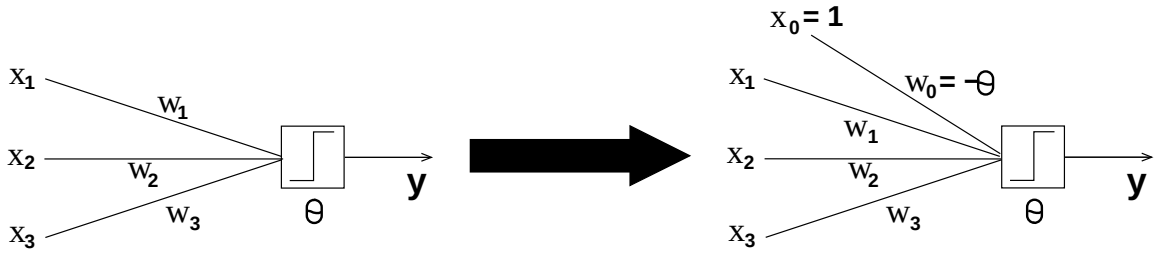


Figura 6: Introducción del peso de la conexión de predisposición; el término de predisposición $w_0(= -\theta)$ puede ser visto como el peso de la conexión entre la unidad de salida y una señal "de prueba" entrante x_0 que siempre es igual a 1.

$$\begin{aligned} y &= f(\textstyle\sum_{i=1}^{n} w_i \, x_i \text{ - } \theta), \\ &= f(\textstyle\sum_{i=1}^{n} w_i \, x_i + w_0), \; w_0 \equiv \text{ - } \theta, \\ &= f(\textstyle\sum_{i=1}^{n} w_i \, x_i), \; x_0 \equiv 1. \end{aligned} \tag{1}$$

donde w_i es un peso modificable asociado con un señal de entrada x_i: y w_0 (= -θ) es el término de predisposición. Para mejorar la eficiencia computacional, se puede introducir el peso de la conexión de predisposición w_0 en lugar del valor umbral θ. La ecuación (1) muestra que el umbral puede ser visto como el peso de la conexión entre la unidad de salida y una señal "de prueba" de entrada x_0 que es siempre igual a 1, como está ilustrado en la Figura 6. En la Ecuación (1) $f(\cdot)$ es la *función de activación* del perceptron y típicamente es una función sigmoide o función escalón:

$$sgn(x) = \begin{cases} 1 & \text{si} \quad x > 0 \\ -1 & \text{si} \quad otro \end{cases} \tag{2}$$

$$step(x) = \begin{cases} 1 & \text{si} \quad x > 0 \\ 0 & \text{si} \quad otro \end{cases} \tag{3}$$

Note que el "detector de características" g_i puede ser cualquier función del patrón de entrada, pero el procedimiento de aprendizaje sólo ajusta los pesos de la conexión a la unidad de salida (en la última capa). Ya que sólo los pesos que se dirigen a la última capa son modificables, el perceptron es tratado usualmente como un perceptron de una sola capa. Comenzando con un conjunto de pesos de conexión aleatorio, el algoritmo de aprendizaje básico para un perceptron de una sóla capa repite los siguientes pasos hasta que los pesos converjan:

1. Seleccionar un vector de entrada $\mathbf{x}$ de un conjunto de datos de entrenamiento.

2. Si el perceptron da una respuesta incorrecta, modificar todos los pesos de conexión w_i de acuerdo a

$$\Delta w_i = \eta t_i x_i,$$

donde t_i es una salida objetivo y η es una tasa de aprendizaje.

La regla de aprendizaje anterior puede aplicadarse también para actualizar un umbral θ (= -w_0) de acuerdo a la ecuación. El valor para la tasa de aprendizaje η puede ser una constante a través del entrenamiento: o puede ser una variable proporcional al error. Una η que es proporcional al error usualmente dirije a una más rapida convergencia pero puede causar un aprendizaje inestable.

El algoritmo de aprendizaje anterior está fuertemente basado en el descenso del gradiente; Rosenblatt , 1962, probó que existe un método para adaptar los pesos. Esto garantiza que converja el algoritmo, a fin de proveer la salida requerida si y sólo si tal conjunto de pesos existe. Esto es llamado el *teorema de convergencia del perceptron*. Además dependiendo de las funciones g_i, los perceptrones pueden ser agrupados en diferentes familias.

En los comienzos de la decada de 1960, los preceptrones dieron importancia al interés y optimismo dirigido hacia la contrucción de sistemas reales inteligentes y con autoaprendizaje. Sin embargo, el entusiasmo inicial ganado después de la publicación sobre perceptrones de Minsky y Papert , 1969, en la cual ellos analizan al perceptron extensamente y concluyen que los perceptrones de una sola capa pueden sólo ser utilizados para problemas de juguete. Uno de sus más desalentadores resultados muestran que un perceptron de una sola capa no puede representar una simple función de OR exclusivo, como se explica a continuación.

III.2.2 Problema del OR exclusivo

El más simple y más bien conocido problema de reconocimiento de patrones en la literatura de redes neuronales es el problema del OR exclusivo (XOR). La tarea es

clasificar un vector de entrada binario a la clase 0 si el vector tiene un número impar de 1's. De otra forma asignarlo a la clase 1. Para un problema XOR de dos entradas binarias, el comportamiento deseado es regulado por una tabla de verdad dada en la Tabla II:

Tabla II: Tabla de verdad del problema XOR.

	X	Y	Clase
Par de E/S deseado 1	0	0	0
Par de E/S deseado 2	0	1	1
Par de E/S deseado 3	1	0	1
Par de E/S deseado 4	1	1	0

Un problema XOR bipolar se define de manera similar excepto que todas las instancias de 1 en la tabla de verdad son reemplazados por -1.

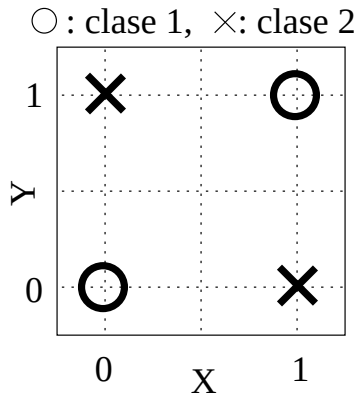


Figura 7: Problema XOR.

El problema XOR no es linealmente separable; como se puede observar en la Figura 7. En otras palabras, no es posible utilizar un perceptron de una sola capa, ver Figura 8(a), para construir una línea recta a fin de particionar el espacio de entrada bidimensional en dos regiones, cada una conteniendo sólo datos de la misma clase. Simbólicamente, usando un perceptron de una sola capa para resolver este problema requiere satisfacer las siguientes cuatro desigualdades:

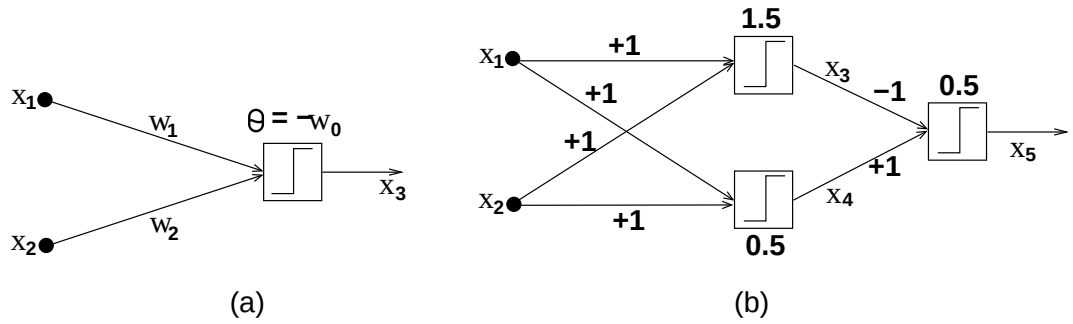


Figura 8: Perceptrones para el problema OR-exclusivo de dos estradas: (a) perceptron de una sola capa, y (b) perceptron de dos capas. Ambos usan la función escalón como función de activación para cada nodo.

$$\begin{array}{rcrcrcrcl} 0 \times w_1 & + 0 & \times & w_2 & + w_0 & \leq 0 & \Longleftrightarrow & w_0 & \leq 0, \\ 0 \times w_1 & + 1 & \times & w_2 & + w_0 & > 0 & \Longleftrightarrow & w_0 & > \text{-}w_2, \\ 1 \times w_1 & + 0 & \times & w_2 & + w_0 & > 0 & \Longleftrightarrow & w_0 & > \text{-}w_1, \\ 1 \times w_1 & + 1 & \times & w_2 & + w_0 & \leq 0 & \Longleftrightarrow & w_0 & \leq \text{-}w_1 \text{-}w_2. \end{array} \tag{4}$$

Sin embargo, el conjunto de desigualdades se contradice solo cuando es considerado como un todo. Afortunadamente, es posible resolver el problema con el perceptron de dos capas ilustrado en la Figura 8(b), en el cual los pesos de conexión y los umbrales están indicados. En resumen, los perceptrones multicapa pueden resolver problemas no separables linealmente y son entonces más poderosos que la version original de una sola capa.

III.3 ADALINE

El *elemento lineal adaptativo* (o Adaline), sugerido por Widrow y Hoff , 1960, representa un ejemplo clásico del más simple sistema inteligente con auto-aprendizaje que puede adaptarse por si mismo para alcanzar una tarea de modelado dada (e.g., filtrado de señales, controladores.) La Figura 9 es un diagrama esquematico de dicha red. Tiene una unidad de salida puramente lineal: por eso la salida y de la red es una

combinación lineal pesada de las entradas mas un término constante:

$$y = \sum_{i=1}^{n} w_i x_i + w_0. \tag{5}$$

En una implementación física simple, las señales de entrada x_i son voltajes y los w_i son conductancias de resistores controlables. La salida de la red es la suma de las corrientes causada por la entrada de voltajes. El problema es encontrar un conjunto adecuado de conductancias (o pesos) tal que el comportamiento entrada-salida de la Adaline sea cerrado a un conjunto de entrada-salida de datos deseado.

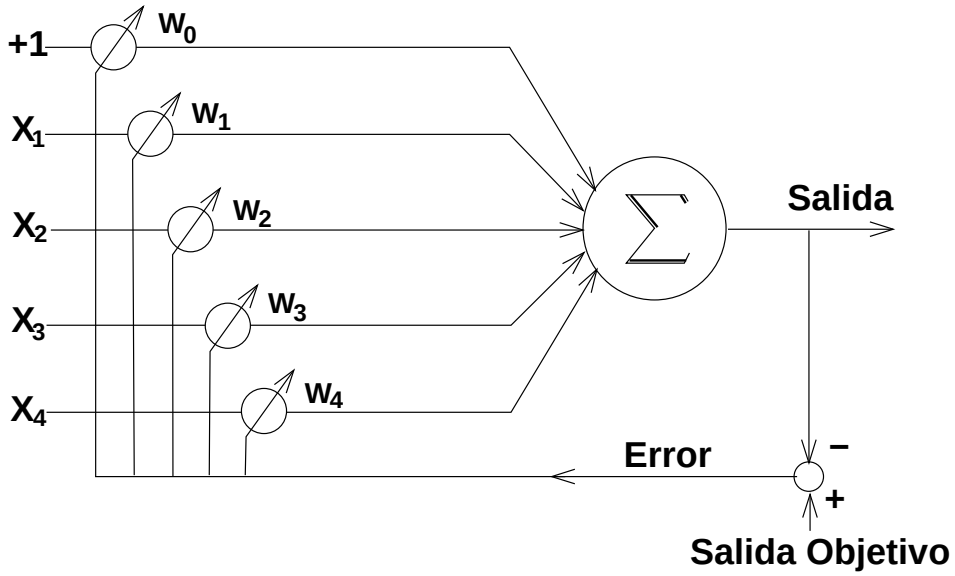


Figura 9: ADALINE (Elemento Lineal Adaptativo).

Es obvio que la ecuación (5) es exactamente un modelo lineal con n + 1 parámetros lineales, de manera que podemos emplear los métodos de mínimos cuadrados para minimizar el error en el sentido de mínimos cuadrados. No obstante, la mayoría de los métodos de mínimos cuadrados requieren cálculos extensos. Para superar ésto Widrow y Hoff , 1960, introdujeron la *regla delta* para ajustar los pesos. Para el p-esimo patrón de entrada-salida, el error medible de una sola salida Adeline puede se expresado como

$$E_p = (t_p - y_p)^2, \tag{6}$$

donde t_p es la salida objetivo y y_p es la salida actual de la Adeline. La derivada parcial de E_p con respecto a cada peso w_i es

$$\frac{\partial E_p}{\partial w_i} = -2(t_p - y_p)x_i. \tag{7}$$

Por lo tanto, para disminuir E_i utilizando el descenso de gradiente, la fórmula de actualización de w_i del p-esimo patrón de entrada-salida es

$$\Delta_p w_i = \eta(t_p - y_p)x_i. \tag{8}$$

Esta fórmula de actualización tiene un fuerte atractivo intuitivo. Esencialmente se plantea que cuando $t_p > y_p$, queremos estímular a y_p incrementando $w_i x_i$; por lo tanto, se debería incrementar w_i si x_i es positivo y decrementar w_i si x_i es negativo. Un razonamiento similar se aplica cuando $t_p < y_p$. Ya que la regla delta trata de minimizar errores cuadrados, es también referida como el *procedimiento de aprendizaje de la mínima media cuadrada* o *regla de aprendizaje de Widrow-Hoff*. Las características de la regla delta son las siguientes:

- Simplicidad: Esto es obvio en la Ecuación (8), sólo se ocupan hacer tres operaciones básicas sencillas.
- Aprendizaje distribuido: El aprendizaje no es dependiente del control central de la red; puede ser realizado localmente a nivel de cada nodo; ver Ecuación (8)
- Aprendizaje en línea (o patrón por patrón): Los pesos son actualizados después de que se presenta cada patrón.

Estas características hacen a Adeline, junto con la regla delta un buen candidato para una implementación simple en hardware.

En la decada de 1960, dos o más componentes Adeline fueron integrados para desarrollar *Madeline* (muchos Adelines) en un intento de implementar funciones lógicas separables no linealmente. Para resolver el problema anterior del XOR, dos Adelines fueron conectados a un dispositivo lógico AND (unidad Madeline) para proveer una salida. No obstante, los sistemas Adeline y Madeline fueron limitados en que sólo tienen una capa con pesos ajustables, tal como los perceptrones de una sola capa.

Adeline y Madeline han sido usados para la cancelación de ruido adaptativo y control inverso adaptativo. En la cancelación del ruido adaptativo, el objetivo es filtrar un componente de interferencia identificando un modelo lineal de una fuente de ruido medible y la correspondiente no medible interferencia: tales aplicaciones incluyen la cancelación de la interferencia en electrocardiogramas, y eliminación de eco de las líneas de transmisión de teléfono en larga distancia.

III.4 Retropropagación de perceptrones multicapa

Como se mencionó anteriormente, la falta de métodos adecuados de entrenamiento para *perceptrones multicapa* (MLP, Multilayer perceptron) provoca una disminución del interés en redes neuronales en los 60's y 70's. Ésto no fue cambiado hasta la reformulación del método de entrenamiento de *retropropagación* para MLP's a mediados de los 80's por Rumelhart *et al.* , 1986.

El perceptron de una sola capa discutido anteriormente es un componente principal de una RN y provee el terreno para el entendimiento actual y para más aplicaciones de RN's. Sin embargo, por la no diferenciación de la función de activación de tipo "supresor de picos", las estrategias de aprendizaje de los primeros perceptrones multicapa

con funciones de activación sigmoidales o de escalón no son obvias aunque funciones de activación contínuas sean utilizadas.

Una retropropagación de MLP, es una red adaptativa cuyos nodos (o neuronas) realizan la misma función de señales entrantes. Esta función del nodo es usualmente una composición de la suma pesada y una función de activación no lineal diferenciable, también conocida como *función de transferencia*. La Figura 10 representa tres de las más utilizadas funciones de activación en la retropropagación de MLP's:

$$\begin{array}{ll} \text{Función logística:} & f(x) = \frac{1}{1+e^{-x}} \\ \text{Función tangente hiperbólica:} & f(x) = \tanh(\text{x}/2) = \frac{1-e^{-x}}{1+e^{-x}} \\ \text{Función identidad:} & f(x) = \text{x}. \end{array} \tag{9}$$

las funciones tangente hiperbólica y logística aproximan las funciones sigmoide y la escalón, respectivamente. Estas funciones proveen derivadas suaves, diferentes de cero con respecto a las señales de entrada. Algunas veces dos funciones de activación son referidas como *funciones de agrupamiento* ya que las entradas de estas funciones están agrupadas en el rango [0,1] o [-1,1]. También son llamadas *funciones sigmoide* porque su forma de curvas en S exhiben suavidad y propiedades asintóticas. (Algunas veces la función tangente hiperbólica está referida como una sigmoidal bipolar y la función logística es referida como una sigmoidal binaria.) Ambas activaciones son empleadas a menudo en los problemas de regresión y clasificación.

Para una red neuronal aproximar una función de valores contínuos no limitada en el intervalo [0, 1] o [-1, 1], dejamos usualmente a la función nodo para la capa de salida ser una suma pesada sin funciones de agrupamiento. Esto es equivalente a la situación en la cual la función de activación es una función identidad, y los nodos de salida de éste tipo son a menudo llamados *nodos lineales*.

Las MLP's de retropropagación son las estructuras de RN más comunmente usadas

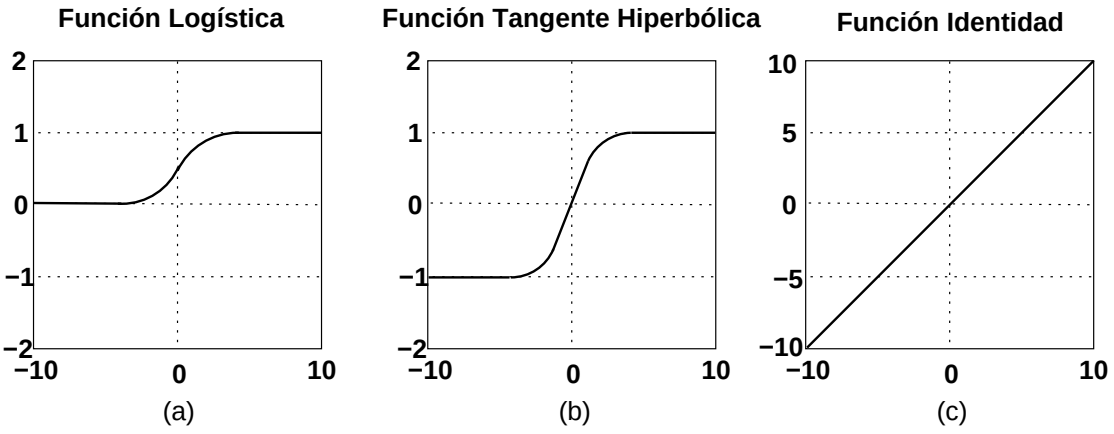


Figura 10: Funciones de activación de retropropagación de MLP's: (a) función logística; (b) función tangente hiperbólica; (c) función identidad.

en aplicaciónes con un amplio rango de áreas, tales como reconocimiento de patrones, procesamiento de señales, compresión de datos, y control automático. Algunas de las más conocidas instancias de aplicaciones incluyen NETtalk, la cual es un MLP bien entrenado para pronunciar texto en inglés, ALVINN (Autonomous Land Vehicle in a Neural Network) de la Universidad Carnegie Mellon, el cual usó un MLP para direccionar un vehículo autónomo; y reconocimiento óptico de caracteres.

III.4.1 Regla de aprendizaje de retropropagación

Por simplicidad, asumimos que los MLP de retropropagación en cuestión utilizan la función logística como una función de activación: el lector está interesado en derivar el procedimiento de retropropagación cuando otros tipos de funciones de activación son utilizadas.

La entrada de la red $\bar{x}$ de un nodo está definido como la suma pesada de las señales entrantes mas un término de predisposición. Por ejemplo, la entrada y la salida de la red en el nodo j en la Figura 11 son:

$$\bar{x}_j = \sum_i w_{ij} x_i + w_j,$$

$$x_j = f(\bar{x}_j = \frac{1}{1 + exp(-\bar{x}_j)}), \tag{10}$$

donde x_i es la salida del nodo i localizado en cualquiera de las capas anteriores, w_{ij} es el peso asociado a la liga que conecta los nodos i y j, y w_j es la predisposición del nodo j. Ya que los pesos w_{ij} son actualmente parámetros internos asociados con cada nodo j, cambiar los pesos de un nodo alterará el comportamiento del nodo y en consecuencia alterará el comportamiento de todo el MLP de retropropagación completo. La Figura 12 muestra un MLP de retropropagación de dos capas con tres entradas en la capa de entrada, tres neuronas en la capa escondida, y dos neuronas de salida en la capa de salida. Por simplicidad, éste MLP de retropropagación será referido como un red 3-3-2, correspondiente al número de nodos en cada capa. (Note que la capa de entrada está compuesta de tres nodos como memorias intermedias para distribuir las señales de entrada; por lo tanto, ésta capa convencionalmente no cuenta como una capa física de un MLP de retropropagación.)

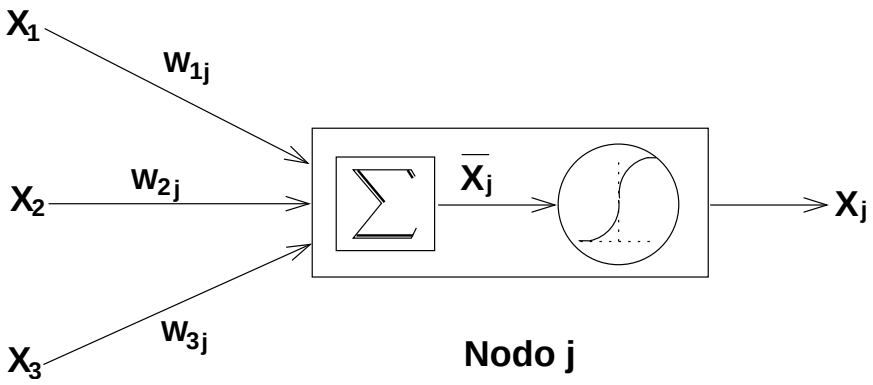


Figura 11: Nodo j de una MLP de retropropagación.

El *error de propagación hacia atrás*, también conocido como la *retropropagación*

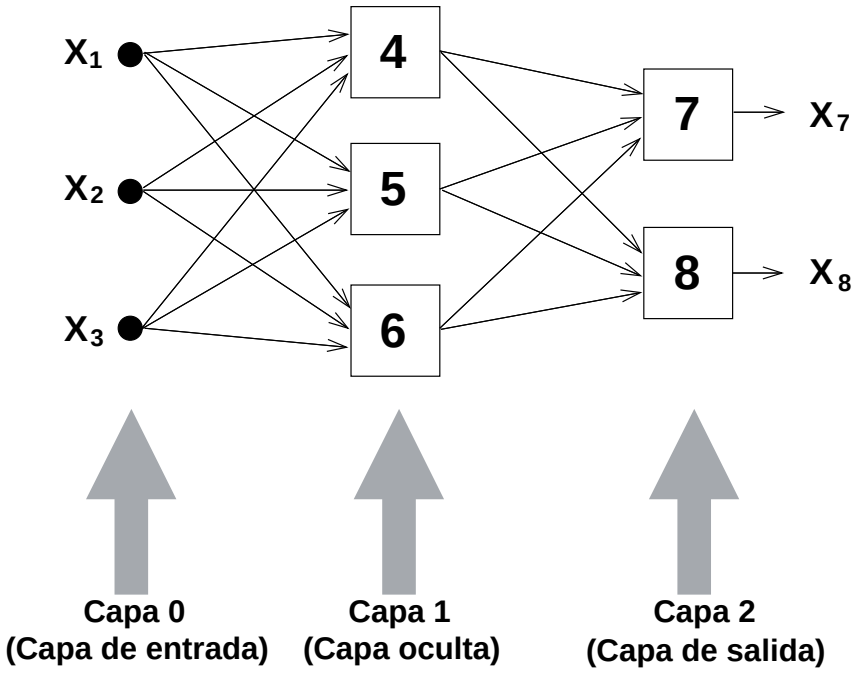


Figura 12: MLP de retropropagación 3-3-2.

(BP, backpropagation) o *regla delta generalizada* (GDR, generalized delta rule) se explica a continuación. Primero, un error cuadrado medido para el p-ésimo par entrada-salida se define como:

$$E_p = \sum_k (d_k - x_k)^2, \tag{11}$$

donde d_k es la salida deseada para el nodo k, y x_k es la salida actual para el nodo k cuando la parte de entrada del par de datos p-esimo es presentado. Para encontrar el vector gradiente, un término de error $\bar{\epsilon}_i$ para el nodo i se define como:

$$\bar{\epsilon}_i = \frac{\partial^+ E_p}{\partial \bar{x}_i}. \tag{12}$$

Por la regla de la cadena, la fórmula para $\bar{\epsilon}_i$ puede escribírse como

$$\bar{\epsilon}_i = \begin{cases} -2(d_i - x_i)\frac{\partial x_i}{\partial \bar{x}_i} = -2(d_i - x_i)x_i(1 - x_i) & \text{si el nodo i es un nodo de salida} \\ \frac{\partial x_i}{\partial \bar{x}_i} = \sum_{j,i<j} \frac{\partial^+ E_p}{\partial \bar{x}_j}\frac{\partial \bar{x}_j}{\partial x_i} = x_i(1 - x_i)\sum_{j,i<j} \bar{\epsilon}_j w_{ij} & \text{otra forma,} \end{cases} \tag{13}$$

donde w_{ij} es el peso de la conexión del nodo i al j; y w_{ij} es cero si no hay conexión directa. Entonces la actualización del peso w_{ki} para aprendizaje en línea (patrón por patrón) es:

$$\Delta w_{ki} = -\eta \frac{\partial^+ E_p}{\partial w_{ki}} = -\eta \frac{\partial^+ E_p}{\partial \bar{x}_i}\frac{\partial \bar{x}_i}{\partial w_{ki}} = -\eta \bar{\epsilon}_i x_k, \tag{14}$$

donde η es una tasa de aprendizaje que afecta la velocidad de convergencia y la estabilidad de los pesos durante el aprendizaje. La fórmula de actualización para la predisposición de cada nodo puede ser derivada similarmente.

Para aprendizaje fuera de línea (por lote), el peso de conexión w_{ki} se actualiza sólo después de la presentación del conjunto de datos entero, o sólo después de una *época*:

$$\Delta w_{ki} = -\eta \frac{\partial^+ E}{\partial w_{ki}} = -\eta \frac{\partial^+ E_p}{\partial w_{ki}}, \tag{15}$$

o, en la forma de vector,

$$\Delta \mathrm{w} = -\eta \frac{\partial^+ E}{\partial \mathrm{w}} = -\eta \nabla_{\mathrm{w}} E, \tag{16}$$

donde $E = \sum_p E_p$. Esto corresponde a una forma de utilizar la verdadera dirección de gradiente basado en el conjunto entero de datos.

III.4.2 Métodos de aceleración en el entrenamiento de MLP

Hoy en día existen bastantes métodos ad hoc para acelerar el entrenamiento de MLP's de retropropagación. Algunos de ellos son aplicables a la retropropagación general del descenso de gradiente, mientras otros son sólo efectivos para MLP's de retropropagación.

Una forma de acelerar el entrenamiento fuera de línea es emplear el tan llamado *término momentum*:

$$\Delta \mathrm{w} = -\eta \nabla_{\mathrm{w}} E + \alpha \Delta \mathrm{w}_{\mathrm{prev}}, \tag{17}$$

donde $\mathrm{w}_{\mathrm{prev}}$ es la cantidad de actualización previa, y la *constante momentum* α, en la práctica, es usualmente establecido entre 0.1 y 1. La suma del término momentum suaviza la actualización de los pesos y tiende a resistir cambios erráticos en los pesos debido al ruido del gradiente o a frecuencias espaciales altas en la superficie de error. Sin embargo, el empleo de los términos momentum no siempre parece acelerar el entrenamiento; es más o menos dependiente de la aplicación.

Otra técnica útil es normalizar la actualización del peso como sigue:

$$\Delta \mathrm{w} = -\kappa \frac{\nabla_{\mathrm{w}} E}{\parallel \nabla_{\mathrm{w}} E \parallel}. \tag{18}$$

Esto causa que el vector de pesos de la red se mueva en la misma dirección utilizando la distancia Euclideana κ en el espacio de pesos con cada actualización, lo cual permite el control de la distancia κ basados en la historia de los errores medidos. Otros métodos para acelerar el entrenamiento de los MLP's de retropropagación incluye el algoritmo de retropropagación rápido, el enfoque delta-bar-delta, el método de filtro de Kalman extendido, optimización de segundo orden, y el enfoque de filtrado óptimo.

Generalmente, un MLP con funciones tangentes hiperbólicas puede ser entrenado

más rapidamente que uno con funciones logísticas. Este es sólo una observación empírica y tiene excepciones, pero es aconsejable intentar ambos tipos de MLP cuando se encuentre una nueva aplicación.

Algunas veces es deseable hacer un *escalamiento de datos* en los datos crudos de entrenamiento y entonces utilizar los datos procesados para entrenar el MLP. En el *escalamiento de salida*, el rango de los valores objetivo está restringido a permanecer dentro del rango de activación de la función sigmoide. Por ejemplo, para un MLP con funciones tangente hiperbólica, los valores objetivo deben estár dentro del rango [-0.9, 0.9], en lugar del rango usual de la función de activación [-1, 1]. Esto impide a la retropropagación dirigir alguno de los pesos de conexión al infinito y desacelerar el entrenamiento. En el *escalamiento de entrada*, el rango de cada entrada es escalado (usualmente de forma lineal) al rango de la función de activación usada. Este escalamiento permite a los pesos de la conexiones tener el mismo orden de magnitud durante el entrenamiento.

Los valores iniciales de los pesos de conexión y predisposición en un MLP deberían ser uniformemente distribuidos a lo largo de un rango pequeño, generalmente [-1, 1]. Si los valores iniciales de estos parámetros modificables son muy largos, entonces algunas de las neuronas deberían saturarse y producir señales de error pequeñas. Por otro lado, si el rango es muy pequeño, entonces el vector gradiente es también pequeño y el aprendizaje será muy lento inicialmente. Note que cuando todos los parámetros libres son ceros, el vector gradiente es siempre cero ya que pasa a ser un punto singular en el paisaje del error. Otra mejor manera de inicializar los parámetros de la red se basa en escojer los valores de los pesos y predisposición tal que las partes "pendiente" de las neuronas en la capa escondida puedan cubrir el espacio de entrada.

Todas las neuronas en un MLP deberían actualizarse aproximadamente a la misma tasa. No obstante, las señales de error en la capa de salida tienden a ser más largas que

aquellas en la capa frontal de la red. Esto puede ser visto directamente en la Ecuación (13), donde $x_i(1-x_i)$ aparece una vez en la capa de salida, dos veces en la capa anterior a la salida, y así sucesivamente. Note que el término $x_i(1-x_i)$ es siempre menor que o igual a 0.25. Por esto las señales de error en las capas frontales tienden a ser más pequeñas debido a múltiples multiplicaciones de este término. Por lo tanto, la tasa de aprendizaje η de las capas frontales deberían ser más largas que las de la capa de salida. Esto es llamado *reescalamiento de la tasa de aprendizaje*.

III.4.3 Poder de aproximación de los MLP

El poder de aproximación de los MLP's de retropropagación ha sido explorado por algunos investigadores. Todavía hay muy poca orientación teórica para determinar el tamaño de la red en términos del número de nodos escondidos y capas escondidas que debería contener. Cybenko , 1989, mostró que un MLP de retropropagación, con una capa escondida y algúna función no lineal, sigmoidal, contínua, y fija, puede aproximar bien cualquier función contínua arbitraria en un conjunto compacto. Cuando se usa como una red neuronal de valores binarios con la función de activación tipo supresor de picos (escalón), un MLP de retropropagación con dos capas escondidas puede formar regiones de decisión complejas arbitrarias para separar diferentes clases. Para la aproximación de funciones también como para clasificación de datos, dos capas escondidas pueden ser requeridas para aprender un rango de una función contínua.

III.5 Aprendizaje reforzado

El *aprendizaje reforzado* ha sido aceptado como un paradigma fundamental para el *aprendizaje de máquina* con un énfasis particular en aspectos computacionales de aprendizaje.

El aprendizaje reforzado es un esquema de aprendizaje a prueba y error a través del cual un agente computacional aprende a realizar una acción apropiada recibiendo una retroalimentación (llamada señal de reforzamiento, puntuación de actuación, grado, etc.) a través de la interacción con el mundo (o ambiente) que incluye un maestro no explicito para cualquier instrucción correcta. El agente que aprende (o aprendiz) se reforza a sí mismo a travéz de las lecciones falladas; acumulando errores que lo dirigirán al exito y no al desatre. Este método de aprendizaje puede ser considerado una simple forma de ajustar el comportamiento, y puede ser encontrado en animales aprendiendo habilidades y arreglandoselas con el ambiente. También coincide con nuestras ideas de sentido común:

> *Si una acción es seguida por un estado satisfactorio de asuntos, o un mejoramiento en el estado de asuntos, entonces la tendencia a producir esa acción es fortalecida o reforzada (recompensada). De otra manera, esa tendencia es debilitada o inhibida (penalizada).*

Hay un vasto cuerpo de literatura sobre aprendizaje reforzado, también conocido como *aprendizaje clasificado*. Ha sido discutido en juegos, control de robots autónomos, y otros. Comunmente las estrategias de aprendizaje reforzado aplicadas son clasificadas dentro de algunas categorías del punto de vista de la arquitectura; Sutton , 1990, delineó cuatro arquitecturas representativas básicas para aprendizaje reforzado:

- Sólo política
- Comparación reforzada
- Heurística crítica adaptativa
- Q-aprendizaje

III.5.1 El fallo es el camino más seguro al exito

Un agente usualmente aprende a transformar una representación de un estado a una acción apropiada, o a la distribución de probabilidad sobre un conjunto de acciones. Esta transformación es llamada *política*. La más simple arquitectura de aprendizaje reforzado consiste solamente de una política ajustable, la cual es llamada una arquitectura *sólo política*.

Viaje del premio

El objetivo del aprendizaje reforzado es encontrar una *política* óptima para seleccionar una serie de acciones por medio de un esquema de recompensa-penalización. Consideramos que la aplicación de un esquema elemental de recompensa-penalización al problema del viaje del premio que requiere encontrar una trayectoria optima al "oro" en la red simple triangular ilustrada en la Figura 13.

Imaginamos que muchos viajeros comienzan su viaje del premio, deseando encontrar oro desde el punto de comienzo A. En cada vértice, hay un poste indicador que tiene una caja con algunas piedras blancas y negras en ella. Un viajero toma una piedra de la caja del poste indicador y sigue ciertas instrucciones. Cuando una piedra "blanca", la cual significa "ve diagonalmente hacia arriba", es tomada, es denotada por la acción u. Inversamente, cuando una piedra "negra" es seleccionada, es denotada por la acción d la cual significa "ve diagonalmente hacia abajo".

Supóngase que el viaje siempre comienza en el vértice A, y el oro es puesto en el vértice final H. Además supongase que los viajeros siguen estrictamente las intrucciones de los postes indicadores.

Cada comportamiento del viajero puede ser descrito como sigue: En el vértice A, recoje una piedra (selección) y la pone en el poste indicador: de acuerdo con la instrucción de la piedra, prosigue al siguiente vértice (acción). Repite éste procedimiento

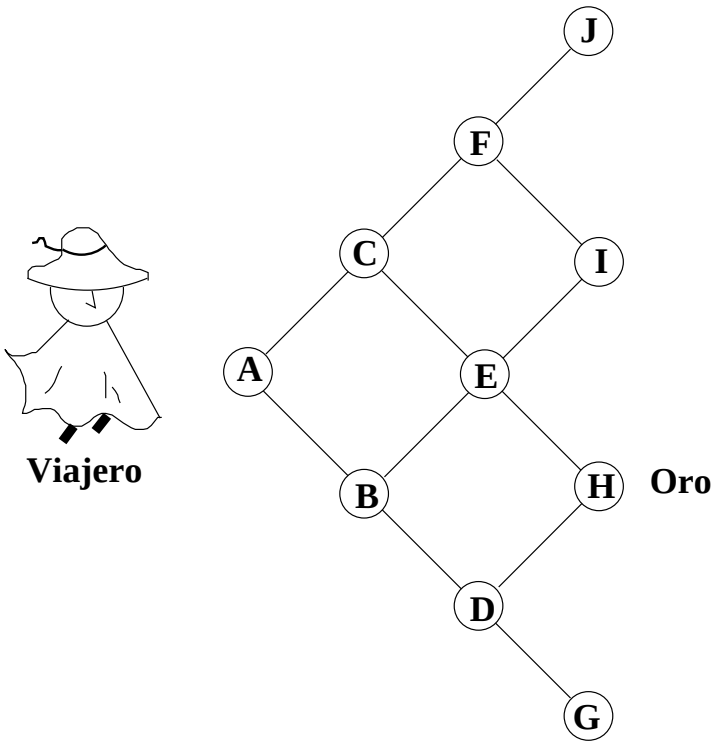


Figura 13: Problema del viaje del premio.

de selección-acción en el segundo y tercer vértices. (Despues de la tercera acción, el viajero alcanzará uno de los cuatro vértices terminales: G, H, I, o J). Cuando el premio se alcanza (éxito), prepara un esquema *hacia atrás*. Cuando el oro no es encontrado (falla), prepara un esquema de *penalización*. Entonces regresa al vértice inicial A; en cada vértice visitado, aplica el esquema de recompensa o penalización. Esto es, poner la piedra tomada dentro de la caja del poste indicador con una piedra adicional del mismo color (recompensa), o sacar la piedra tomada del poste indicador (penalización). Cuando el viajero regresa, el proximo viajero retomará el camino con un poco más de información. Repitiendose el mismo viaje muchas veces. Obviamente, la probabilidad de encontrar una política óptima se incrementará cuando más y más viajes se realizen.

En conformidad con la terminología, podemos tabular los términos relevantes como sigue:

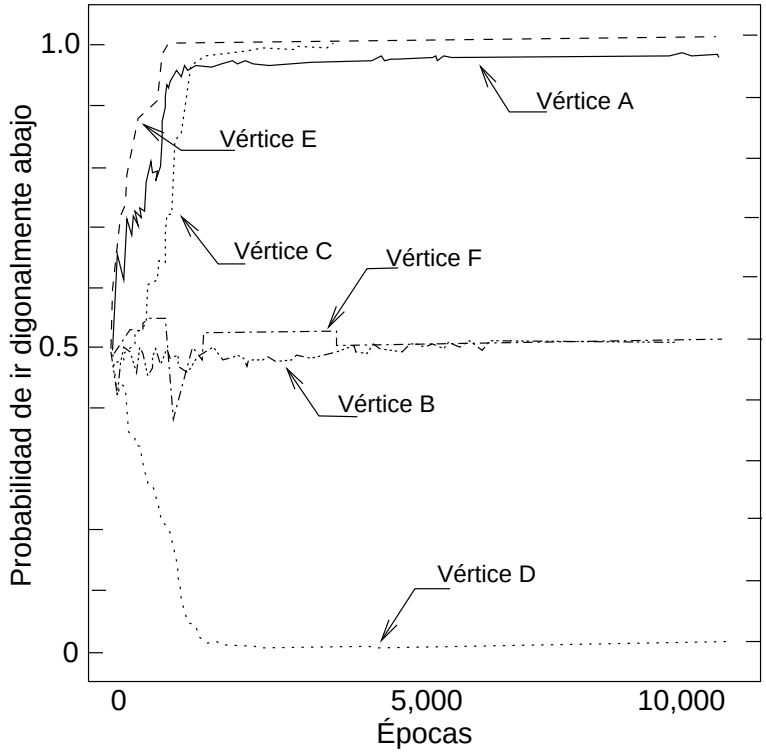


Figura 14: Cambio de la probabilidad de la acción d, "ve diagonalmente hacia abajo", en cada vértice no terminal (A-F) como progresan las pruebas en el problema del viaje del premio.

configuración de la trayectoria de la red	$\longleftrightarrow$	mundo o ambiente
viajero	$\longleftrightarrow$	agente o aprendiz
vértice (intersección)	$\longleftrightarrow$	estado
recoger una piedra	$\longleftrightarrow$	seleccionar una acción
secuencia de tres piedras	$\longleftrightarrow$	política (o trayectoria)

Es fácil simular tales viajes en computadora. Definimos la probabilidad P_{abajo} de la acción d, para cada estado como sigue:

$$P_{\text{abajo}} = \frac{Num_{\text{negro}}}{Num_{\text{negro}} + Num_{\text{blanco}}}, \tag{19}$$

donde Num_{negro} es el número de piedras negras y Num_{blanco} es el número de piedras blancas. De manera acorde, la probabilidad P_{arriba} de la otra acción u, puede definirse como:

$$P_{\text{arriba}} = \frac{Num_{\text{blanco}}}{Num_{\text{negro}} + Num_{\text{blanco}}} = 1 - P_{\text{abajo}}. \tag{20}$$

La Figura 14 muestra el cambio de P_{abajo} en los seis vértices no terminales (A ... F) como pruebas que progresaron. Inicialmente, todos los postes indicadores tienen 20 piedras negras y 20 blancas, y $P_{\text{abajo}} = P_{\text{arriba}} = 0.5$. En los vértices C y E, la acción especificada fue la acción d donde la acción dictada fue la acción u en el vértice D. Un alto P_{abajo} en el vértice A (punto inicial) favorece a la acción d. Esto debe ser porque en el siguiente vértice B, no tenemos oportunidad de perder una trayectoria hacia el oro no importa la acción que hagamos, la situación es diferente en el otro vértice alternativo C. Esta situación resultante confirma que tal esquema simple de recompensa-penalización es aplicable a aprender una "política óptima".

Michie , 1961, aplicó éste esquema de recompensa-penalización a un juego de gato, usando algunos cientos de cajas de cerillos con cuentas coloreadas. Las cajas de cerillos corresponden a los posibles estados del juego, y las cuentas coloreadas especifican

acciones admisibles. Él llamó a éste modelo MENACE (Matchbox Educable Naughts And Crosses Engine), mostró que el esquema llevó a MENACE a aprender una "política óptima" para ganar el juego del gato. Más tarde, Michie propuso el sistema de "cajas" basado en un esquema similar a MENACE para resolver el problema de control de balanceo del poste.

III.5.2 Aprendizaje de diferencia temporal

Los métodos de *diferencia temporal* (TD) son una clase de procedimientos de aprendizaje incremental especializado para predicción a través del credito que es asignado basado en la diferencia entre predicciones temporalmente sucesivas.

En estudios anteriores, Samuel , 1959, utilizó un método de TD en el programa del juego de damas para mejorar las funciones de evaluación parametrizadas, también conocidas como funciones lineales de las variables de entrada. De esta forma a través de la afinación por medio de la experiencia, el programa podría aprender en base a su experiencia y de esta forma mejorar su desempeño gracias a la actualización de parámetros. Más específicamente, el programa de Samuel usó la diferencia de la evaluación entre una tabla de configuración y una tabla de configuración futura probable. Elige la más ventajosa posición en la ausencia de información completa en cualquier estado temporal del juego.

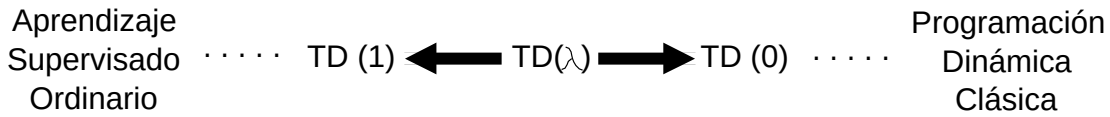


Figura 15: Espectro de aprendizaje TD. TD(λ) migra varios grados del espectro del aprendizaje TD de acuerdo al valor de λ.

Formulación de TD

En la forma general de los métodos de TD, TD(λ), los parámetros modificables w del predictor del agente obedece la siguiente regla de actualización con una tasa de aprendizaje α:

$$TD(\lambda) : \Delta w_t = \alpha(V_{t+1} - V_t) \sum_{k=1}^{t} \lambda^{t-k} \nabla_w V_k, \tag{21}$$

donde V_t es el valor de predicción en un tiempo t y λ es un parámetro (restante) reciente en el rango de 0 a 1. Los ajustes a las predicciones que ocurrieron en k pasos (o estados) en el pasado son exponencialmente pesados, en donde predicciones más recientes hacen más grandes los cambios en los pesos. Esto puede igualar las estrategias del cerebro biológico para decidir que tan fuerte el estimulo recientemente recibido debería ser utilizado en combinación con el presente estímulo para determinar acciones. Esto puede ser visto como un procedimiento de aprendizaje supervisado usual para el par de predicción actual V_t (salida "actual") y su predicción subsecuente V_{t+1} (salida "deseada") en el término de error. En éste sentido, el aprendizaje reforzado (TD) puede considerarse como una forma de aprendizaje error-corrección supervisado, el cual a menudo minimiza el error cuadrático E_{td} entre el resultado final z y la predicción actual V_t.

$$E_{td} = \frac{1}{2} \{z - V_t\}^2 = \frac{1}{2} \left\{ \sum_{k=t}^{m} (V_{k+1} - V_k) \right\}^2, \tag{22}$$

donde $V_{m+1} = z$. La Figura 15 muestra que TD(λ) migra varios grados del espectro de aprendizaje TD de acuerdo al valor de λ.

En dos casos particularmente extremos en los cuales $\lambda = 1$ y $\lambda = 0$, se tienen TD(1) y TD(0), respectivamente, como sigue:

$$TD(1) : \Delta w_t = \alpha(V_{t+1} - V_t) \sum_{k=1}^{t} \nabla_w V_k, \tag{23}$$

y

$$TD(0) : \Delta w_t = \alpha(V_{t+1} - V_t) \nabla_w V_t. \tag{24}$$

Note que aquí se utiliza una convención "$0_0 = 1$". En el caso especial en el cual TD(1) como se define en la Ecuación (23), todas las predicciones pasadas hacen contribuciones iguales a las alteraciones de los pesos; ésto significa que todos los estados son igualmente pesados. Porque $\Delta w_t \propto -\nabla_w E_{td} = \nabla_w V_t(z - V_t)$, debido a la Ecuación (22), la ecuación (23) puede ser escrita como sigue:

$$\Delta w_t = \alpha(z - V_t) \nabla_w V_t, \tag{25}$$

El error de predicción es representado como la diferencia entre el resultado final z y la predicción actual V_t.

Considérese la diferencia entre la Ecuación (23) y la Ecuación (25). La Ecuación (25) es más cercana actualmente a un procedimiento de *aprendizaje supervisado ordinario* para el par de predicción actual V_t (salida actual) y su consiguiente final z (salida objetivo). Usando la Ecuación (25), Δw_t puede ser determinada sólo después de que toda la secuencia de acciones ha sido completada y el resultado final z está disponible. Por lo tanto, todos los pares de predicción de estado deben ser recordados para hacer éste ajuste. En otras palabras, la Ecuación (25) no puede ser calculada incrementalmente en problemas multipaso (estado). Recordando el problema anterior del viaje del premio; el modelo del poste indicador realiza un tipo de aprendizaje similar al estipulado en la Ecuación (25). Secuencias pasadas deben ser recordadas para hacer ajustes

de piedras; ésto es, el viajero debe recordar rastros pasados para aplicar el esquema recompensa-penalización a cada poste indicador, y todos los estados reciben señales de reforzamiento iguales. (i.e. una piedra). Por otro lado, la Ecuación (23) ofrece una manera para calcular incrementalmente, lo cual ahorra el espacio en la memoria requerida para almacenar valores pasados.

En el otro extremo en el cual TD(0) está definido como en la Ecuación (24), sólo la más reciente predicción afecta las alteraciones. Esto es más cercano a un procedimiento de *programación dinámica* (DP discutido más adelante).

Sutton , 1988, introdujo un juego de ejemplo, ilustrado en la Figura 16, para clarificar la ineficiencia de los métodos de aprendizaje supervisado en comparación con los métodos de TD. La Figura 16 muestra un ejemplo donde los métodos supervisados de aprendizaje evaluan pobremente un estado nuevo, el cual es alcanzado por primera vez, y entonces sigue la trayectoria marcada por las lineas punteadas. Los métodos de aprendizaje TD predicen un estimado de un estado nuevo, V_t(NEW), considerando predicciones sucesivas temporalmente, incluyendo V_{p+1}(BAD), junto con la victoria observada (+1), mientras que los métodos de aprendizaje supervisado asocian a V_t(NEW) completamente con V_{m+1}(WIN) ($\equiv +1$). La Figura 16 representa un caso en el cual la trayectoria seguida desde un nuevo estado alcanza un inusual triunfo vía un estado malo que ha dejado 90% del tiempo a perder y sólo 10% a ganar en la experiencia pasada. En éste caso, los métodos de TD evaluan razonablemente el nuevo estado ajustando los valores de los estados observados en una manera temporalmente sucesiva. Los métodos de aprendizaje supervisado, por otro lado, asocian el estado nuevo totalmente con la victoria observada, aunque el nuevo estado es mucho menos prometedor.

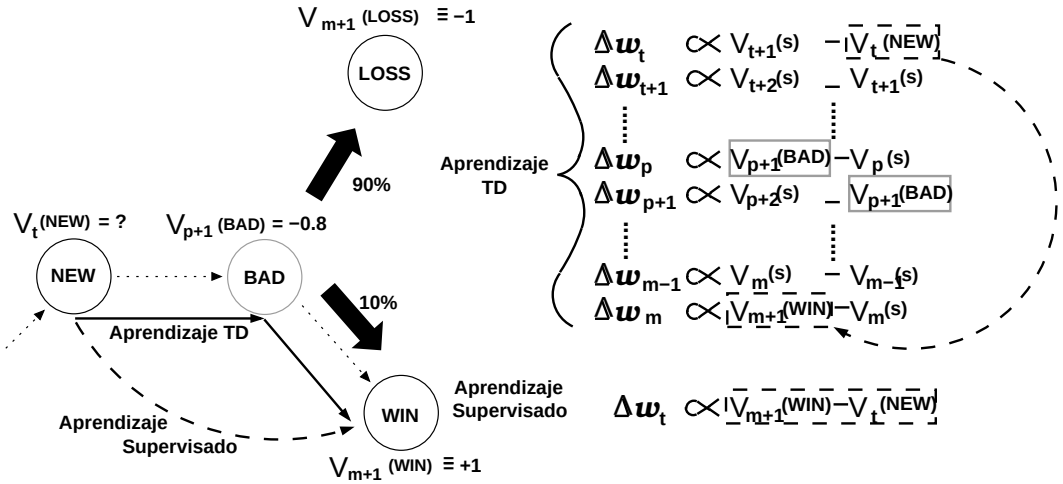


Figura 16: En ejemplo de métodos supervisados de aprendizaje.

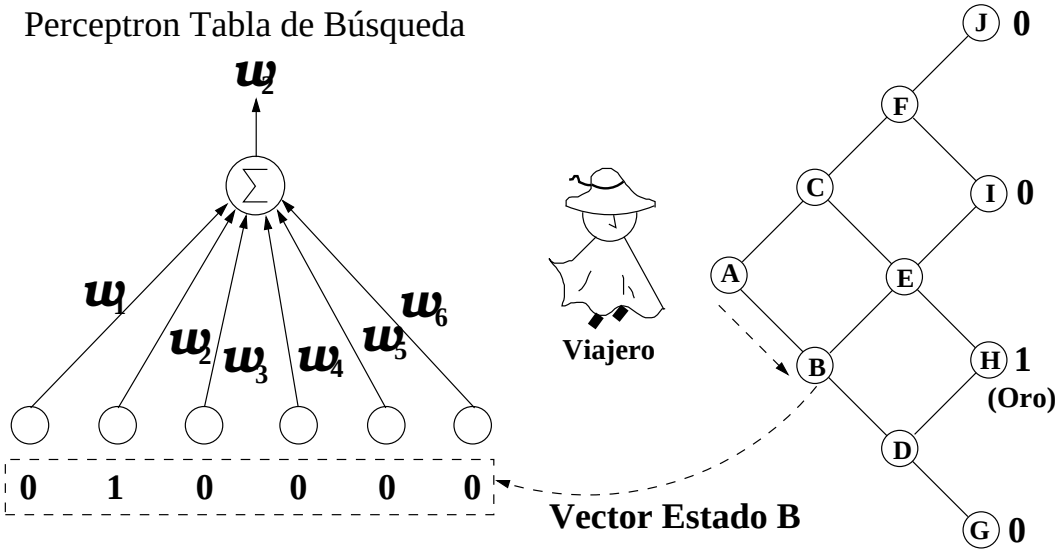


Figura 17: Perceptron tabla de búsqueda para aproximar los valores esperados en el problema del viaje del premio.

El premio esperado

Volvemos a visitar el paraíso del premio discutido anteriormente. Aquí se muestra como el TD(0) de la Ecuación (24) trabaja utilizando el perceptron de tabla de búsqueda con vectores estado linealmente independientes representados en la Figura 17. En éste caso, se puede aplicar la regla de *aprendizaje supervisado lineal* (Widrow-Hoff): se tiene $V_t = w^T s_t$, y así $\nabla_w V_t = s_t$. Por lo tanto,

$$\text{Lineal TD(0): } \Delta w_t = \alpha(w^T s_{t+1} - w^T s_t)s_t.$$

Note que hay seis vértices no terminales en la Figura 17. Los estados (i.e. vértices) son expresados como vectores base linealmente independientes $[s(i)]$ de longitud 6. En otras palabras, el vértice A puede ser definido como un vector, $(100000)^T$, y el vértice B puede ser definido como otro vector linealmente independiente, $(010000)^T$, y así sucesivamente formando una base..

Suponga que nuestro resultado del premio está definido como $z = 1$ para el vértice objetivo H donde el oro está presente, y definido como $z = 0$ para los otros vértices terminales G, I, y J, donde el oro no está presente. Por ésta elección de z, el valor esperado de cada vértice es equivalente a la probabilidad de alcanzar el oro (vértice H) desde ese vértice. Sutton , 1988, dicutió un ejemplo de caminata aleatoria, en la cual él introdujo una interpretación probabilistica similar.

Un agente aleatorio e igualmente probable selecciona la acción u, "ve diagonalmente hacia arriba", o la acción d, "ve diagonalmente hacia abajo", en cada vértice. Esta probabilidad no es cambiada basada en la experiencia, como lo fue en el problema de optimización del viaje del premio. Dada ésta "política 0.5-0.5" particular, el agente aprende los valores esperados. Para la tercer acción, la recompensa terminal proveida por el mundo es utilizada como el valor resultante más que la salida de la red TD. Esto es concebido como la realización de las condiciones de restricción.

Tabla III: Probabilidades predichas para seis vértices no terminales en tres etapas de aprendizaje distintas usando un perceptron de tabla de búsqueda con una tasa de aprendizaje pequeña (0.001). RMSE significa "root-mean-squared error".

Epoca	Vértice A	Vértice B	Vértice C	Vértice D	Vértice E	Vértice F	RMSE
10,000	0.3109	0.4248	0.2163	0.4178	0.4718	0.0231	0.05624
20,000	0.3656	0.4922	0.2536	0.5066	0.4995	0.0019	0.00590
40,000	0.3742	0.5008	0.2496	0.5016	0.5013	0.0	0.00155
Objetivo	0.375	0.5	0.25	0.5	0.5	0.0	0

Las probabilidades ideales de alcanzar el oro (i.e., las predicciones deseadas) para cada uno de los estados no terminales son 0.375, 0.5, 0.25, 0.5, 0.5, y 0.0 para los vértices A, B, C, D, E, y F, respectivamente. Los resultados obtenidos del perceptron que forman la tabla de búsqueda con una tasa de aprendizaje pequeña fija igual a 0.001, son mostrados en la tabla III; los cuales requirieron muchas iteraciones para la convergencia.

Predicción de resultados acumulados

Considérese un caso en el cual cada acción en una secuencia incurre en un costo. Por ejemplo, en la red de ruta simple dibujada en la Figura 18, los números entre cualquiera de dos vértices representan el costo de viajar ese intervalo. En cada vértice, necesitamos V_t para estimar el costo acumulado restante más que el costo total de la secuencia. Los métodos de TD pueden ser esperados para tratar con este caso. Esto es, el método de TD no está reducido a la predicción solamente del resultado final de la secuencia, la cual puede también ser empleada para estimar cantidades acumuladas sobre las secuencias.

Dejese que c_{t+1} denote el costo actual incurrido entre los tiempos t y $t+1$. Queremos igualar V_t al valor esperado de $z_t = \sum_{k=t}^{m} c_{k+1}$, donde m es el número de vectores de

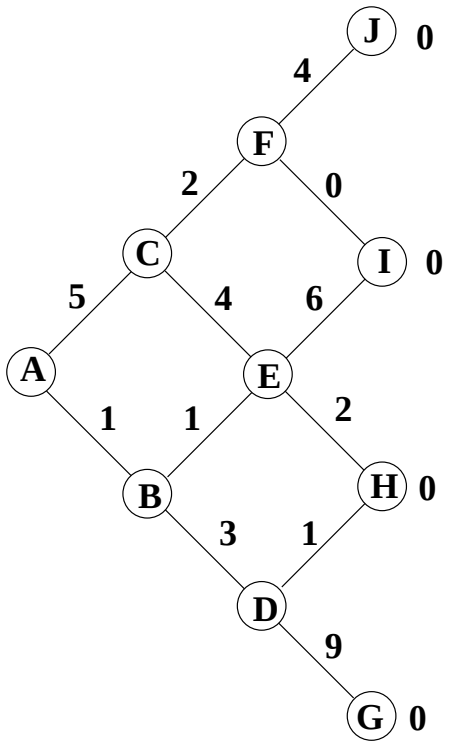


Figura 18: Problema simple de costo de trayectoria.

observación en la secuencia.

Como en la Ecuación (25), el error de predicción puede ser representado como:

$$\begin{aligned} \text{(resultado final) - (predicción actual)} &= z_t - V_t \\ &= \textstyle\sum_{k=t}^{m}(c_{k+1} - V_t) \\ &= \textstyle\sum_{k=t}^{m}(c_{k+1} + V_{k+1} - V_k), \end{aligned} \tag{26}$$

donde $V_{m+1} = 0$ (condiciones límite). Podemos así derivar la regla de actualización para el siguiente TD(λ) acumulado:

$$\Delta w_t = \alpha(c_{t+1} + V_{t+1} - V_t) \sum_{k=1}^{t} \lambda^{t-k} \nabla_w V_k. \tag{27}$$

En problemas de predicción infinita, no hay estados meta para determinar la suma

z_t. Para prevenir la divergencia de la suma, pueden ser introducidos factores de descuento. Así, el objetivo del agente es minimizar la suma de descuento de futuros costos dados por:

$$\sum_{k=0}^{\infty} \gamma^k c_{t+k+1} = c_{t+1} + \gamma c_{t+2} + \gamma^2 c_{t+3} + \cdots \tag{28}$$

En éste caso, la ecuación 27 será:

$$\Delta w_t = \alpha(c_{t+1} + \gamma V_{t+1} - V_t \sum_{k=1}^{t} \lambda^{t-k} \nabla_w V_k. \tag{29}$$

TD(γ) ha sido probado que converge sólo por una red lineal con entradas de vectores estado linealmente independientes. Sin embargo, construyendo una RN no lineal con vectores estado linealmente no independientes que usó el TD(γ) para jugar el juego del gato, Tesauro fue capaz de reportar una aplicación del juego exitosa (TD-gammon) y discutió temas prácticos concernientes al aprendizaje TD.

III.5.3 El arte de la programación dinámica

La *programación dinámica* (PD) es un procedimiento de optimización que es aplicable particularmente a problemas que requieren una secuencia de decisiones interrelacionadas.

En relación con la PD, algunos investigadores han discutido el aprendizaje reforzado. Watkins y Dayan , 1992, propuso una clase de Q-aprendizaje que emplea "PD incremental", lo cual será descrito más tarde. Se deben enfatizar los siguientes dos aspectos de PD con respecto al aprendizaje reforzado:

- La PD aproxima sucesivamente funciones de evaluación óptimas resolviendo las relaciones de recurrencia, en lugar de conducir una búsqueda en el espacio de

estados.

- La PD Respalda las evaluaciones de los estados en una propiedad fundamental de procedimientos iterativos empleados para resolver las relaciones de recurrencia.

Formulación de la programación dinámica clásica

Para los propósitos de discuciones siguientes, presentamos un breve vistazo al arte de la PD básica. Aunque hay muchas variaciones de PD, nos enfocaremos en conceptos mayores comunes a todos los procedimientos de PD.

Primero, delimitamos el *principio de optimalidad*, la columna vertebral de la PD, la cual fluye de la intuición:

> *Una política óptima como la propiedad que cualquiera que sean el estado inicial y la decisión inicial; las decisiones restantes deben contribuir a una política óptima con la consideración del estado resultante de la primer decisión.*

A la luz del principio de optimalidad, los elementos del procedimiento básico (hacia átras) de PD son como siguen:

- Se reconoce que un "problema completo" dado puede ser resuelto si los valores de las mejores soluciones de ciertos subproblemas pueden ser determinados de acuerdo al principio de optimalidad.

- Se da cuenta de que si se comienza del final o cerca del final del "problema completo", los subproblemas son tan simples como para tener soluciones triviales. El principal atractivo (hacia átras) de PD reside en tal facilidad.

En el arte de la PD, es importante definir un "espacio de estados" apropiadamente y elegir los argumentos de la función de valor óptimo (la regla de asignar valores a varios

subproblemas) de modo que un proceso real bajo consideración tiene la propiedad de Markov y así el principio de optimalidad es válido. La formulación usual de la PD consiste en determinar las cuatro definiciones apropiadas: (1) una función de valor óptimo, (2) una función de política óptima, (3) una relación de recurrencia, y (4) condiciones de frontera.

Recordando el problema ilustrado en la Figura 18, allí se busca el costo esperado de una política dada utilizando métodos de TD. (Sin opción de acción para ser hecha, esto no es un problema de PD.) Ahora considerando que el objetivo del agente aprendiz es seleccionar acciones de modo que lleva al mundo a un estado meta (i.e. vértice terminal) $g \in G \subset S$ en un costo acumulado mínimo, donde G es un conjunto de estados meta y S es el conjunto de todos los estados. Asimismo se introduce una medida de evaluación (i.e., la función de valor óptimo $V(s)$) como sigue:

$$V(s) \equiv \text{el valor de costo mínimo de ir desde un estado } s \text{ a una meta } g.$$

La relación de recurrencia dada por el principio de optimalidad únicamente define esos valores. Esto es, el costo mínimo de un estado s debe igualar el costo de la mejor acción a, mas el costo mínimo del siguiente estado designado por la acción a:

$$V(s) = min_{a \in \text{acciones}} \left\{ \text{cost}(s,a) + V(\text{next}(s,a)) \right\}, \quad \forall s \in S - G, \tag{30}$$

donde $\text{next}(s,a)$ denota el estado subsecuente al estado s dictado por la acción a, y $\text{cost}(s,a)$ significa el costo incurrido de la acción a de la siguiente transición. En otras palabras, la evaluación $V(s)$ del estado s debería ser igual al mejor "$\text{cost}(s,a)$ mas el valor del siguiente estado $t \equiv \text{next}(s,a)$ que puede ser alcanzado con una acción a". Los valores de los estados meta (o vértices terminales) son definidos por:

$$V(s) = 0, \quad \forall s \in G. \tag{31}$$

La Ecuación (31) muestra las condiciones de frontera. (Note que la función de política óptima π es obvia desde la Ecuación (30); esto es, $\pi(s) = a$ tal que $V(s) = min_{a \in \text{acciones}} \{\text{cost}(s, a) + V(\text{next}(s, a))\}$.)

Para un caso *estocástico* de éste problema de ruta de mínimo costo, generalmente se supone que cuando la acción d es sugerida, es seguida con una probabilidad P_{d1}; con probabilidad $P_{u1}(= 1 - P_{d1})$, donde se prueba la acción opuesta u. Asimismo, cuando la acción u se introduce, es seguida con probabilidad P_{u2}; con probabilidad $P_{d2}(= 1 - P_{u2})$, se realiza la acción d. Ahora $V(s)$ es la función de valor óptimo esperado:

$V(s) \equiv$ el valor esperado del costo mínimo de ir desde un estado s a una meta g.

Entonces podemos definir especificamente la relación de recurrencia en una forma general como sigue:

$$V(s) = \min \begin{bmatrix} P_{d1} \{\text{cost}(s, d) + V(t_d)\} + P_{u1} \{\text{cost}(s, u) + V(t_u)\} \\ P_{u2} \{\text{cost}(s, u) + V(t_u)\} + P_{d2} \{\text{cost}(s, d) + V(t_d)\} \end{bmatrix}, \qquad (32)$$

donde $t_d \equiv \text{next}(s, d)$, y $t_u \equiv \text{next}(s, u)$.

Los métodos de TD aproximan la función de evaluación para una política dada sin conocer las probabilidades del estado-transición y la función que determina los valores de las recompensas esperadas. Barto *et al.* , 1990, mencionaron que "El proceso de aprendizaje TD es una aproximación Monte-Carlo de un método de aproximación de PD sucesivo". También discutieron que los métodos TD llevan una gran semejanza a la PD desde un punto de vista del aprendizaje. El procedimiento mencionado de PD hacia átras trabaja desde el final de la tarea de decisión hasta el principio. Parece estar dificilmente relacionado con los procesos de aprendizaje animal, debido a este procesamiento de átras hacia adelante. Barto *et al.* , 1990, mostraron como el aprendizaje TD puede obtener el mismo resultado que PD para los pases hacia adelante repetidos a través de una tarea de decisión. El cálculo es mejorado incrementalmente moviendo

hacia delante los estados meta. Este punto de vista puede ser encontrado en la llamada PD hacia delante.

Programación dinámica incremental

Aplicar la PD clásica requiere conocimiento de las probabilidades de estado-transición, como se demuestra en la Ecuación (32). En la ausencia de información explicita de la probabilidad, se puede aproximar la función de valor. Para mejorar la aproximación $\hat{V}(s)$ de la función de valor, es posible aplicar la PD incremental (también conocida como PD de aproximación) dada por la siguiente ecuación funcional:

$$\hat{V}(s) \leftarrow \min_{a \in A} \left\{ \text{cost}(s,a) + \hat{V}(\text{next}(s,a)) \right\}, \quad \forall s \in S - G, \tag{33}$$

donde para $s \in G$, $\hat{V}(s) = V(s) = 0$ (condiciones límite). Sutton , 1991, declaró que *la aplicación iterativa de la Ecuación (33) a un estado después de otro hace de* $\hat{V}$ *una aproximación cada vez mejor de* V*. Ésta operación es, en esencia, una búsqueda una de etapa hacia adelante del estado* s*, seguida por un reemplazo del valor aproximado* $\hat{V}(s)$ *con su valor de respaldo.* Este procedimiento parece convincente, pero hay una advertencia importante que la convergencia puede no ser monotónica; esto es, el procedimiento convergerá a valores correctos si lo hace a menudo en todos los estados posibles. Cada estado puede no nesesariamente mejorar a cada respaldo porque el valor de aproximación $\hat{V}(s)$ será peor si un estado tiene un valor correcto y el siguiente estado tiene uno erróneo; el valor entonces será erróneo temporalmente. En problemas estocásticos, sólo el promedio de muchos valores (erróneos) puede ser bueno, o puede alcanzar a ser correcto si la tasa de aprendizaje se va a cero apropiadamente. En este sentido, el desempeño puede mejorar incrementalmente mientras el procedimiento está siendo llevado a cabo.

Cuando la función de valor alcanza el óptimo, la política actual se convierte más óptima por grados. La política óptima puede (se espera) ser obtenida seleccionando acciones que minimizen el lado derecho de la Ecuación (30) en cada estado. Esto es, el proceso de aprendizaje converge cuando la Ecuación (30) se mantiene para todos los estados. Watkins y Dayan , 1992, usaron este concepto para extender los métodos de TD y propusieron una clase de aprendizaje llamado Q-aprendizaje.

Cuando el valor predictor del agente se realiza por una función de aproximación RN, el parámetro de peso de la regla de actualización utiliza simplemente el desajuste entre el lado izquierdo de la Ecuación 33 y el lado derecho (i.e., el error TD en las ecuaciones recursivas porque tales ecuaciones recursivas pueden expresar las relaciones estructurales secuenciales deseadas en el problema dado.) Aquí notamos que la propiedad respaldo de la PD comparte conceptos básicos con los métodos de TD.

III.5.4 Crítica heurística adaptativa

En ésta sección, discutimos una estrategia de aprendizaje reforzado basado en la llamada *crítica heurística adaptativa* (CHA) o modelo *crítica-actor*. Esta provee una forma de itentar encontrar ambas acciones óptimas y valores esperados. La frase "aprendizaje con crítica" fue utilizada por Widrow *et al.* , 1973, para diferenciarla del aprendizaje supervisado ordinario caracterizado por la frase "aprendizaje con maestro". En algunas situaciones, donde hay menos información disponible sobre las salidas deseadas, sólo señales correctas o equivocadas serán proveídas.

El modelo CHA típicamente incluye dos componentes principales: el módulo de crítica (evaluación/predicción) y el módulo de acción (control). La crítica genera un estimado de la función de valor (o evaluación) de unos vectores de estado y reforzamiento externo provisto por el mundo (o ambiente) como entradas. Esto es, la crítica juega un rol importante en la predicción de la función de evaluación. La crítica es

adaptativa porque su componente predictor es actualizado empleando métodos de TD. El módulo de acción intenta aprender habilidades de control óptimo o tomar decisiones.

Crítica como neurona

Barto *et al.* , 1983, introdujeron un modelo de crítica adaptativa como neurona para un problema de control de balanceo del poste. Ellos discutieron su modelo en conjunto con un primer estudio del sistema Boxes que Michie y Chambers , 1968 propusieron.

El sistema Boxes fue construido para resolver el problema de control de balanceo del poste; su concepto central fue "descomposición de tarea" a través del cual el espacio de estados fue particionado en 162 subespacios no sobrepuestos llamados "cajas", y diferentes controladores fueros usados en cada espacio de estados (caja) digitalizado. La no *generalización* fue intentada a través de subespacios. Como el trabajo más reciente de Michie con MENACE, ya discutido, el modelo Boxes es conducido al resultado (especificamente), en éste caso es conducido al fallo. Esto es, el modelo no recibe retroalimentación sobre la realización hasta que el poste caiga (fallo). A través de muchos intentos, el sistema Boxes aprende a resolver el problema de control no lineal en el cual cada fallo es un escalón al exito.

Inspirado por el método Boxes, Barto *et al.* , 1983, construyeron un modelo de crítica adaptativa que consiste del elemento crítico adaptativo (ECA) y el elemento de búsqueda asociativo (EBA) basado en un algoritmo "asociativo recompensa-penalización". Cada componente se expresa en una ADELINE —que ellos llaman un "elemento conexionista como neurona". El EBA implementa y ajusta la política de decisión (o reglas de control). El ECA aprende a proveer las evaluaciones actuales de las decisiones de control en virtud de señales de fallo. Especificamente, el ECA predice la señal de "reforzamiento interno" $\hat{r}(t+1)$ (asociado con estados de entradas particulares):

$$\hat{r}(t+1) = r_{t+1} + \gamma V_{t+1} - V_t. \tag{34}$$

donde V_t es la predicción actual y γ es un factor de descuento. Este $\hat{r}(t+1)$ corresponde al error TD discutido anteriormente. El $\hat{r}(t+1)$ se envia al EBA para determinar una acción de control.

La comparación del desempeño con el sistema Boxes señala un avance del modelo de crítica de Bartos et al., "EBA con reforzamiento interno proveido por un ECA":

> *El sistema de cajas está restringido en que su diseño fue basado en el conocimiento a priori de que el tiempo hasta el fallo era para servir como criterio de evaluación y que el proceso de aprendizaje sería dividido en distintas pruebas que terminarían siempre con un señal de fallo. El EBA, por otro lado, es capaz de trabajar para alcanzar eventos recompensantes y evitar eventos penalizantes los cuales podrían ocurrir en cualquier momento. No es manejado exclusivamente por fallos, y su operación está especificada sin referencia a la noción de una prueba.*

En su modelo de crítica adaptativa, Barto *et al.* , 1983, usaron 162 componentes (vectores estado) linealmente independientes como entradas. En otras palabras, éste modelo fue equivalente a una tabla de búsqueda grande similar al perceptron de tabla de búsqueda mostrado en la Figura 17. Así, éste modelo no ofrece la posibilidad de la generalización entre estados. Para problemas de espacios de estados grandes, el utilizar tablas de búsqueda no es práctico, como visitar todos los estados. Así, el agente necesita generalizar de una cantidad limitada de experiencia utilizando una representación compacta de los vectores de estado de entrada.

Algoritmo de crítica neural adaptativa

En ésta implementación se provee una descripción generalizada de un algoritmo CHA implementado en formas funcionales parametrizadas como aproximadores de funciones de RN (aproximadores de funciones de valor y acción). El modelo CHA básicamente consiste en dos RN's: la RN de valor y la RN de acción. La RN de valor aproxima funciones de evaluación, transformando estados a valores esperados, mientras que la RN de acción genera una acción plausible (o legal), transformando estados a acciones.

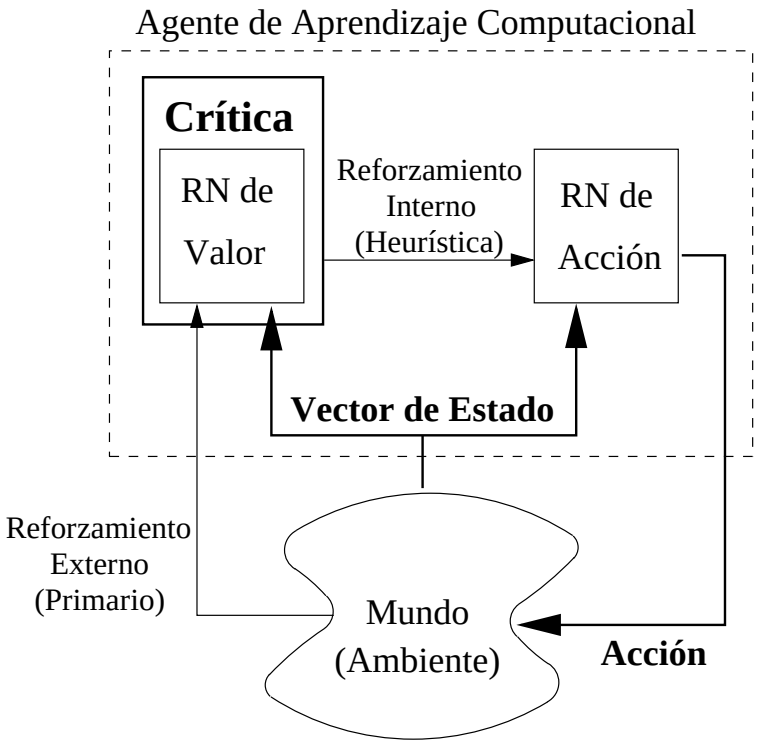


Figura 19: Modelo de CHA que utiliza aproximadores de función de red neuronal: la RN de valor y la RN de acción.

La Figura 19 ilustra un diagrama de bloques del modelo CHA basado en RN. La crítica adaptativa recibe reforzamiento externo (primario) del mundo y lo transforma en reforzamiento interno (heurístico).

La regla de actualización de peso sigue el esquema de minimización de error usualmente aplicado en aprendizaje supervisado definiendo el error cuadratico E_{td} como sigue:

$$E_{td} = \frac{1}{2} error^2,$$

$$\begin{aligned} error \quad \equiv \quad & \text{(valor incurrido por una acción seleccionada)} \\ & +\gamma \text{ (el valor esperado del estado sucesivamente observado producido} \\ & \qquad \text{por la RN de valor)} \\ & - \text{ (el valor esperado del estado actual producido por la RN de valor).} \end{aligned}$$

donde γ es una tasa de descuento, ver Ecuación (22). Ambas, la RN de acción y la RN de valor son entrenadas simultáneamente empleando el E_{TD}. El Algoritmo 1 explica una implementación del concepto CHA.

Algoritmo 1: *Crítica de heurística adaptativa usando aproximadores de función de redes neuronales.*

1. Observar el estado actual: $s \leftarrow$ estado actual s_n.
2. Usar la RN de valor para tener $V(s) : e \leftarrow V(s)$.
3. Seleccionar una acción a_n usando la salida de la RN de acción.
4. Ejecutar la acción a_n.
5. Observar el nuevo estado sucesivo t_n y el reforzamiento r_n.
6. Usar la RN de valor para calcular $V(t_n)$.
7. $E \leftarrow r_n + \gamma V(t_n)$.
8. Ajustar la RN de valor propagando el error $(= E - e)$.
9. Ajustar la RN de acción de acuerdo al error.

Recordando las Ecuaciones (27) y (29). El Algoritmo 1 muestra los procedimientos de aprendizaje junto con el procesamiento. Los coeficientes de peso de la RN de acción y la RN de valor se ajustan para alimentar valores y acciones aprendidos por métodos TD.

El Algoritmo 1 demuestra como utilizar la salida de una RN de valor para controlar la salida de un RN de acción por medio de la retropropagación. Para ser precisos, supongase que se quiere maximizar el valor esperado (i.e. recompensa). Cada acción debería ser elegida para maximizar la suma de todas las futuras recompensas. Así dada una acción a_n, si el valor del siguiente estado $V(t_n)$ [o $\gamma V(t_n)$] mas el reforzamiento externo r_n es más grande que el valor del estado actual $V(s_n)$ (i.e., $E-e>0$), la acción luce mejor que la esperada previamente, y por lo tanto, esa acción debería ser reforzada. En cambio, si la desigualdad contraria se mantiene, la acción no parece mejor. Esto es porque se desea maximizar las futuras recompensas acumuladas. La acción debería ser por lo tanto inhibida. Esto nos conduce a un algoritmo de recompensa-penalización. En contraste al Algoritmo 1, en el algoritmo recompensa-inacción, las acciones no son actualizadas cuando la acción seleccionada no luce mejor. Este algoritmo está basado en el concepto que antes que otras accciones son puestas en práctica, nada acerca de su efectividad puede ser aprendido de una sola experiencia.

En este sentido, la RN de acción y la RN de valor evolucionan juntas en el intento de encontrar una política. En otras palabras, los métodos TD son empleados para resolver problemas de asignación de credito temporal, y el algoritmo de retropropagación se utiliza por problemas de asignación de credito estructural.

III.5.5 Q-aprendizaje

El Q-aprendizaje es una forma de modelo libre de aprendizaje reforzado.

Concepto básico

El Q-aprendizaje es una manera simple de resolver problemas de acción Markoviana con información incompleta basado en la *función Q acción valor* que tranforma pares estado-acción a resultados esperados. La idea de asignar valores a los pares estado-acción puede ser vista en PD. Watkins y Dayan , 1992, llamaron Q-aprendizaje a la versión incremental de la PD. El Q-aprendizaje mejora sucesivamente sus evaluaciones de acciones particulares en estados particulares, justo como la PD incremental, discutida anteriormente, mejora sus evaluaciones de estados particulares.

El aprendizaje procede como con los métodos de TD; en donde un agente intenta una acción en un estado particular y evalúa su consecuencia en términos de la recompensa o penalización inmediata que recibe del mundo y su estimación del valor del estado resultante de la acción tomada. La ayuda del agente no es meramente maximizar su recompenza inmediata en el estado actual, sino maximizar su recompenza acumulada que recibe sobre algun periodo de tiempo futuro.

La arquitectura de aprendizaje reforzado CHA requiere dos memorias intermedias fundamentales: una para la función de evaluación y una para la política. Por otro lado, el Q-aprendizaje mantiene sólo una: un par compuesto de un estado s y una acción a (i.e. un Q-valor estimado de tomar a en s). En su lugar, el Q-aprendizaje requiere de complejidad adicional en determinar la política de los Q-valores.

El objetivo en el Q-aprendizaje es estimar los valores de una política óptima. El valor de un estado puede ser definido como el valor del mejor par estado-acción del estado:

$$V(s) = \max_a Q(s, a). \tag{35}$$

La política óptima está determinada de acuerdo a la función de política π:

$$\pi(s) = a \quad \text{tal que} \quad V(s) = Q(s,a) = \max_{b \in acciones} Q(s,b). \tag{36}$$

En el Q-aprendizaje de un paso, sólo la función Q de acción-valor del más reciente par estado-acción se actualiza después de un paso de retraso. La regla de actualización para los Q-valores en la n-ésima etapa, donde s_n es el estado actual y a_n es la acción seleccionada, está basada en métodos de TD:

$$Q_n(s,a) = \min \begin{cases} Q_{n-1}(s,a) + \eta_n[r_n + \gamma V_{n-1}(t_n) - Q_{n-1}(s,a)] & \text{si } s = s_n \text{ y } a = a_n, \\ Q_{n-1}(s,a) \quad \text{otra manera,} \end{cases} \tag{37}$$

donde

$$V_{n-1}(t) = \max_{b \in \text{acciones}} \{Q_{n-1}(t,b)\}.$$

En las etapas más tempranas de aprendizaje, los Q-valores pueden no reflejar exactamente la política que definieron implicitamente. Probando todas las acciones en todos los estados repetidamente, el agente aprende cual es la mejor de todas, juzgado por la recompensa descontada. En otras palabras, el "actual Q(s,a)" es "el Q-valor esperado de tomar la acción a en el estado s, y entonces utilizar acciones óptimas en todos los estados futuros".

Implementación

Ahora consideremos implementar un RN de Q-aprendizaje (Q-red). El Algoritmo 2 presenta un procedimiento de entrenamiento para tal Q-red, como se ilustra en la Figura 20.

Algoritmo 2: *Q-aprendizaje de un paso usando una red neuronal de función de aproximación.*

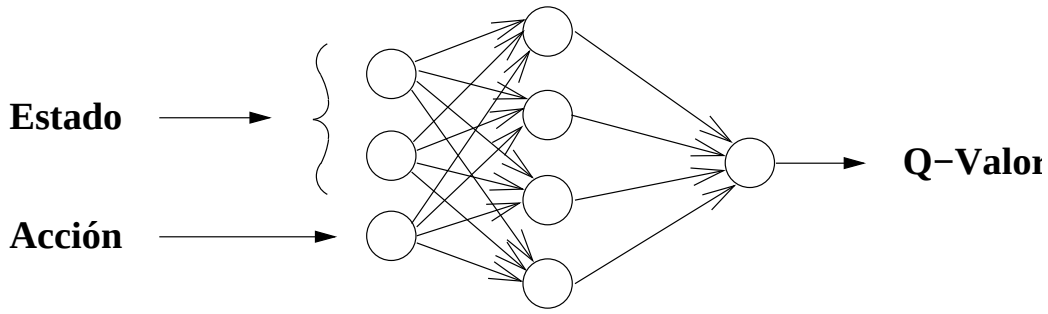


Figura 20: Arquitectura de una Q-red.

1. Observar el estado actual s_n.

2. Seleccionar una acción a_n por medio de un procedimiento estocástico.

3. Para la acción seleccionada a_n, utilizar la Q-red para calcular U_a : $U_a \leftarrow Q_{n-1}(s_n, a_n)$.

4. Ejecutar la acción a_n.

5. Observar el nuevo estado resultado t_n y reforzamiento $r_n : t \leftarrow t_n$.

6. Usar la Q-red para calcular $Q_{n-1}(t, b), b \in$ acciones.

7. $u \leftarrow r_n + \max_{b \in \text{acciones}} Q_{n-1}(t, b)$.

8. Ajustar la Q-red retropropagando el error de un paso:

$$\Delta U = \begin{cases} u - U_a & \text{si } s = s_n \text{ y } a = a_n, \\ 0 & \text{otra manera,} \end{cases} \tag{38}$$

una acción seleccionada usualmente iguala a $\pi(s)$ como está definido en la Ecuación (36), pero ocasionalmente llega a una alternativa; por ejemplo, la política es estocásticamene implementada de acuerdo a una distribución de Boltzmann:

$$Prob(a_i) = \frac{\exp[\frac{Q(s,a_i)}{T}]}{\sum_k \exp[\frac{Q(s,a_k)}{T}]}, \tag{39}$$

donde T es el parámetro de temperatura para un proceso de templado.

Esta sección concluye con una nota de un teorema de convergencia: el Q-aprendizaje basado en tablas de búsqueda ha sido probado para converger a valores y decisiones óptimos, mientras que el aprendizaje CHA ilustrado en la Figura 19 no la ha sido. Watkins y Dayan , 1992, mostraron especificamente que el Q-aprendizaje converge a los valores de acción óptimos con una probabilidad de 1 siempre que todas las acciones sean repetidamente muestreadas en todos los estados con valores de acción discretos, y la tasa de aprendizaje se vaya a cero apropiadamente.

III.5.6 Aprendizaje reforzado por medio de computación evolutiva

Animar a la brigada

Los sistemas clasificadores son sistemas de producción de mensajes de paso que aprenden secuencias temporales de reglas en cadena (llamadas clasificadores) a través de la asignación de crédito basado en el algoritmo de *animar a la brigada* y en una regla de descubrimiento basada en un AG (algoritmo genético). Animar a la brigada es un algoritmo que ajusta la fuerza de los clasificadores y determina cuál acción debe ser tomada. La ecuación de alteración de las fuerzas de los clasificadores determina cuál acción debe tomarse. La ecuación de alteración de las fuerzas de los clasificadores no es idéntica pero es similar a la regla de actualización de Q-valores en la Ecuación (37) y la fórmula de TD. La regla de descubrimiento es hecha por el AG, y por eso es estocástica por naturaleza. Además, el AG forma nuevos clasificadores plausibles a través de operaciones genéticas.

Sutton , 1988, especificó una diferencia entre los métodos de TD y el animar a una brigada. El animar a la brigada asigna crédito basado en reglas que activan otras reglas, mientras que los métodos de TD asignan crédito basados solamente en sucesiones temporales. El animar la brigada combina ambas asignaciones de créditos temporal y estructural en un solo mecanismo más arbitrario, aunque el mecanismo puede no necesariamente resolver correctamente los problemas de optimización.

Reforzamientos genéticos

Algunos investigadores emplean un *algoritmo genético* (AG) en el aprendizaje reforzado. Odetayo y McGregor , 1989, aplicaron un aprendizaje reforzado basado en un AG para el problema de control de balanceo del poste. Como con el enfoque del sistema Boxes, discretizaron el espacio de estados en 54 regiones; cada región contenía una regla de producción para especificar una acción (empuja izquierda o derecha). Un cromosoma consiste en 54 reglas cada una de las cuales representa "1" (empuja izquierda) o "0" (empuja derecha). Los cromosomas fueron valorados por eficiencia en el balanceo del poste y fueron sujetos a las operaciones genéticas usuales: reproducción, cruzamiento, y mutación. Fue reportado que el aprendizaje reforzado basado en un AG requiere menos iteraciones que el sistema Boxes y CHA.

Ackley y Littman , 1992, introdujeron el *aprendizaje reforzado evolutivo* (ARE), el cual combina evolución genética con aprendizaje de RN y un "ecosistema" de vida artificial. Está basado en la hipótesis de que "la evolución y el aprendizaje progresan sinergeticamente". Más especificamente, los pesos iniciales de las RN's de valor y acción son especificados geneticamente; los pesos de la RN de valor son evolucionados por un AG, y la salida de la RN de valor se emplea para entrenar la RN de acción por medio de lo que Ackley y Littman , 1992, llamaron *retropropagación de reforzamiento complementario* (RPRC). La RPRC, está basada en un algoritmo simple de

TD, encarnando una regla heurística que la salida deseada en reforzamiento negativo es el complemento de la salida generada por la RN de acción. Su ARE puede ser visto como una implementación de un concepto genético de CHA.

III.6 Aprendizaje no supervisado

En esta sección, discutiremos varios sistemas neurales que son catalogados frecuentemente fuera de la clase de aprendizaje supervisado puro de redes neuronales (RN).

Cuando no está disponible un maestro o una crítica de instrucción, sólo los vectores de entrada pueden ser empleados para el aprendizaje. Tal enfoque se conoce como aprendizaje sin supervisión, o como es comunmente referido *aprendizaje no supervisado*. Un sistema de aprendizaje no supervisado (o agente) evoluciona para extraer características o regularidades en los patrones presentados, sin mencionar cuales salidas o clases asociadas con los patrones de entrada son las deseadas. En otras palabras, el sistema de aprendizaje detecta o categoriza características persistentes sin ninguna retroalimentación del ambiente. Así, el aprendizaje no supervisado es frecuentemente empleado para agrupamiento de datos, extracción de características, y detección de similaridad.

El aprendizaje no supervisado de RN's intenta aprender a responder a diferentes patrones de entrada con diferentes partes de la red. La red es a menudo entrenada para fortalecer el "tiroteo" para responder a los patrones que frecuentemente ocurren, de ese modo dirijiendo al tan llamado sinónimo *estimadores de probabilidad*. En ésta manera, la red desarrolla ciertas representaciones internas para codificar patrones de entrada.

III.6.1 Redes de aprendizaje competitivo

Sin información disponible respecto a las salidas deseadas, el aprendizaje no supervisado actualiza los pesos sólo en la base de los patrones de entrada. La *red de aprendizaje competitivo* es un esquema popular para alcanzar éste tipo de agrupamiento o clasificación de datos no supervisada; La Figura 21 presenta un ejemplo. Todas la unidades de entrada i estan conectadas a todas las unidades de salida j con pesos w_{ij}. El número de entradas es la dimensión de la entrada, mientras que el número de salidas es igual al número de grupos en que los datos serán divididos. Una posición del centro del grupo está especificada por el vector de peso conectado a la unidad de salida correspondiente. Por ejemplo la red en la Figura 21, los datos de entrada tri-dimensionales son divididos en cuatro grupos, y los centros de los grupos, denotados como los pesos, son actualizados vía la regla de aprendizaje competitivo.

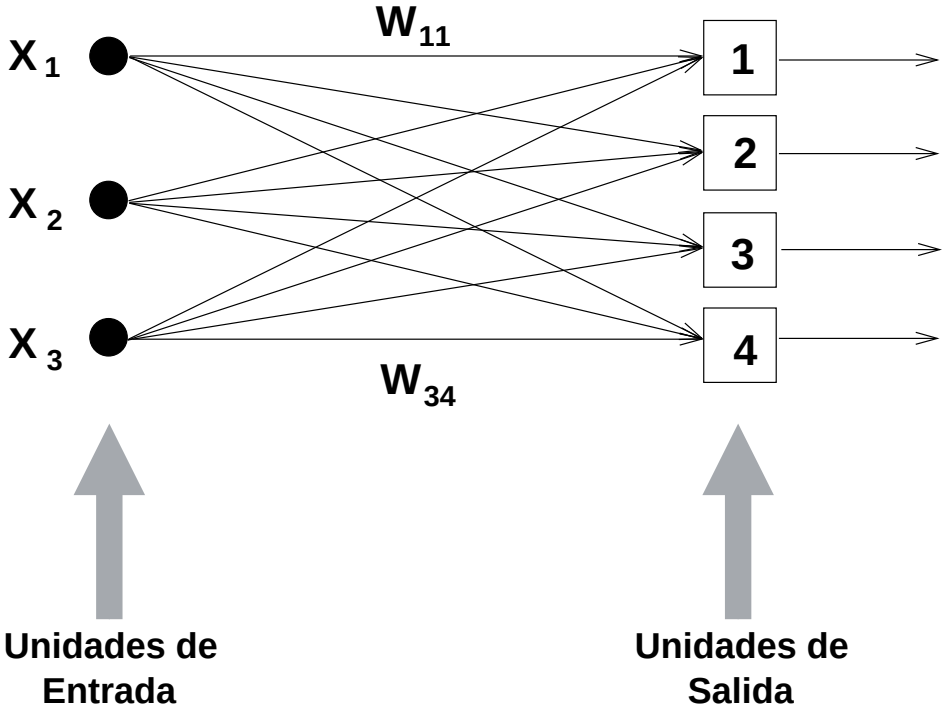


Figura 21: Red de aprendizaje competitivo.

El vector de entrada x=$[x_1, x_2, x_3]^T$ y el vector de peso w$_j$ = $[w_{1j}, w_{2j}, w_{3j}]^T$ para

una unidad de salida j en general se supone que están normalizados. El valor de activación a_j de la unidad de salida j es entonces calculado por el producto interno de los vectores de entrada y de peso:

$$a_j = \sum_{i=1}^{3} x_i w_{ij} = \mathrm{x}^T \mathrm{w}_j = \mathrm{w}_j^T \mathrm{x}. \tag{40}$$

A continuación, la unidad de salida con la activación más alta debe seleccionarse con el fín de realizar más procesamiento el cual está implicado en la *competitividad*. Asumiendo que la unidad de salida k tiene la activación máxima, los pesos dirigidos a ésta unidad están actualizados de acuerdo a la regla competitiva o a la llamada regla de aprendizaje "el ganador toma todo" como sigue:

$$\mathrm{w}_k(t+1) = \frac{\mathrm{w}_k(t) + \eta(\mathrm{x}(t) - \mathrm{w}_k(t))}{\| \mathrm{w}_k(t) + \eta(\mathrm{x}(t) - \mathrm{w}_k(t)) \|}. \tag{41}$$

El peso precedente de la fórmula de actualización incluye una operación de normalización para asegurar que el peso actualizado es siempre de norma 1. Notablemente, sólo los pesos de la salida ganadora k se actualizan y todos los demás pesos permanecen sin cambio.

La fórmula de actualización en la Ecuación (41) implementa un esquema secuencial para encontrar los centros de los grupos de un conjunto de datos del cual las entradas son de longitud igual a la unidad. Cuando una entrada x es presentada a la red, el vector peso más cercano a x gira hacia él. Consecuentemente, los vectores de peso se mueven hacia aquellas áreas donde más entradas aparecen y eventualmente los vectores peso se vuelven los centros de los grupos para el conjunto de datos.

Usar la distancia Euclidiana como una *medida de disimilitud* es un esquema más general de aprendizaje competitivo en el cual la activación de la unidad de salida j es

$$a_j = \left(\sum_{i=1}^{3} (x_i - w_{ij})^2 \right)^{0.5} = \| \mathrm{x} - \mathrm{w}_j \| . \tag{42}$$

Los pesos de la unidad de salida con la más pequeña activación son actualizados de acuerdo a

$$\mathrm{w}_k(t+1) = \mathrm{w}_k(t) + \eta(\mathrm{x}(t) - \mathrm{w}_k(t)). \tag{43}$$

En la ecuación anterior, los pesos de la unidad ganadora se mueven hacia la entrada x. En éste caso, ni los datos ni los pesos deben ser de longitud unidad.

Una red de aprendizaje competitivo realiza un proceso de agrupamiento en línea sobre los patrones de entrada. Cuando el proceso está completo, los datos de entrada se dividen en grupos disjuntos tal que las similitudes entre individuos en el mismo grupo son más grandes que aquellas en grupos diferentes. Aquí se introducen dos métricas de similitud: la medida de similitud del producto interno en la Ecuación (40) y la medida de disimilitud de la distancia Euclideana en la Ecuación (42). Obviamente, otras métricas pueden ser empleadas en su lugar, y diferentes selecciones llevan a diferentes resultados de agrupamiento. Cuando la distancia Euclideana es adoptada, puede ser probado que la fórmula de actualización en la Ecuación (43), es actualmente una versión en línea del descenso de gradiente que minimiza la siguiente función objetivo:

$$E = \sum_{p} \| \mathrm{w}_{f(\mathrm{x}_p)} - \mathrm{x}_(p) \|^2, \tag{44}$$

donde $f_{(\mathrm{x}_p)}$ es la neurona ganadora cuando la entrada x_p es tomada y $\mathrm{w}_{f_{(\mathrm{x}_p)}}$ es el centro de la clase a donde x_p pertenece.

Una larga familia de algoritmos de agrupamiento por lote (o fuera de línea) pueden ser utilizados para encontrar centros de grupos que minimicen la Ecuación (44). Uno

de estos algoritmos es el agrupamiento K-medias.

Una limitación del aprendizaje competitivo es que algunos de los vectores peso que son inicializados a valores aleatorios, pueden estar lejos de cualquier vector de entrada y, subsecuentemente, nunca consiguen actualizarse. Tal situación puede ser prevenida inicializando los pesos a muestras de los mismos datos de entrada, de ese modo asegurando que todos los pesos se actualizen cuando todos los patrones de entrada son presentados. Una alternativa sería actualizar los pesos de ambas unidades las ganadoras y las perdedoras, pero empleando una tasa de aprendizaje significativamente más pequeña η para las perdedoras. Esto es comunmente referido como *aprendizaje de goteo*.

Cambiar dinámicamente la tasa de aprendizaje η en la fórmula de actualización de pesos de las Ecuaciones (41) o (43) es generalmente lo deseado. Un valor grande inicial de η explora el espacio de datos ampliamente más tarde, un valor más pequeño cada vez refina los pesos. La operación es similar a la calendarización o al templado simulado. Por lo tanto, se utiliza una de las siguientes fórmulas para cambiar η:

$$\begin{cases} \eta(t) = \eta_0 e^{-\alpha t}, \text{ with } \alpha > 0, \text{ or} \\ \eta(t) = \eta_0 t^{-\alpha}, \text{ with } \alpha \leq 1, \text{ or} \\ \eta(t) = \eta(1 - \alpha t), \text{ with } 0 < \alpha < (\max\{t\})^{-1}. \end{cases} \tag{45}$$

El aprendizaje competitivo carece de la capacidad de agregar nuevos grupos cuando se considere necesario. Además, si la tasa de aprendizaje η es constante, el aprendizaje competitivo no garantiza estabilidad en la formación de los grupos. La unidad ganadora que responde a un patrón particular puede continuar cambiando durante el entrenamiento. Por otro lado, η, si decrece con el tiempo puede volverse muy pequeña para actualizar los centros de los grupos cuando nuevos datos de una naturaleza diferente de probabilidad sean presentados. Carpenter y Grossberg , 1988, se refieren a

tal ocurrencia como *dilema de estabilidad-plasticidad*, el cual es común en el diseño de sistemas de aprendizaje inteligente. En general, un agente aprendiz (o sistema) debería ser *plástico*, o adaptativo en reaccionar a ambientes cambiantes; mientras que, debería ser *estable* para preservar el conocimiento adquirido previamente.

Si las unidades de salida de un red de aprendizaje competitivo se arreglan de una manera geométrica (tal como en un vector uni-dimensional o un arreglo bi-dimensional), entonces se pueden actualizar los pesos de los ganadores tan bien como los de los vecinos perdedores.

Después de que el aprendizaje competitivo terminó; el espacio de entrada se divide en un número de grupos disjuntos, cada uno de los cuales está representado por un centro de grupo. Estos centros de grupos son también conocidos como plantillas, vector referencia, o vector código. Para un vector entrada, se puede utilizar la correspondiente plantilla para representar el vector de entrada más que al vector en si mismo. Tal enfoque es llamado *cuantización de vector* y ha sido empleado para compresión de datos en procesamiento de imágenes y sistemas de comunicación.

III.6.2 Aprendizaje Hebbian

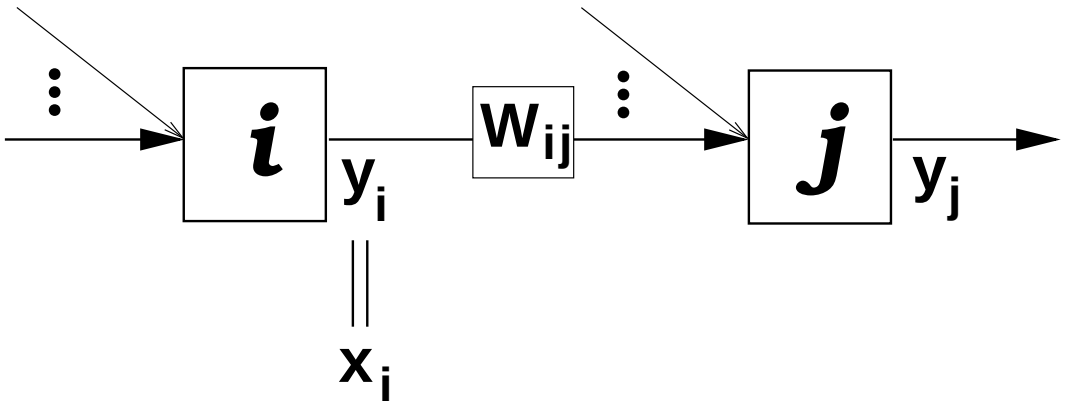


Figura 22: Topología de una red simple para aprendizaje Hebbian: el peso w_{ij} reside entre dos neuronas i y j.

Hebb , 1949, describió un método simple de aprendizaje de cambio del peso sináptico. Cuando dos células se disparan simultaneamente (i.e., tienen respuestas fuertes), su fuerza de conexión (o peso) se incrementa. Tal fenomeno es el tan llamado *aprendizaje Hebbian*, donde el que peso que se incrementa entre dos neuronas es proporcional a la frecuencia a la cual se disparan juntas. Entre varias fórmulas matemáticas de éste principio, la más simple es expresada como sigue:

$$\Delta w_{ij} = \eta y_i y_j, \tag{46}$$

donde η es la tasa de aprendizaje. Ya que los pesos son ajustados de acuerdo a la correlación de las neuronas de salida, la fórmula precedente es un tipo de *regla de aprendizaje correlacional*. La i-ésima neurona de salida y_i puede ser considerada como una entrada (x_i) a otra neurona j (Figura 22), de modo que la Ecuación (46) puede ser escrita como:

$$\Delta w_{ij} = \eta y_j x_i. \tag{47}$$

Reformulando, un peso se supone que cambia proporcionalmente con la correlación de las señales de entrada y salida. Utilizando una función de neurona $f(\cdot)$, y_j está dada por

$$y_j = f(\mathrm{w}_j^T \mathrm{x}).$$

Así, la Ecuación (47) es equivalente a la siguiente:

$$\Delta w_{ij} = \eta f(\mathrm{w}_j^T \mathrm{x}) x_i. \tag{48}$$

Una secuencia de patrones de aprendizaje indexados por p es asumido aquí para ser presentados a la red. En resumen, todos los pesos iniciales son cero. Usando la

Ecuación (46), la cantidad de actualización de un peso después de que el conjunto entero de datos es presentado será

$$\Delta w_{ij} = \eta \sum_{p} y_{ip} y_{jp}. \tag{49}$$

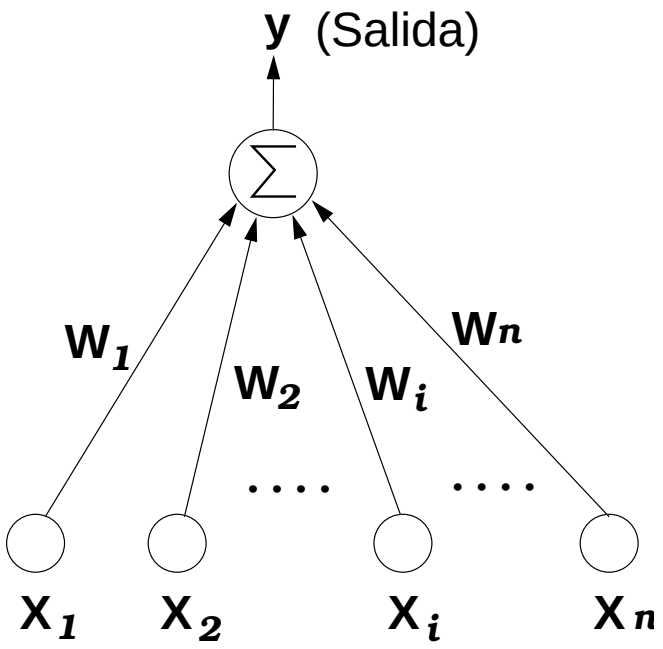


Figura 23: Red de una sola salida y una sola capa con aprendizaje Hebbian para análisis de componentes principales.

Los patrones de entrada frecuentes tienen el mayor impacto en los pesos y, eventualmente, causan que la red produzca las salidas más grandes. Aplicar el aprendizaje Hebbian sencillo dado en la Ecuación (46), causa un crecimiento sin restricciones de los pesos. Por eso, en algunos casos, la regla Hebbian se modifica para contener el crecimiento ilimitado de los pesos. La normalización de pesos es uno de tales métodos.

La racionalidad detrás de la regla de aprendizaje Hebbian es más facil de entender por medio de una red de una capa sencilla de n entradas y una salida con funciones de activación identidad, como se muestra en la Figura 23. La salida y es igual a $\sum_{i=1}^{n} w_i x_i$, o en forma de matriz,

$$y = \mathrm{w}^T\mathrm{x} = \mathrm{x}^T\mathrm{w}, \tag{50}$$

donde $\mathrm{x} = [x_1, \ldots, x_n]^T$ es el vector de entrada y $\mathrm{w} = [w_1, \ldots, w_n]^T$ es el vector de peso. La regla de aprendizaje Hebbian correspondiente será:

$$\Delta\mathrm{w} = \eta y \mathrm{x}. \tag{51}$$

La regla de aprendizaje anterior es un esquema en línea ascendente-descendente que minimiza la función objetivo siguente:

$$J = \frac{1}{2}\sum_p y_p^2 = \frac{1}{2}\sum_p (\mathrm{x}_p^T\mathrm{w})^2, \tag{52}$$

donde x_p y y_p son la p-ésima entrada y la salida deseada correspondiente, respectivamente. Si w es un vector unidad, entonces $\mathrm{x}_p^T\mathrm{w}$ es la proyección del vector x_p en la dirección especificada por un vector w. La minimización de J implica encontrar la w de longitud igual a la unidad que mejor aproxime la representación de la dirección del conjunto de datos entero.

Otros algoritmos de aprendizaje tales como en el aprendizaje de red Hopfield y el aprendizaje de correlación supervisado a menudo reflejan el principio de aprendizaje Hebbian. Todos estos comparten la propiedad correlacional de sus fórmulas.

Capítulo IV

Arquitectura Saphira para robots móviles autónomos

IV.1 Introducción

Con el proposito de que los robot móviles, puedan realizar tareas útiles y ser aceptados en ambientes abiertos, no cabe duda de que deben ser autónomos. Esto es: ser capaces de adquirir información y realizar tareas sin intervención humana. La autonomía tiene muchos diferentes aspectos; aquí nos concentramos en tres apectos centrales: la habilidad de atender a otro agente, tomar información acerca del ambiente, y llevar a cabo tareas asignadas. Las tres envuelven operaciones complejas de sensado y planeación por parte del robot.

IV.2 Agentes móviles autónomos

¿Cuáles son las capacidades mínimas para un agente móvil autónomo? Claramente, la pregunta es demasiado amplia. Así, es necesario conocer más acerca de la tarea y del ambiente. Otras preguntas complementarias podrían ser: ¿Qué se supone que el agente haga —tendrá un repertorio limitado de rutinas simples, o tendrá que encontrar la forma de cómo realizar asignaciones complejas? ¿Habrá que realizar ingeniería especial en el ambiente, o tendrá el agente que lidiar con un espacio no modificado? ¿Cómo será juzgado su desempeño? ¿Tendrá que interactuar con personas, y en qué manera?

IV.3 El robot servidor

La arquitectura Saphira está diseñada para operar como un robot servidor, esto es, una plataforma móvil que provee un conjunto de servicios robóticos en un formato estandar. Este paradigma cliente/servidor se sintetiza en Saphira el cual posee las particularidades de cualquier robot, lo cual nos permite portarlo fácilmente a diferentes robots que se adieran al protocolo cliente/servidor.

El robot servidor es responsable de controlar la operación de bajo nivel de los motores basados en mandos desde un cliente, y de operar los sensores y empacar los resultados para enviarlos de regreso al cliente. La Figura 24 muestra un sistema operativo servidor típico. En un rápido ciclo de interrupción, el servidor controla la energía de los motores y reviza las lecturas de los codificadores. En un ciclo más lento, maneja los sonares y cualquier otro dispositivo sensor o de comunicación. En lo alto de los controles de bajo nivel, el servidor implementa un conjunto de servicios básicos.

- Control de movimiento: Velocidad hacia delante-reversa, dirección ángular. El servidor tiene valores de control para velocidad y derección, y controla los motores en un rapido ciclo (10-50 ms) de tal manera que el robot mantiene los valores de control. El cliente envía, mandos para cambiar los valores de control, así controla el movimiento del robot en un ciclo más lento (100 ms).

- Integración de la posición. El servidor actualiza el cálculo muerto, interno de la posición del robot utilizando información de los codificadores.

- Sonar y otros sensores. El servidor maneja los sensores, controlando el rango de tiempo de los sonares, y cualquier otro dispositivo tal como una cámara pan/tilt. El cliente puede enviar mandos para cambiar el itinerario de los sensores.

- Comunicación. El servidor envía un paquete de información al cliente cada 100ms,

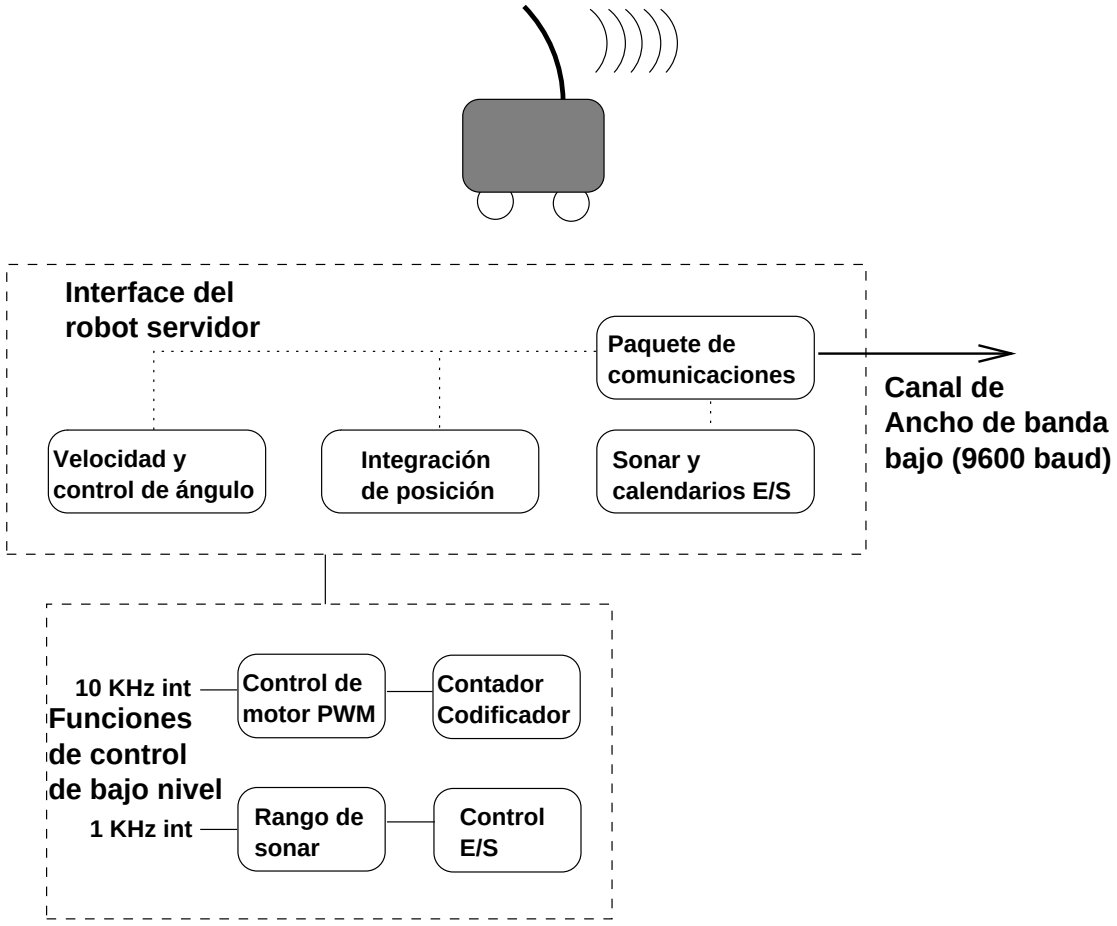


Figura 24: Sistema operativo servidor. Los servicios básicos incluyen control de motor, integración de posición, y control de sensor. Estos servicios son derivados de actuadores de motor y sensores en el robot.

conteniendo información de la posición, velocidad, y lecturas de los sensores. Recibe mandos del cliente para actualizar sus valores de control e itinerarios de los sensores.

Mediante el uso de valores de control, el servidor puede comunicarse con un cliente sobre un canal con ancho de banda bajo y poca latencia, ya que el servo control de alta velocidad de motores y sensores se ejecuta localmente. Por ejemplo, se ha logrado controlar exitosamente un robot ejecutando Saphira en una máquina huesped remota con una conexión telefónica al lugar del robot servidor.

Con sensores de visión, la cantidad de información procesada requerida implica que, o todo el procesamiento debe ser hecho a bordo, o que el ancho de banda debe ser incrementado. En este trabajo se pretende utilizar el primer método, produciendo información para la evaluación de obstáculos.

La mayoría de las plataformas de robots móviles pueden soportar el tipo de operaciones de servidor especificadas en el protocolo. Hasta ahora, Saphira ha controlado cuatro diferentes plataformas de robot.

- Flakey (SRI Internacional). Este es una plataforma común de robot móvil construída en 1984. Su forma es un cilíndro octagonal, con aproximadamente 0.5 metros de diámetro y un metro de alto. Dos ruedas de propulsión de 7" de diámetro están localizadas en los lados. Además, tiene ruedas pequeñas que son para lograr un balance enfrente y atrás. Los sensores de Flakey incluyen un anillo de 12 sonares en la base, y un par de cámaras estéreo montadas en una cabeza pan/tilt. Flakey también tiene un sistema de reconocimiento de voz contínuo con altavoz independiente llamado CORONA, desarrollado en SRI, y un programa texto-discurso estandar para hablar. La interface de voz está integrada como parte del cliente saphira, más que del robot servidor.

- Erratic y Pioneer (SRI International and Real World Interface, Inc.). Erratic es una versión más pequeña de Flakey con el mismo diferencial y sonares, pero sin las capacidades de visión. Pioneer es una versión comercial del Erratic hecha por Real World Interface, Inc.

- Khepera (Swiss Federal Institute of Technology). El Khepera en un pequeño robot con sólo 2 pulgadas de ancho, con ruedas diferenciales y sensores infra-rojos (IR) de proximidad. También tiene modulos adheribles para visión y agarre.

- B14/B21 (Real World Interface, Inc.) Esta investigación es de proposito comercial enfocada a, robots móviles con un amplio rango de capacidades sensoriales, incluyendo sonares, IR de próximidad, y sistemas de visión. En lugar del manejo diferencial, tiene un sistema de micro-manejo en el cual tres ruedas de manejo voltean al unísono.

El cliente Saphira, que controla la operación de alto nivel del robot, puede existir incluso en el robot mismo, o en una computadora huesped que no esté a bordo del robot. Los robots más pequeños tienen típicamente espacio sólo para el servidor, y el cliente Saphira corre sobre otra computadora huesped que no se encuentra a bordo. El pioneer es un caso intermedio interesante: Saphira se puede ejecutar sobre una laptop encima del robot, o en una computadora huesped que no esté a bordo, conectada por medio de un radio modem.

En el Flakey y el B14/B21, todos los sistemas corren a bordo. Flakey tiene 2 procesadores con configuración Sparc-station, con un procesador dedicado al servidor, y al sistema de visión el otro procesador se encarga de Saphira y el sistema de entendimiento de frases habladas.

IV.4 La arquitectura Saphira

La arquitectura Saphira es un sistema integrado de control y sondeo para aplicaciones para robótica. En el corazón del sistema se encuentra el Espacio Perceptual Local EPL (Figura 25) y está diseñado para acomodar varios niveles de interpretación de la información del sensor, también como información a priori de fuentes como mapas. Actualmente, las tecnologías importantes más representativas son:

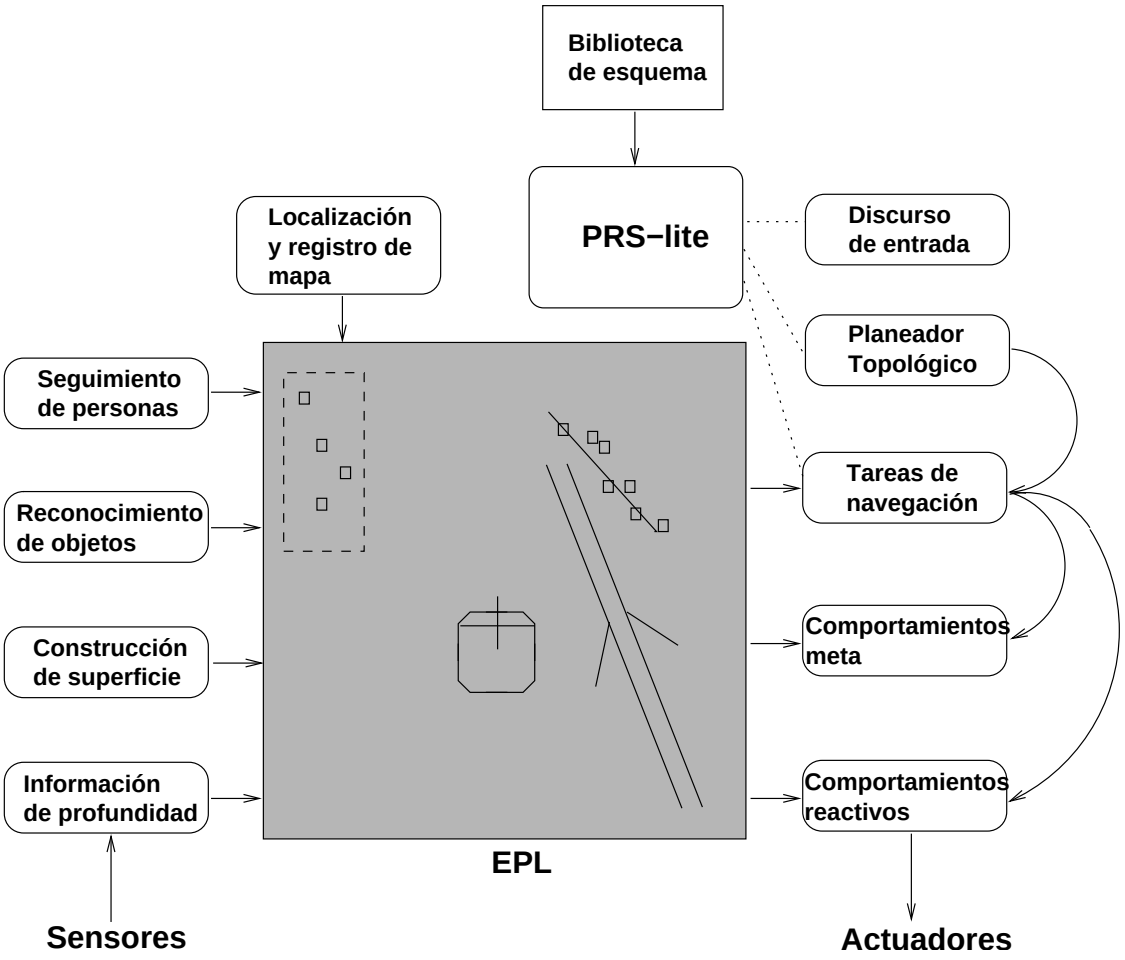


Figura 25: Arquitectura del sistema Saphira.

- Una representación basada en malla similar a las mallas de ocupación de Moravec y Elfe, construida de la fución de lecturas del sensor.

- Representaciones más analíticas de las características de la superficie, la cual interpreta datos de sensor relativos a modelos del ambiente.

- Descripciones semánticas del mundo, usando estructuras tales como corredores o puertas (artefactos). Los artefactos son el producto de la interpretación abajo -arriba de las lecturas del sensor, o del refinamiento de arriba a abajo de la información del mapa.

El EPL da al robot una conciencia de su ambiente inmediato, y es crítico en la tarea de fusionar información de sensor, la planeación de movimiento local, y la integración de la información del mapa. El EPL proporciona a la arquitectura Saphira su coherencia en la representación. Como se ha podido apreciar, la interacción de las lecturas de sensor e interpretaciones que toman lugar aquí, permite a Saphira coordinar constantemente sus nociones internas del mundo con sus impresiciones sensoriales. Se puede pensar en los artefactos como si fueran suposiciones que hace Saphira acerca del mundo, y la mayoría de las acciones son planeadas y ejecutadas con respecto a estas suposiciones.

En los términos de Brooks, la organización es parcialmente vertical y parcialmente horizontal. La organización vertical ocurre en ambas, las rutinas perceptuales están a la izquierda, las rutinas de acción están a la derecha. La dimensión vertical nos da una indicación del nivel de procesamiento, con comportamientos de alto nivel y rutinas perceptuales en la parte superior. El control es coordinado por el PRS-lite, el cual instancia rutinas para navegación, planeación, ejecución de monitoreo, y coordinación perceptual. Varias rutinas perceptuales son responsables de recabar información de los sensores al EPL y procesar dicha información con el fin de producir información de la superficie que puede ser utilizada por el reconocimiento de objetos y rutinas de navegación. En el lado de la acción, los comportamientos de más bajo nivel, se

ocupan de la información del terreno para hacer evasión de obstáculos. Los bloques básicos de construcción de comportamientos son reglas difusas, las cuales proporcionan la habilidad al robot de reaccionar con gracia al ambiente graduando la fuerza de la reacción (e.g. voltear a la izquierda) de acuerdo a la fuerza de los estímulos (e.g. distancia de un obstáculo a la derecha).

Comportamientos más complicados que realizan acciones dirigidas a metas son utilizados para guiar los comportamientos reactivos, y utilizar información de la superficie y artefactos al EPL como puntos de control para movimiento. En éste nivel, las reglas difusas se mezclan, posiblemente para resolver propósitos conflictivos dentro de una secuencia de acción suave. Finalmente, al nivel de tarea de los comportamientos complejos son secuenciados y su progreso es monitoreado a través de eventos en el EPL. La organización horizontal se desarrolla con el proposito de que los comportamientos puedan resolver la información apropiada en el EPL. Comportamientos que son de tiempo crítico, tal como la evasión de obstáculos, dependen más de un procesamiento en los sensores el cual es muy simple con objeto de que esten disponibles rápidamente. Sin embargo, estas rutinas pueden también hacer uso de otra información cuando está disponible, e.g. información a priori del mapa acerca de obstáculos esperados.

IV.4.1 Comportamientos

Al nivel de control, la arquitectura Saphira está basada en el comportamiento. El problema de control se descompone en pequeñas unidades de control llamadas comportamientos básicos, como evasión de obstáculos o seguimiento de un corredor. Unas de las características distintivas de Saphira es que los comportamientos están escritos y combinados empleando técnicas basádas en lógica difusa. La Figura 26 muestra los principales componentes del procesado de comportamientos en Saphira. Cada comportamiento contiene una función para calcular variables difusas apropiadas desde el EPL,

y un conjunto de reglas que representan el grado de preferencia de ciertas acciones. Las salidas de los comportamientos están combinadas y un valor de control es escogido para cada canal de control.

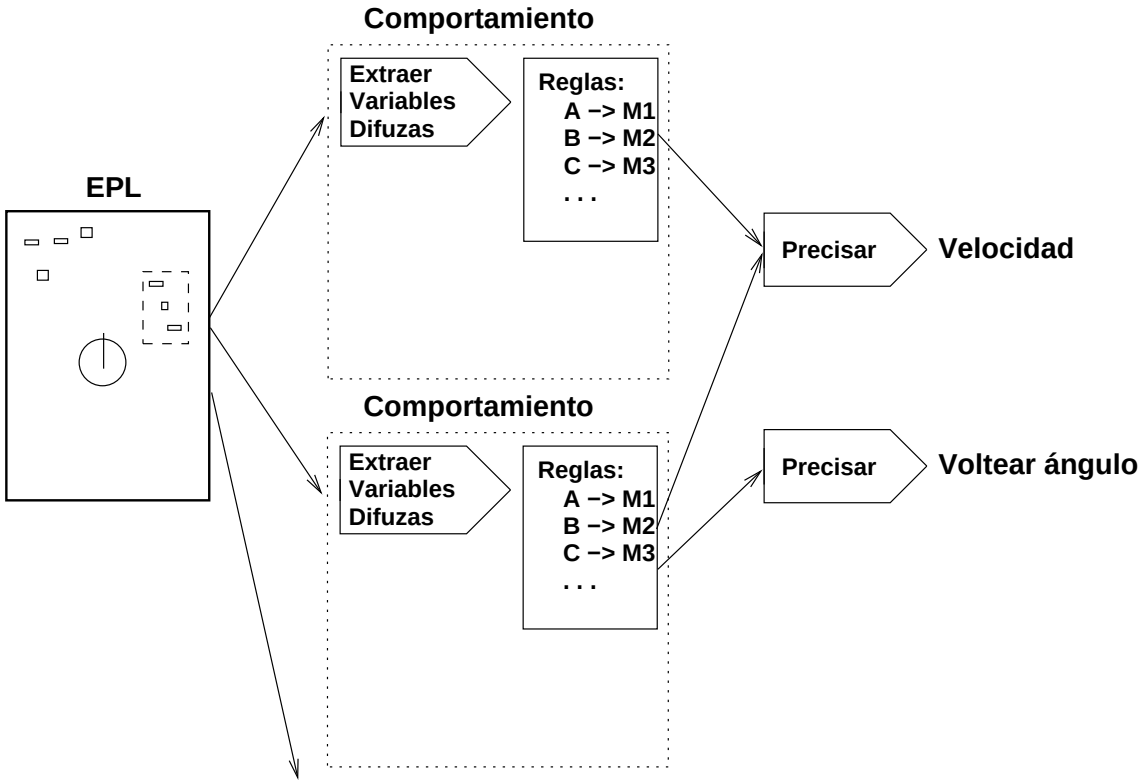


Figura 26: Comportamientos difusos.

Cada comportamiento consiste de una función de actualización y un conjunto de reglas difusas. El propósito de la función de actualización es extraer información del EPL y convertirlo en un conjunto de variables difusas apropiado para el comportamiento. Por ejemplo, un comportamiento de evasión de obstaculos podría tener las siguientes variables, indicando donde la trayectoria del robot está bloqueada:

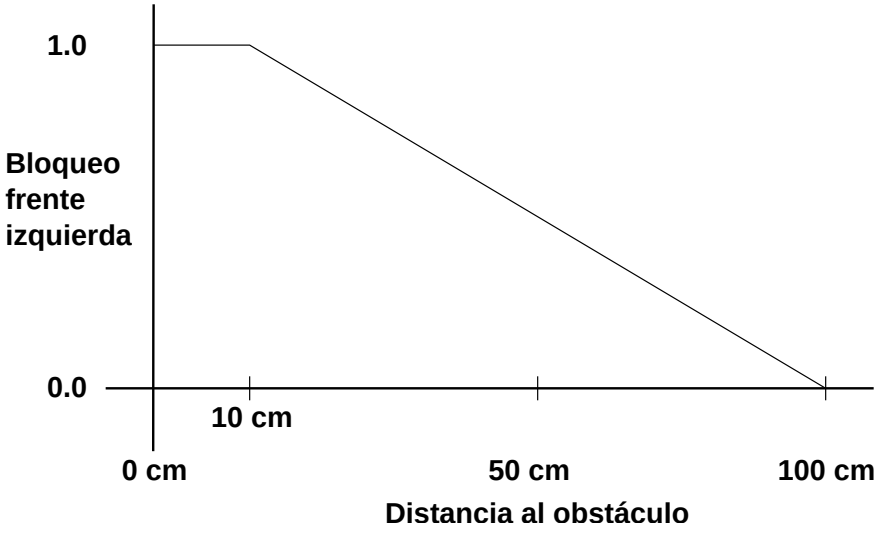
front - left - blocked

front - right - blocked

side - left - blocked

side - right - blocked

Cada variable difusa toma un valor del intervalo [0...1], indicando el grado en el cual ésta condición se mantiene. Las variables difusas son calculadas utilizando una función de transferencia del estado del EPL al de la variable. Suponiendo que la parte izquierda del frente del robot está completamente bloqueada si hay un obstáculo 10 cms enfrente, y desbloqueada si no hay nada 100 cms enfrente. En medio de estos dos criterios sería adecuado definir la variable difusa de *front - left- blocked* la cual podría ser representada por la siguiente función de tranferencia en el rango [0,1]. Su función de tranferencia luce como sigue:



Las variables difusas son las entradas al conjunto de reglas de control difusas de la forma

$$A \longrightarrow C,$$

donde A es una fórmula difusa compuesta por predicados difusos y las conectivas difusas AND (y), OR (o) y NOT (no), y C es una acción de control. Por ejemplo, las reglas para evasión de obstáculos podrian ser escritas como:

front-left-blocked AND (NOT *front-right-blocked*) AND (NOT *side-right-blocked*) $\longrightarrow$ turn right

front-right-blocked AND (NOT *front-left-blocked*) AND (NOT *side-left-blocked*) $\longrightarrow$ turn left

front-right-blocked OR *front-left-blocked* $\longrightarrow$ slow down

Note que éstas reglas se refieren a la dirección y a la velocidad del robot. En general, el robot puede tener muchos canales de control, cada uno con la habilidad de realizar alguna función de movimiento. Flackey como todo robot servidor que utiliza Saphira tiene la habilidad de voltear a una dirección y mantener una velocidad; otras funciones pueden incluir agarradores o manipuladores, cabezas pan/tilt para una cámara, y más. un comportamiento puede afectar tantos de esos canales de control como necesite, mientras deje los otros solos.

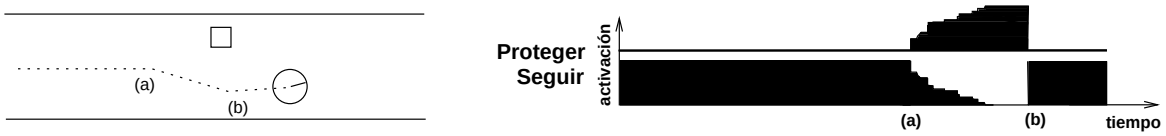


Figura 27: Mezclado dependiente del contexto de los comportamientos seguir y evadir.

Los resultados de la aplicación de una regla es una función de conveniencia, la cual dice al comportamiento cuanto le ayuda un valor de control en particular. Para cada canal de control, los resultados de todas las reglas de comportamiento para ese canal son promediadas para obtener el valor de control final. El promedio incluye una función de peso básada en 2 propiedades: prioridad de un comportamiento y su contexto de activación. A los comportamientos que son más importantes se les da una mayor prioridad, e.g., la evasión de obstáculos tiene una prioridad mayor comparada con seguir un muro, ya que no se desea que el robot tenga alguna colisión. Sin embargo, dejar que el comportamiento de evasión de colisiones tenga un control completo sobre el robot, incluso cuando no hay obstáculos, no es razonable ya que en éste contexto es el seguir el muro el que debe tomar precedencia. Así que mientras las prioridades

están determinadas, el contexto de cambios de activación, y los controles de los comportamientos son mezclados de acuerdo a la prioridad del comportamiento y cómo está activado. Llamamos a éste balanceo *mezcla dependiente del contexto*, ya que toma en cuenta cuándo debería responder un comportamiento en un contexto particular. La Figura 27 muestra un ejemplo de mezcla dependiente del contexto en operación, en el caso particular en el que el robot sigue un muro y evade un obstáculo. Después del punto (a), el comportamiento Proteger comienza por activarse, y su más alta prioridad domina al comportamiento Seguir. En el punto (b), el obstáculo se evade, y Seguir se reanuda.

La mezcla dependiente del contexto ha probado ser una una técnica efectiva para coordinar comportamientos reactivos y orientados a metas. Los comportamientos no sólo se encienden y apagan. Además, sus preferencias están combinadas con respecto a su conveniencia. Una importante consecuencia es que las preferencias de los comportamientos de búsqueda de metas son considerados todavía durante maniobras reactivas guiando las opciones de control hacia el alcance de las metas. En el ejemplo anterior, se supone que el obstáculo está justo enfrente del robot (e.g., el robot termina volteando hacia la pared) y puede así ser evadido volteando a la derecha e izquierda. Entonces, el comportamiento combinado escoge el lado que fomente mejor el seguimiento del corredor.

Es interesante comparar el mezclado dependiente del contexto con la técnica del campo potencial artificial, introducida primero por Khatib , 1986, y ahora extensivamente utilizada en el dominio robótico. En el enfoque del campo potencial, una meta es representada por un potencial, midiendo la conveniencia de cada estado desde el punto de vista de la meta. Por ejemplo, la meta de evadir obstáculos es representada por un campo potencial teniendo valores máximos alrededor de los obstáculos; y la meta de alcanzar determinada ubicación es representada por un campo teniendo valores minimos

en esa ubicación. En cada punto, el robot responde a una pseudo-fuerza proporcional al vector gradiente del campo. Aunque hay algunas diferencias técnicas, en general el enfoque de regla difusa puede también ser visto como generando un gradiente de conveniencia mostrando la mejor dirección para que el robot se mueva en cada punto en el campo. La principal diferencia, sin embargo, está en cómo el problema de control complejo es descompuesto. El enfoque de control difuso permite partir las acciones de control de acuerdo a comportamientos deseados del robot, y entonces combinar los comportamientos por medio del mezclado dependiente del contexto. En contraste, el método de campo potencial plantea el reto de cómo plantear un campo potencial global para alcanzar un resultado particular, conceptualmente una tarea mucho más dura. De modo interesante, el enfoque de esquema de motor trae una orientación de comportamiento a campos de potencial, descomponiendo el campo potencial global en un conjunto de subcampos, cada uno dispuesto para alcanzar una meta particular.

IV.4.2 Coherencia

Comportamientos reactivos tales como evasión de obstáculos a menudo pueden tomar su entrada directamente de las lecturas del sensor, quizas con alguna transformación y filtrado. La mayoría de los comportamientos dirigidos a metas pueden a menudo beneficiarse del empleo de artefactos, representaciones internas de objetos o configuraciones de objetos. Esto es verdadero cuando los sensores proporcionan solamente información esporádica e incierta acerca del ambiente. Por ejemplo, al seguir un robot un corredor no será capaz de sondear el corredor con sus sonares laterales cuando atraviese puertas abiertas o uniones de pasillos. Sería peligroso suspender el comportamiento en este punto, ya que sobre una distancia pequeña el cálculo muerto del robot es suficientemente bueno para seguir un "corredor virtual" hasta que la abertura haya pasado.

En otras situaciones un artefacto puede representar una entidad geométrica artificial

que guíe el comportamiento. Tal situación ocurre frecuentemente en la navegación humana, e.g., al cruzar una calle uno tiende a permanecer dentro de un sendero definido por la banqueta en cada lado, incluso cuando no hay pintado un cruce de peatones. De forma similar, en el comportamiento para seguir un corredor, el robot se guía por un artefacto sendero que está posicionado a un pie de los muros.

De acuerdo con estas estrategias de comportamiento, los artefactos en Saphira vienen de 3 fuentes:

- De información a priori. Típicamente, el robot iniciará con un mapa de los corredores y oficinas en su ambiente.

- De características perceptuales. Cuando un proceso perceptual reconoce un objeto nuevo, puede agregar ese objeto a la lista de artefactos.

- Indirectamente, de otros artefactos o información de la meta. Por ejemplo, si el usuario proporciona el mando, "Mover 3 metros adelante", un artefacto meta se crea en una posición tres metros en frente del robot.

Los artefactos siempre tienen una clase (MURO, CORREDOR, POSICIÓN, etc.), información geométrica acerca de su posición y extención, y una identidad única, de tal suerte que dos artefactos en la misma clase no sean el mismo. Pueden tener información acerca de dependencias geométricas con otros artefactos.

Normalmente, los artefactos en el EPL, se actualizan basados en el mecanismo del cálculo muerto del robot, el cual es confiable sólo bajo distancias cortas. La coherencia es la propiedad de actualizar las posiciones de los artefactos en el EPL basados en la percepción, de manera que el modelo del ambiente del robot permanece registrado con la posición del robot cuando este se mueva. Para estender cómo trabaja ésto, nos referimos a la siguiente relación:

característica $\Longleftrightarrow$ objeto hipótesis $\Longleftrightarrow$ artefacto

Las características están basadas en información del sensor, usualmente representando información de la superficie. Una característica típica sería una superficie lineal, representada por un segmento recto en el EPL.

Un objeto hipótesis es un conjunto de características que podrían corresponder al objeto del mundo real. Por ejemplo, dos rupturas en una superficie lineal podrían ser una puerta, y esas dos características podrían estar agrupadas con el fin de formar una hipótesis que defina una puerta. Las hipótesis objeto no tiene ninguna identidad particular, i.e., una hipótesis de puerta no tiene información de que es la puerta de una oficina en particular.

Los tres tipos de representación —características, objetos hipótesis y artefactos— coexisten en el EPL. Como el diagrama implica, no hay una relación estricta en cómo uno es creado y manipulado por el otro. Dependiendo de la tarea, es posible ir en muchas direcciones diferentes. En la construcción del mapa, por ejemplo, las características se agrupan en objetos hipótesis, los cuales son reconocidos como artefactos. Aunque este proceso es en general de abajo para arriba, hay tambien un importante componente de arriba hacia abajo, en el cual artefactos previamente reconocidos se comparan contra las hipótesis actuales, y sólo esas hipótesis las cuales se refieren a posibles nuevos objetos tienen el estatus de artefactos.

En la práctica, se ha encontrado que la introducción de artefactos simplifica mucho el diseño de comportamientos, permitiendo desacoplar el problema de control de los problemas al interpretar datos con ruido del sensor. La metodología para escribir comportamientos ha sido primero escribir pequeños conjuntos de reglas para tipos de movimientos elementales basados en artefactos simples, como seguir una línea, o alcanzar una localización; y entonces enfocarse en las estrategias para mantener estos artefactos anclados a las características correctas en el ambiente. Los comportamientos

resultantes a menudo se prueban para ser más robustos que controladores puramente reactivos.

En este documento no podemos describir todas las maneras en que la conexión de la característica al artefacto se realiza. Nos concentraremos en dos áreas: extracción de características de información perceptual, y el proceso de anclaje, en el cual los artefactos actuales se mantienen de forma coherente con el ambiente con el proposito de compararlos contra la características u objetos hipótesis.

Extracción de características

Para navegar a través de regiones extendidas, Saphira utiliza un mapa global que contiene un conocimiento impreciso de objetos en el dominio, especialmente paredes, puertas y uniones de corredores. Usar un mapa depende de la extracción confiable de información del objeto de señales perceptuales. Se gastaron muchos años tratando de producir una interpretación de objetos a partir de señales altamente inciertas de sonares y del par estereo. El mejor método que se ha encontrado es utilizar lecturas de sonar de abertura extendida, quizás aumentando con información de profundidad de un sistema estereo. Conforme Flakey se mueve, las lecturas de sus sonares laterales son acumulados como una serie de puntos representando posibles superficies a un lado del robot. Esto da a la resolución de un sensor una larga abertura a lo largo de la dirección de movimiento. Corriendo un algoritmo de características lineal robusto, podemos encontrar segmentos de pared y puertas con algún grado de confianza.

La útilidad de ésta técnica se basa en que permite la medición de características lineales confiables casi sin falsos positivos. Es difícil tratar con falsos positivos, porque usarlos para localización pone al robot en la posición incorrecta dentro de su mapa interno, y las comparaciones subsecuentes contra las características fallarán.

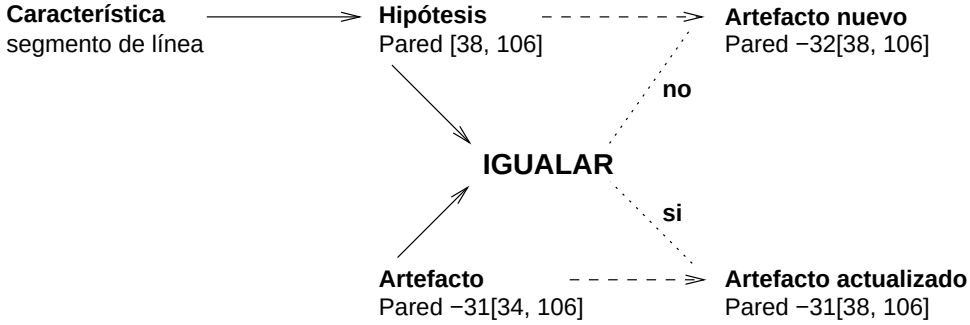
Extraer características de paredes y puertas hace facíl construir una mapa global

automáticamente, explorando un área. El mapa es impreciso porque hay un error en el sistema de cálculo muerto (cálculos anteriores), y porque los modelos para objetos espaciales es lineal, e.g., los corredores, están representados como 2 líneas rectas paralelas. Cuando las características que se construyen pueden ser combinadas en objetos hipótesis, las cuales son comparadas contra artefactos actuales, y promueven a nuevos artefactos cuando no son iguales. En la práctica, es posible construir un mapa de corredores.

Anclaje

Los artefactos existen como representaciones internas del ambiente. Cuando los objetos físicos que un artefacto refiere son percibidos por los sensores, podemos emplear esa información para actualizar la posición del artefacto con respecto al robot. Esto es necesario para garantizar ese comportamiento haciendo que el artefacto opere con respecto al objeto actual, más que con respecto a una suposición a priori. Le llamamos anclaje al proceso de (1) igualar una característica u objeto hipótesis a un artefacto, y (2) actualizar el artefacto utilizando ésta información perceptual.

En Saphira, la estructura de tomar una decisión para el problema de anclaje toma la siguiente forma:



Cuando las características son percibidas, Saphira intenta convertirlas a objetos hipótesis, ya que estas se igualan de una forma más confiable que las características individuales. Estas hipótesis son igualadas contra artefactos que existen en el EPL. Si

se igualan a un artefacto, la igualación produce información para actualizar (anclar) la posición del artefacto. Si no, son candidatas para su inclusión como nuevos artefactos en el mapa.

Si un artefacto que está a la vista de aparatos perceptuales no puede ser igualada a un objeto hipótesis, entonces Saphira trata de igualarlo a características perceptuales individuales. Esto es útil, por ejemplo, cuando el robot está recorrriendo un pasillo e intenta voltear hacia una puerta. Sólo un extremo de la puerta es inicialmente encontrado porque el otro extremo no está a la vista de los sonares laterales. Esta información es suficiente para anclar el artefacto puerta, y permite al robot proseguir con el comportamiento de atravezar la puerta.

Los artefactos que no tienen soporte perceptual cuando deberían (i.e., cuando están en el rango de las cámaras o sonares), son candidatos para retirarlos del EPL. En general el proceso de retirar debe tomar en cuenta el ambiente del robot. Por ejemplo, paredes y corredores son estructuras puestas que siempre están presentes, mientras que las puertas pueden estar cerradas (y por lo tanto no ser mostradas como características). Los candidatos más obvios para retirar son objetos transitorios como obstáculos. Hoy en día no se dispone de algúna teoría general de cómo representar el aspecto dinámico de objetos y su relación con reconocimiento de artefactos, pero depende de algoritmos de proposito general para cada tipo de objetos.

La Figura 28 muestra un ejemplo de anclaje en Saphira. La figura muestra el EPL en 2 momentos concecutivos durante el seguimiento del corredor. El comportamiento de seguir el corredor actúa con respecto a un artefacto corredor, representado por las 2 líneas dobles. Este artefacto está inicialmente alojado en los niveles de planeación del mapa (a). Note que la posición del artefacto no corresponde a la de la pared actual, el cual se visualiza por los conjuntos de lecturas de sonar (puntos pequeños). Esto puede ser porque el mapa es incorrecto o, más comunmente, por la inexactitud de

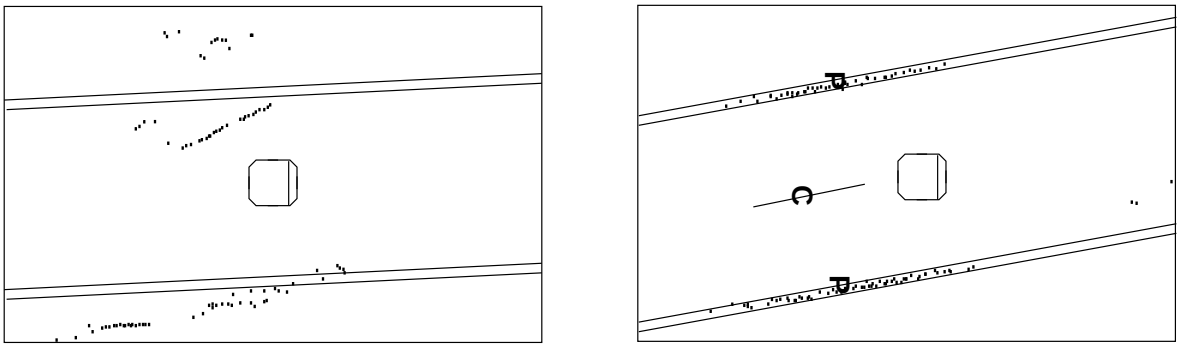


Figura 28: Anclaje de un artefacto corredor a las lecturas del sensor para el seguimiento del corredor.

la auto-localización. Dado que suficientes lecturas de sonar están juntas, las rutinas perceptuales de Saphira infieren la existencia de 2 paredes paralelas (marcadas con "P") y por ello forman un corredor hipótesis ("C"). El artefacto es entonces anclado a ésta hipótesis (b), y el movimiento procede ahora con respecto al actual corredor. El anclaje provee una respuesta de ciclo cerrado cuando datos relevantes del sensor están disponibles. Cuando los datos no están disponibles, e.g.,si las paredes se vuelven poco claras, los artefactos actúan como suposiciones para el movimiento.

Actualmente, el anclaje se utiliza exitosamente por comportamientos individuales o esquemas de activación que controlan el comportamiento. Por ejemplo, se utiliza por el comportamiento atravezar puerta para seguir la localización de la puerta conforme se mueve hacia dentro o fuera del cuarto. Es también empleado para mantener al robot globalmente registrado con respecto al mapa. El proceso de registro es suficientemente robusto que el mapa original, realizado por el robot, no tiene que ser muy preciso (y ésto no puede realizarse, si el reconocimiento muerto es pobre). El proceso de anclaje mantendrá la posición del robot actualizada con respecto a objetos relevantes en el ambiente. Esto asume, por supuesto, que el robot puede encontrar e igualar objetos de la clase correcta. Por ejemplo, en el caso en el que el robot recorra un corredor

largo sin características, el robot permanecerá correctamente registrado en el centro del corredor, pero el error en su posición longitudinal crecerá. Por eso es importante encontrar puertas o brechas en el corredor a intervalos razonables. Utilizando la estrategia de anclaje en corredores, puertas y uniones, se ha sido capaz de mantener al robot localizado en periodos largos arbitrariamente (i.e., hasta que las baterias se descargan).

Así la ayuda del anclaje que sirve para corregir información incierta anterior utilizando percepción, y mantiene al robot localizado en el mapa mientras navega. Pero hay todavía algunos problemas que no se han tomado en cuenta con éste esquema. Uno es que las percepciones del sonar son, en general, no adecuadas para mantener al robot localizado en ambientes complejos. Por ejemplo, en cuartos poblados, el robot rápidamente se confunde por que no puede encontrar un conjunto suficientemente largo de segmentos lineales para igualar. Para resolver este problema, se han considerado algoritmos basados en correlación usando el sistema de visión estereo. Manteniendo un seguimiento de un número de pequeños parches de superficie que son suficientemente distinctivos como características, esperamos ser capaces de usar los mismos algoritmos de registro para localizar confiablemente al robot en ambientes de interiores y exteriores complejos.

Un segundo problema es el reconocimiento de lugares. La localización cuando el robot no tiene conocimiento de su posición actual.

Seguimiento de personas

Saphira incorpora un simple seguidor de personas basado en información estereo. El algoritmo de seguimiento no realiza correlaciones sobre marcos sucesivos para mantener el seguimiento del objeto. Además, mira objetos parecidos a personas en cada marco sucesivo sin referencia a los anteriores, formula una hipótesis, y entonces la pasa a las

rutinas de registro del EPL. Los objetos parecidos a personas son detectados por un proceso de comparación corriendo sobre los datos estereo. Tres bandas de rango de información como altura del torso se examina por un perfil de forma, con respecto al ancho de un adulto. Si dos de las tres bandas concuerdan, entonces una persona es detectada en el centro de la forma. Como estamos solamente empleando una porción de la información estereo, podemos ejecutar el seguidor de personas a una taza de 10 Hz.

El algoritmo de registro toma una decisión acerca de cuando la persona es la misma que una representada por un artefacto, y también actualiza la posición del artefacto o crea uno nuevo. Nosotros mantenemos un modelo de velocidad simple de artefactos, el cual ayuda al proceso de registro a tomar mejores desiciones de igualación. También es utilizado por un proceso de centrado el cual puede mantener en el centro de la cámara mirando a la persona que está siendo seguida. Si un artefacto persona no recibe soporte por un periodo de tiempo, se remueve del EPL.

De forma interesante, el proceso de registro hace posible seguir a la persona al rededor de esquinas y a través de puertas, donde ellos pueden estar temporalmente perdidos de vista. El artefacto representa su más probable posición, y las rutinas de visión se mantienen buscando en esta área hasta que la persona es re-adquirida, o no puede ser encontrada.

Una limitación del presente sistema es su inhabilidad de distinguir diferentes personas. El rango de resolución del sistema no es sufucientemente vasto así es que distinguir personas por su forma no es posible, incluso sin la complicación del movimiento, posición del brazo y otras.

IV.4.3 Control ejecutivo: PRL-lite

Los comportamientos proveen un control de bajo nivel para las acciones físicas efectuadas por el sistema. Sobre ese nivel, hay una necesidad de referirse a comportamientos para metas y objetivos especificos que el robot debe emprender. Este proceso administrativo involucra determinar cuando activar/desactivar comportamientos como parte de la ejecución de una tarea, también cómo coordinarlas con otras actividades en el sistema. PRS-lite es un controlador reactivo basado en el Sistema de Razonamiento Procedural (PRS-CL) que realiza este proceso dentro de Saphira.

Muchas de las principales capacidades provistas por PRS-lite son compartidas por la mayoría de los controladores de tareas de la presente generación. Estos incluyen la integración suave de actividades de manejo de metas y manejo de eventos, responde puntualmente a cambios inesperados en el mundo, y la descomposición jerárquica de tareas. Algunas capacidades adicionales, no obstante, distinguen al PRS-lite de otros sistemas de control reactivo (incluyendo el PRS-CL). Estos incluyen el manejo de procesos continuos de interacción, un rico conjunto de mecanismos de control declarativo, y una semántica de metas generalizada que reemplaza los conceptos de exito o fallo con niveles de satisfacción meta.

Vista general

La representación básica del PRS-lite es el esquema de actividades, una especificación parametrizada de conocimiento procedimental para alcanzar un objetivo determinado. Este conocimiento procedural es representado como una lista ordenada de conjuntos meta cuya sucesiva realización permitirá conseguir el objetivo del esquema. Un conjunto meta está compuesto por una o más declaraciones meta (o metas), donde cada una consiste en un operador meta aplicado a una lista de argumentos. Un conjunto meta puede estar opcionalmente identificado por una etiqueta única en su esquema.

Intuitivamente, un conjunto meta corresponde a una secuencia ordenada de metas que están para ser alcanzadas como una unidad atómica. Un compilador transforma las especificaciones del esquema en máquinas parametrizadas de estados finitos (realizando optimizaciones cuando es apropiado), cuyos arcos están etiquetados con conjuntos meta individuales para llevarse a cabo. Los esquemas de actividades son lanzados tomando casos de sus parámetros e intentandolos dentro del sistema. Tales esquemas de instalación son referidos como intenciones. Multiples intenciones pueden estar activas simultáneamente, proporcionando una capacidad multitarea.

Diferentes modalidades metas son soportadas, como se resume en la Figura 29. Los tipos de meta pueden estar categorizados como acción o secuencia. Las metas de acción tienen su campo en funciones ejecutables (llamadas acciones primitivas), pruebas del estado del ambiente (representadas en el modelo interno del mundo), o en la activación/desactivación de intenciones y comportamientos. Esta última habilidad permite la descomposición jerarquica de metas. Las metas secuencia proveen la ejecución condicional de metas y ordenación sofisticada de estas más allá del procesamiento por omisión o el lineal, así como también varias formas de paralelismo.

Acciones Meta:

Prueba	revisar condición
Ejecutar	ejecutar una acción primitiva
=	asignación de variables locales
Esperar	esperar por una condición
Intentar	consignar intenciones (bloqueado/no bloqueado)
No intentar	terminar intenciones

Secuencias Meta:

Si	metas condicionales
Y	metas paralelas (con únion)
Dividir	metas paralelas (sin únion)
Ir a	bifurcarse a metas etiquetadas

Figura 29: Modalidades Meta del PRS-lite.

Sobre todo, PRS-lite puede ser empleado para generar y manejar un bosque de gráfos dirigidos cuyos nodos representan cada uno un conjunto meta de algún esquema de actividad. El nodo raíz de cada gráfo representa una meta del más alto nivel, con sus sucesores generados por el refinamiento jerárquico de metas. Nos referimos al conjunto de gráfos activos en un momento dado como las estructuras de intención actuales del sistema. Los nodos hoja de las estructuras de intención se les llama nodos actuales, y sus conjuntos meta asociados son llamados conjuntos meta actuales. Note que una estructura de intención puede tener múltiples nodos actuales por la inclución de metas secuencias paralelas en el esquema de definición del lenguaje.

Un ejecutor maneja intenciones en tiempo de ejecución. Repetidamente opera un ciclo corto de procesamiento en el cual considera los conjuntos meta actuales en cada estructura de intención. Realiza cualquier acción requerida para alcanzar sus metas constituyentes, y actualiza el conjunto de nodos actuales. La decisión de limitar el procesamiento a un solo conjunto meta por cada nodo hoja en una estructura intención, asegura toda la responsabilidad a nuevos eventos. Dada la granularidad de procesamiento, la responsabilidad del número de intensiones activas, el grado de paralelismo en esas intenciones, el tamaño de los conjuntos meta, y las acciones primitivas que están debajo y que son ejecutadas. El diseño ha probado ser adecuado para las tareas consideradas hasta la fecha.

IV.4.4 Modalidades meta

Las metas de la acción proporcionan las operaciones más básicas en el sistema. Las metas *Prueba* proveen la habilidad de probar las creencias acerca del estado actual del mundo externo. Dentro de Saphira, las creencias están caracterizadas por una combinación del espacio perceptual local y un conjunto de variables de ambiente. Ejecutar

metas proporciona la ejecución de acciones primitivas, las cuales pueden realizar contabilidad interna, el establecimiento de las variables de ambiente, o la realización de acciones externas especificas por el sistema. Las acciones externas para Flakey incluyen la generación de una expresión por el módulo de síntesis de discurso, y la invocación de un planeador de ruta. Las metas permiten la encuadernación de variables locales dentro de una intención.

Las metas *Intento* llevan a la activación de intenciones y comportamientos. Como tales, permiten la expansión jerárquica de intenciones a través de un refinamiento repetido de metas. Las metas *No-intento* proveen la habilidad complementaria para términar explicitamente las intenciones activas antes de que recorran su camino completo. Esta habilidad es crítica cuando opera en dominios dinámicos e impredecibles, donde el cambio rápido entre actividades es escencial. Las intenciones pueden tener asignadas prioridades que determinan el orden en el cual procederan a pasar al ejecutor. Las intenciones pueden también llamarse cuando se activen, permitiendoles ser referenciadas por otras intenciones (en particular, Metas No-intento).

Una característica crítica de Intento es que soporta las invocaciones de intenciones en modo no bloqueado o bloqueado. En modo no bloqueado, la inteción se activa y el control procede a la siguiente meta. En esencia, la intención no bloqueada es engendrada como un proceso independiente dentro del contexto de la intención padre; la intención no bloqueada persistirá hasta que se complete o su intención padre termine. En contraste, el modo de bloqueo deshabilita la actualización de la meta presente dentro de la intención padre hasta que el hijo se complete. Cualquier intención activada anteriormente por la intención padre continuará siendo procesada. Los grados de bloqueo son también soportados: una intención puede estar bloqueada hasta que un críterio de exito designado es satisfecho. Esta capacidad es valiosa para controlar comportamientos implementados como reglas difusas, las cuales proveen de una

métrica natural para definir grados de exito (particularmente, los predicados difusos que modelan el estado del mundo). Como una simple ilustración, Flakey tiene un comportamiento de dirección de cara que orienta al robot hacia una dirección determinada. Este comportamiento se invoca con diferentes umbrales en diferentes contextos, dependiendo de qué tan crítica es para ser precisamente orientado a esa dirección.

Las metas *Espera* permiten la suspención limitada de una intención hasta que cierta condición o evento ocurra. La modalidad de meta Espera es crítica en el marco en que permite la sincronización entre intenciones actuales a través del uso de variables compartidas.

Las metas *Secuencia* permiten una ordenación más sofisticada de metas y mecanismos de selección que hacen el procesado por omisión o lineal de conjuntos meta. Las metas *Secuencia* pueden ser anidadas a profundidades arbitrarias, dando paso a un marco rico de trabajo para especificar estrategias de control. Las metas *Si* soportan la activación condicional de una meta. Las metas *Ir* soportan un flujo no lineal de control dentro de un esquema de actividad, permitiendo al conjunto meta actual el poder ser transformado a cualquier otro conjunto meta etiquetado en el esquema. La iteración puede ser especificada a través de combinaciones de metas *Si* e *Ir*. Dos formas de paralelismo son provistas por medio de las modalidades de meta *División* y *And*. El paralelismo *División* engendra conjuntos de metas actuales independientes, con control en la intención padre que dirige al conjunto meta sucesor. Cada hilo de actividad para las metas engendradas continúa hasta que se completa, o termina la intención padre. En contraste, el paralelismo *And* trata a las metas paralelas como una unidad; el procesamiento de la intención padre se suspende hasta que cada uno de los hilos ocasionados por las submetas *And* terminen.

IV.4.5 Metodología y uso

El comportamiento dirigido a una meta se realiza a través de esquemas para satisfacer tareas individuales. El comportamiento reactivo, dirigido a eventos se produce por intenciones que utilizan metas *Espera* las cuales suspenden hasta que una condición o evento resulte.

Un lenguaje común para el diseño de esquemas de actividad es definir una intención sombrilla para un objetivo específico, el cual a su vez invoca dos intenciones: una intención de bajo nivel para alcanzar el objetivo, y una intencion "monitor" (combinando así actividades dirigidas a meta y evento). Los monitores utilizan metas *Espera* para detectar cambios en el mundo que podrían influenciar las acciones requeridas para alcanzar el objetivo completo del esquema de nivel más alto. Ciertos monitores pueden identificar condiciones de falla que invalidarían el alcance realizado por la tarea actual. Otros proveen una reacción a eventos inesperados que requieren atención inmediata. Los monitores pueden revisar efectos fortuitos que eliminan la necesidad de ciertas metas en intenciones activas, y modificar las estructuras de las intenciones de manera acorde.

Para ilustrar el empleo de varias modalidades meta y lenguajes, la Figura 30 presenta versiones simplificadas de esquemas de actividad utilizados por Flakey para realizar tareas de navegación básica. El esquema: planear-y-ejecutar codifica un procedimiento para generar y ejecutar robustamente un plan para navegar a un destino designado. El destino se especifica como un parámetro al esquema, y se representa como un artefacto en el EPL, así ligando nociones abstractas del lugar de las creencias del robot acerca de su ambiente.

El conjunto meta inicial en el esquema (con la etiqueta: planear) utiliza una meta *And* aplicada a tres submetas para realizar ciertas inicializaciones. La primer submeta *Ejecutar* invoca una función *decir* que realiza un mando de generación discurso. La

```
(def. intención :planear−y−ejecutar
   :parámetros (dest)
   :metas
   '(( :planear
      (AND
        (EJECUTAR (decir "Planeando ruta para ~a" dest))
        (EJECUTAR (establecer *falló−ejecución* nulo))
        (= planear (encontrar ruta *región actual* dest)))   )
     (SI (planear es nulo)  (IR  :finalizar))
     (INTENTAR :monitor planear ()  :bloqueado nulo)
     (INTENTAR :seguir ruta ((ruta . planear))
     :bloqueado t : nombre seguirlo)
     (SI *falló−ejecución* (IR :planear))
     (:finalizar
      (SI (planear es nulo)
          (EJECUTAR (decir "No hay rutas transitables")))  )  )  )

(def. intención :monitor−planear
   :parámetros ()
   :metas
   '((:monitor (ESPERAR  *falló−ejecución*))
    (:limpiar (NO INTENTAR ' seguirlo))  ))
```

Figura 30: .Esquemas de actividad para navegación dirigida.

segunda meta *Ejecutar* inicializa la variable de ambiente *falló-ejecución*, la cual es utilizada para codificar información acerca del estatus del plan de ejecución. Esta variable es un ejemplo de información de estado dentro de PRS-lite que provee coordinación entre intenciones. La submeta final invoca un función *encuentra-ruta* que produce un plan topológico para navegar desde la localización actual al destino. La navegación dentro de Saphira es al nivel de regiones (puertas, uniones, pasillos); el planeador de rutas produce una secuencia de tales regiones que deberían ser atravezadas para alcanzar el destino objetivo.

Después de realizar las inicializaciones necesarias, el esquema tiene la intención monitor de no bloqueo *monitor-planear*, seguido por una intención de bloqueo *seguir-ruta*. Esta última intención (no mostrada aquí) se cicla a través de una ruta calculada, lanzando varias intenciones de más bajo nivel como las requeridas para navegar entre regiones sucesivas en la ruta generada. Las regiones de más bajo nivel pueden encontrar dificultades, las cuales se señalan estableciendo la variable de ambiente *falló-ejecución*. La meta *Espera* en la intención *monitor-planear* detectaría tal evento, y entonces procesaría la meta. Satisfacer esta meta haría desactivar la intención *seguir-ruta*, con la intención monitor terminando. Si ninguna intención de más bajo nivel señala una falla, la intención de bloqueo *seguir-ruta* se completará eventualmente, permitiendo el procesamiento de la intención *planear-y-ejecutar* restante. La intención *monitor-planear* es terminada automáticamente cuando su intención padre *planear-y-ejecutar* termine.

La Figura 31 despliega una postal de las estructuras de las intenciones en un punto durante una corrida en la cual Flakey emplea los esquemas anteriores para ejecutar una tarea dada. Cada línea en el despliegue consiste de: una marca inicial, indicando cuando la intención está bloqueada ($*$) o no bloqueada ($\circ$), el nombre del esquema de la actividad (e.g., *Liberar-Objeto*), un identificador único para el caso particular del

esquema (e.g., una etiqueta tal como *Seguir-Eso* si fue especificado en la meta *Intento*, si no un nombre asignado tal como I3674), y también el próximo estado de ejecución (para una intención) o B (para un comportamiento). En el instante capturado por este despliegue, PRS-lite tiene dos intenciones activas en el más alto nivel: (correspondiente a dos distintos objetivos usuario-especificado) *Liberar-Objeto* y *Evadir*. La intención *Evadir* tiene sólo un hilo activo en este punto, llamado el comportamiento para evadir colisiones (*Evadir-Colisión*). Note que aunque en el pasado o el futuro, ésta intención puede desencadenar muchas otras actividades. De mayor interés es el estado de ejecución para la intención *Liberar-Objeto*. En su nivel más alto, esta intención padre tiene la intención hija sencilla *planear-ejecutar*, la cual a su vez está ejecutando el esquema *seguir-ruta* mientras simultáneamente monitorea fallas de ejecución (vía *monitor-planear*). Como parte del esquema seguimiento de ruta, el robot se está moviendo actualmente desde un corredor a una unión, el cual ha activado una intención para moverse hacia un objetivo específico. En al más bajo nivel, tres comportamientos están activos simultáneamente, llamados *Seguir*, *Orientar*, y *Proteger objetivo*.

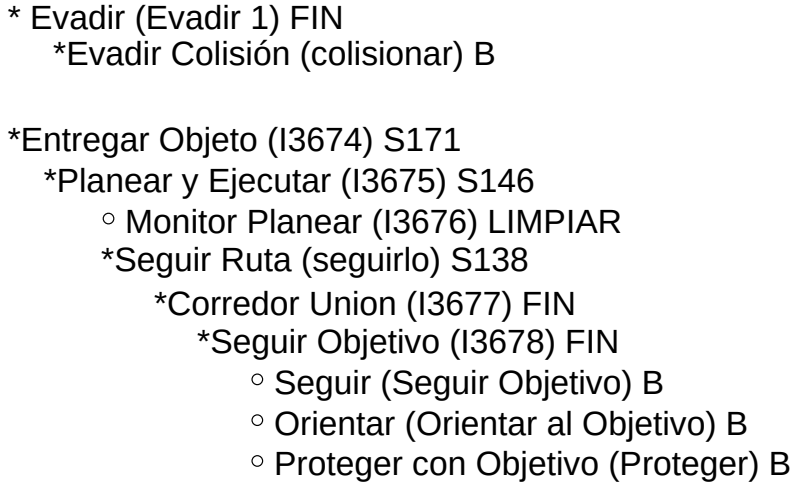


Figura 31: Imagen de las estructuras de intención durante la ejecución de una tarea dada.

IV.5 Comunicación

En este punto, nuestras ideas acerca de la comunicación han sido fuertemente influenciadas por la comunidad de Entendimiento del Lenguaje Natural, especialmente la teoría basada en planeación de actos de discurso. Eventualmente, sería bueno contar con un sistema de entendimiento completo que realice un reconocimiento de intenciones sobre las palabras del que habla, y formar planes apropiados. Para el escenario, nos concentramos en el área de identificación referente, igualando los términos del que habla a objetos en el ambiente inmediato del robot. En esta sección describimos el sistema de entrada de discurso, y los esquemas que se llevan a cabo, toma de aviso y mandos de tarea.

IV.5.1 Discurso de entrada

Un sistema de reconocimiento de discurso excelente es el desarrollado en SRI, llamado CORONA. CORONA tiene modelos de reconocimiento independientes del que habla que no necesita entrenamiento, aunque tendrá mejor exactitud con modelos individualmente sintonizados. CORONA acepta una gramática BNF de frases, y produce cadenas de palabras habladas. En una computadora Sparc 10-51, opera cerca de dos veces el tiempo real en la gramática simple que usamos, i.e., una sentencia de 5 segundos tomaría cerca de 10 segundos para su reconocimiento.

Unos de los más duros problemas en la comunicación de voz es hacerle saber al supervisor cuando Flakey ha escuchado una palabra, y lo que el estado de su entendimiento es. Emplearon varias técnicas:

Clave. Puede haber muchos extraños parloteos durante las interacciones. La gramática fue creada con una palabra clave inicial, de manera que sólo las

frases que empiezan con la palabra “Flakey” son reconocidas. Esto trabaja extremadamente bien; es natural dirigir mandos y sentencias al robot en esta forma. Una buena característica adicional sería utilizar una interpretación direccional de fuentes de sonido para obtener la atención de Flakey.

Estado del proceso. CORONA utiliza un “apuntador final”, un periodo de energía de discurso baja, para señalar el fin de una frase hablada. Desde este punto hay un retraso hasta que el discurso es procesado; el que habla puede ser confundido acerca de cuando su estrada fue recibida, y cuando fue reconocida. Implementamos una simple bandera de final de discurso; Flakey dice “um” cuando el punto final es alcanzado. Así el que habla puede decir que su entrada fue recibida y esta siendo procesada.

Resultados. Es importante dar una retroalimentación al que habla acerca de la interpretación de su entrada. Por ejemplo, si el que habla dice “vuelta a la izquierda”, y es interpretado como “ve hacia delante”, el que habla será mitificado por el comportamiento del robot. De modo que Flakey reconocería cada mando hablado, también parafraseandolo, o moviendo sus camaras en reconocimiento. Si la frase no fue reconocida, Flakey dirá “Qué?”.

IV.5.2 Gestos

También el reconocimiento de gestos forma parte del proceso de atención. La idea fué usar una mezcla de discurso y gestos para referencia, e.g., “Esta oficina (apuntando a la derecha o izquierda) es de Karen”. Se efectuó un manejo para extraer suficiente información del sistema estereo, para reconocer los brazos apuntando a la izquierda o a la derecha, pero no se tuvo suficiente tiempo para integrarlo correctamente con el discurso de entrada, incluso para referencia simple. Esto sería un buen proyecto para

trabajo futuro, con un más elaborado sistema de entendimiento de lenguaje natural.

IV.5.3 Movimiento directo

Estos son mandos directos para mover hacia delante o voltear, o mirar (mover las cámaras) en cierta dirección, e.g., "mira detras de ti". Mandos de movimiento directo son implementados como una secuencia simple de comportamientos con condiciones de terminación bien definidas y tiempos fuera cortos. Por ejemplo, si a Flakey se le dice que se mueva hacia delante mientras se orienta a la pared, se volteará hacia la pared y se moverá hacia delante cerca de un metro, ya que la evasión de obstáculos está activa todo el tiempo. Sería más inteligente, por supuesto, reconocer que no es posible moverse hacia delante, y detener este hecho.

IV.5.4 Atender y seguir

Estos mandos envuelven coordinación de sondeo y acción, mediados por artefactos en el EPL. Incluso esquemas de actividades muy simples, cuando están coordinados con el sondeo, pueden dar la apariencia de un robot bien entrenado con habilidades humanas.

El esquema de Atender encuentra objetos parecidos a personas y llevan a Flakey orientado a la persona y a una distancia de un metro. El mando "mirame" o "encuentrame" provoca este esquema. Flakey realiza un escaneo desde la posición actual de la cámara al extremo izquierdo y derecho de la cabeza pan/tilt, dado un campo de visión completo de 360 grados. El primer objeto persona detectado por el algoritmo estereo de seguimiento se coloca en el EPL, y un esquema de actividad posiciona al robot orientado al objeto y mantiene las cámaras centradas en la persona. El éxito del esquema Atender se indica cuando dice "Aquí estoy"; la falla, después de un escaneo completo, por un "No puedo encontrarlo!".

Capítulo V

Algoritmos genéticos

V.1 Introducción

En este capítulo, se introduce al tema de los algoritmos genéticos: qué son, de dónde vinieron, y cómo se comparan o diferencian de otros procedimientos de búsqueda. Se Ilustra cómo trabajan, y se iniciará por entender su poder a través del concepto de un esquema o plantilla de similaridad.

V.1.1 ¿Qué son los algoritmos genéticos?

Los algoritmos genéticos son algoritmos de búsqueda basados en mécanismos de selección natural y genética natural. Estos algoritmos combinan la sobrevivencia del más apto entre estructuras de cadena con un intercambio de información estructurada pero aleatoria, para formar un algoritmo de búsqueda con algo relacionado a la búsqueda humana. En cada generación, un nuevo conjunto de creaturas artificiales (cadenas) se crea usando bits y piezas de los más aptos. Una parte nueva ocasionalmente se prueba para provocar una buena medición basado en la supervivencia del más apto. Mientras que son heurísticos, los algoritmos genéticos no son simples caminatas aleatorias. Explotan eficientemente información histórica para especular sobre nuevos puntos de búsqueda con un desempeño mejorado esperado.

Los algoritmos genéticos fueron desarrollados por Holland , 1992, sus colegas y sus estudiantes en la Universidad de Michigan. Las metas de su investigación han sido dos: (1) abstraer y explicar rigurosamente el proceso adaptativo de sistemas naturales, y

(2) diseñar sistemas artificiales de software que mantengan los mecanismos importantes de los sistemas naturales. Este enfoque ha dirigido a descubrimientos importantes en ambos sistemas de ciencia natural y artificial.

El tema central de investigación sobre algoritmos genéticos ha sido la *robustez*, el balance entre eficiencia y eficacia para sobrevivir en muchos diferentes ambientes. Las implicaciones de robustez para sistemas artificiales son muchas. Si los sistemas artificiales pueden hacerse más robustos, el costo de rediseñar puede ser reducido o eliminado. Si más altos niveles de adaptación pueden ser alcanzados, los sistemas existentes pueden desempeñar sus funciones más tiempo y de mejor manera. Los diseñadores de sistemas artificiales —ambos software y hardware, los sistemas de ingeniería, sistemas de computación, o sistemas de negocios— pueden sólo maravillarse de la robustez, eficiencia, y la fexibilidad de los sistemas biológicos. Las características de auto-reparación, auto-guia, y repreducción son la regla en los sistemas biológicos, mientras que apenas existen en los más sofisticados sistemas artificiales.

Así, hemos llegado a una conclusión interesante: dónde el desempeño robusto es deseado (y ¿dónde no lo es?), la naturaleza lo hace mejor; los secretos de la adaptación y sobrevivencia son mejor aprendidos por el estudio cuidadoso del ejemplo biológico. Aún no aceptamos el método del algoritmo genético por apelación a este solo argumento de la belleza de la naturaleza. Los algoritmos genéticos están teoricamente y empiricamente probados que proveen una búsqueda robusta en espacios complejos. La primer monografía en el tópico es *Adaptación en sistemas naturales y artificiales* de Holland , 1992. Muchos artículos y disertaciones establecen la validéz de la técnica en función de la optimización y aplicaciones de control. Habiendo sido establecido como un enfoque valido para problemas que requieren búsqueda eficiente y efectiva, los algoritmos genéticos están encontrando ahora más amplia aplicación en círculos de

negocios, científicos, y de ingeniería. Las razones detrás del número creciente de aplicaciones es claro. Estos algoritmos son computacionalmente simples pero poderosos en su mejoramiento de la búsqueda. Además, no están fundamentalmente límitados por suposiciones restrictivas acerca del espacio de búsqueda (las suposiciones concernientes a la continuidad, existencia de derivadas, unimodalidad, y otras cosas).

V.1.2 Robustez de la optimización tradicional y los métodos de búsqueda

Es importante preguntar si los métodos de búsqueda convencionales conocen nuestros requerimientos de robustez. La literatura actual identifica tres tipos principales de métodos: basados en el cálculo, enumerativos, y aleatorios.

Los métodos basados en el cálculo han sido estudiados fuertemente. Se subdividen en dos clases principales: indirectos y directos. Los métodos indirectos buscan el extremo local resolviendo usualmente el conjunto no lineal de ecuaciones resultante de establecer el gradiente de la función objetivo igual a cero. Esta es la generalización multidimensional de la noción de cálculo elemental de puntos extremos, como se ilustra en la Figura 32. Dada una función suave, sin restricciones, encontrar un posible pico comienza por la búsqueda restringida de esos puntos con pendiente de cero en todas direcciones. Por otro lado, los métodos de búsqueda directa buscan el óptimo local brincando sobre la función y moviendose en una dirección relacionada al gradiente local. Esta es simplemente la noción de *subir la colina*: encontrar el óptimo local, subir la función en la dirección más empinada permisible. Mientras que ambos métodos basados en el cálculo han sido mejorados, extendidos, utilizados, y reutilizados, un simple razonamiento muestra su falta de robustez.

Primero, ambos métodos son locales en alcance; el óptimo que buscan es el mejor

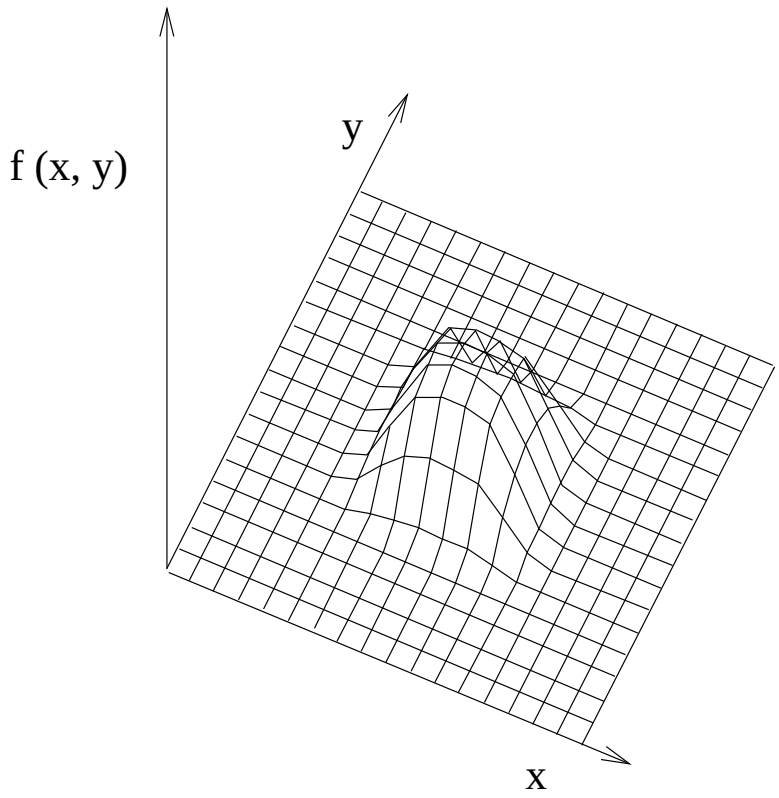


Figura 32: La función de un sólo pico es fácil para los métodos basados en el cálculo.

en el vecindario del punto actual. Por ejemplo, supongase que la Figura 32 muestra una porción del dominio completo de interés. Una imagen más completa se muestra en la Figura 33. Claramente, comenzar la búsqueda o los procedimientos de encontrar ceros en el vecindario del pico más bajo nos causará perder el evento principal (el pico más alto). Además, una vez que el pico más bajo ha sido alcanzado, se debe buscar mejorar aún más a través de comienzos aleatorios u otros trucos. Segundo, los métodos basados en el cálculo dependen de la existencia de derivadas (valores de pendiente bien definidos). Incluso si permitimos una aproximación numerica de derivadas, esto es una deficiencia severa. Muchos espacios de parámetros prácticos tienen una pequeña estima por la noción de una derivada y la suavidad que esta implica. Teóricos interesados en la optimización han sido demasiado serviciales para aceptar el legado de los grandes matemáticos del siglo XVIII y XIX quienes pintaron un mundo limpio de funciones objetivo cuadráticas, restricciones ideales, y derivadas siempre presentes. El mundo real de búsqueda está cargado con discontinuidades multimodalidad y un enorme espacio de búsqueda ruidoso. Estos fenómenos como están representados en una función de cálculo menos amigable en la Figura 34. No sorprende que los métodos dependientes de requerimientos restrictivos de continuidad y existencia de derivadas son inconvenientes para todos no sólo para dominios de problemas muy limitados. Por esta razón y por su inherente falta de alcance de búsqueda, debemos no aceptar los métodos basados en el cálculo. Estos métodos son insuficientemente robustos en dominios imprevistos.

Los esquemas enumerativos han sido considerados en muchas formas y tamaños. La idea es bastante directa; dentro de un espacio de búsqueda finito, o un espacio de búsqueda discretizado infinito, el algoritmo de búsqueda comienza buscando en valores de la función objetivo en cada punto en el espacio, uno a la vez. Aunque la simplicidad de este tipo de algoritmo es atractivo, y la enumeración es un tipo muy humano de búsqueda (cuando el número de probabilidades es pequeño), tales esquemas deben ser

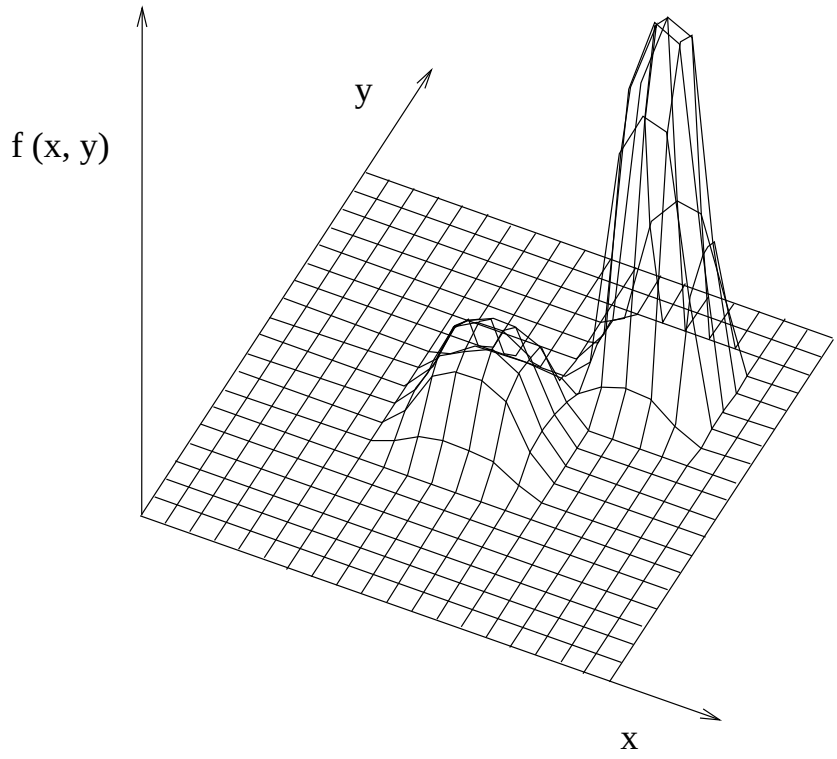


Figura 33: La función multi-pico causa un dilema. ¿Cuál colina deberíamos escalar?.

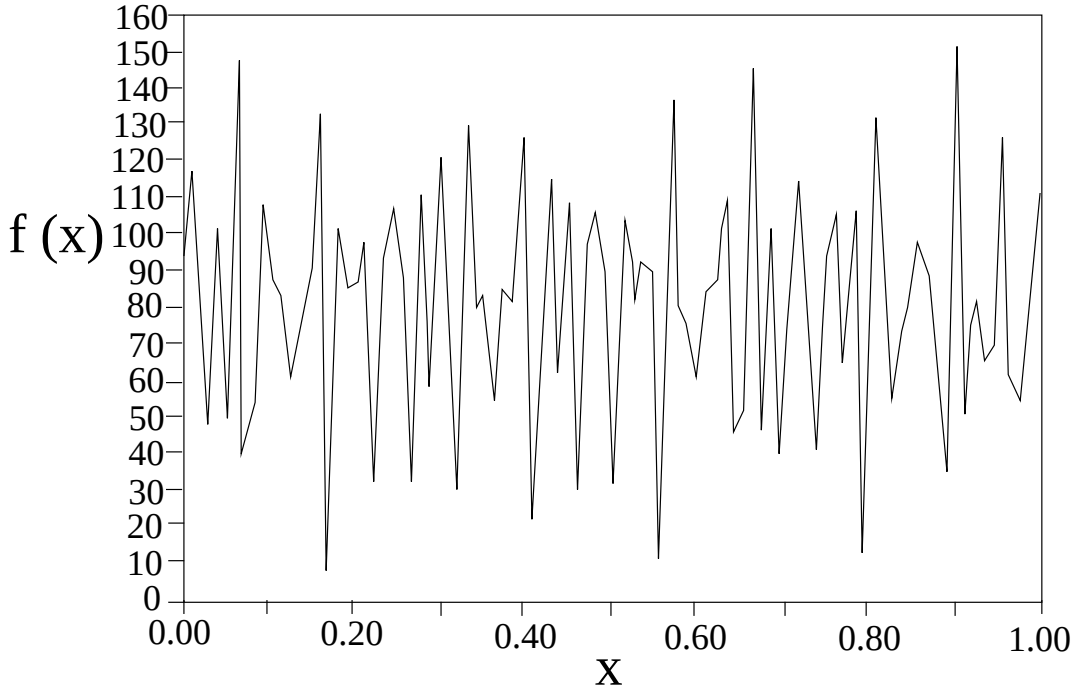


Figura 34: Muchas funciones son ruidosas y discontinuas y por lo tanto no convenientes para los métodos de búsqueda tradicionales.

descontados de la robustez por una simple razón: la falta de eficiencia. Muchos espacios prácticos son simplemente muy grandes para buscar uno a la vez y todavía tener una oportunidad de utilizar la información para algún fin práctico. Incluso el esquema de *programación dinámica* cae en los problemas de moderar el tamaño y la complejidad, sufriendo un problema llamado "la maldición de la dimensionalidad" por su creador Bellman , 1961. Debemos concluir que los esquemas enumerativos menos inteligentes son similarmente, ineficientes en problemas reales.

Los algoritmos de búsqueda aleatoria han alcanzado una creciente popularidad una vez que los investigadores han reconocido los defectos de los esquemas basados en el cálculo y los enumerativos. Todavía, las caminatas aleatorias y los esquemas aleatorios que buscan y guardan al mejor, deben también ser descontados por los requerimientos de eficiencia. De las búsquedas aleatorias, en corridas largas, se puede esperar que no sean mejores que los esquemas enumerativos. En nuestra prisa por descontar los métodos estrictamente aleatorios, debemos ser cuidadosos para separarlos de las técnicas aleatorizadas. Los algoritmos genéticos son un ejemplo de un procedimiento de búsqueda que emplea una elección aleatoria como una herramienta para guiar a una búsqueda altamente explotadora a través de una codificación del espacio de parámetros. Usando la elección aleatoria como una herramienta en un proceso de búsqueda dirigida parece extraño al principio, pero la naturaleza contiene muchos ejemplos. Otra técnica de búsqueda popular actual, *templado simulado*, utilizaa procesos aleatorios para ayudar a guiar su forma de búsqueda de estados de energía mínimos. La cosa importante a reconocer en esta coyuntura es que la búsqueda aleatoria no implica necesariamente búsqueda sin dirección.

Mientras nuestra discución no ha sido una examinación exaustiva de un gran número de métodos de optimización tradicional, terminamos con una conclusión algo inquietante: los métodos de búsqueda convencionales no son robustos. Esto no implica que

no son útiles. Los esquemas mencionados en la literatura tienen incontables combinaciones híbridas y permutaciones las cuales han sido utilizadas exitosamente en muchas aplicaciones. Sin embargo, cuando se estudian problemas más complejos, un gran número de métodos serán necesarios. Para poner este punto en perspectiva, inspeccionar el problema de espectro de la Figura 35. En la figura un indice de efectividad mítico se grafica a lo largo de un problema continuo para un esquema especializado, un esquema enumerativo, y un esquema robusto idealizado. La técnica de gradiente se desarrolla bien en su clase de problema bien definido, pero se vuelve altamente ineficiente en otro contexto. Por otro lado, el esquema enumerativo se desarrolla con la misma ineficiencia a través del espectro de problemas, como se ha mostrado por la curva de más bajo desempeño. Mucho más deseable sería una curva de desempeño como la etiquetada por el esquema robusto. Sería valioso sacrificar el desempeño pico sobre un problema en particular, para alcanzar un nivel relativamente alto de desempeño a través del espectro de problemas. (Por supuesto, con métodos amplios y eficientes podemos crear siempre esquemas híbridos que combinan lo mejor del método de búsqueda local con el esquema robusto más general.)

V.1.3 Las metas de la optimización

Antes de examinar los mecanismos y el poder de un algoritmo genético simple, debemos ser más claros acerca de nuestras metas cuando decimos que queremos optimizar una función o un proceso. ¿Qué estamos tratando de alcanzar cuando optimizamos? El enfoque convencional es presentado bien por Beightler *et al.* , 1979:

> El deseo del hombre por la perfección encuentra su camino en la teoría de la optimización. Estudia cómo describir y atender lo que es lo mejor, una vez que sabe como medir y alterar lo que es Bueno o Malo ... *La teoría*

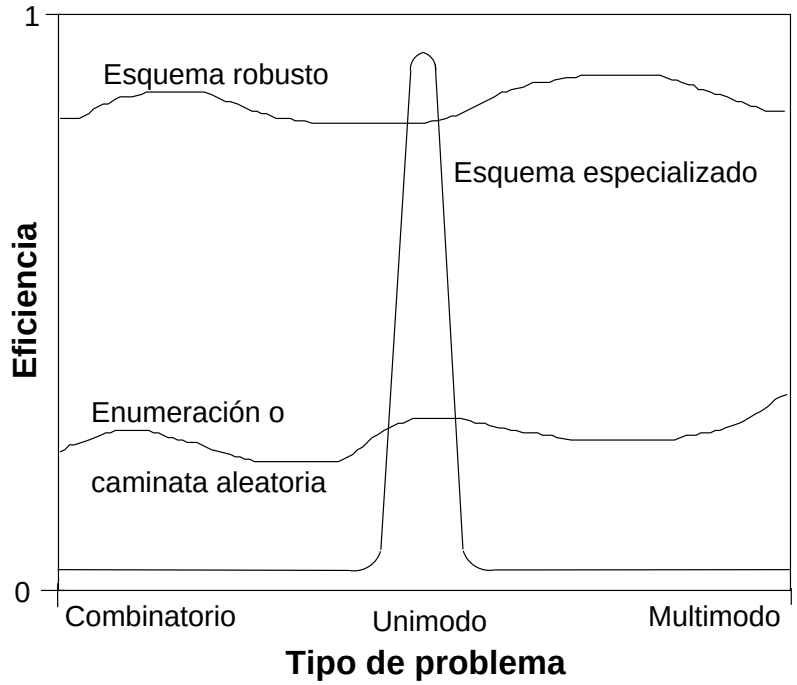


Figura 35: Muchos esquemas tradicionales trabajan bien en un dominio de problema angosto. Los esquemas enumerativos y caminatas aleatorias trabajan ineficientemente a lo largo de una amplio espectro. Un método robusto trabaja bien a lo largo de un amplio espectro de problemas.

> *de optimización* rodea el estudio cuantitativo de los óptimos y los métodos para encontrarlos.

Así la optimización busca mejorar el desempeño hacia algún punto o puntos óptimos. Note que esta definición tiene dos partes: (1) buscamos mejoramiento para alcanzar algo y (2) un punto óptimo. Hay una clara distinción entre el *proceso* de perfeccionamiento y el *destino* u óptimo. Aún, juzgando los precedimientos de optimización comunmente nos enfocamos solamente en la convergencia (¿puede el método alcanzar el óptimo?) y olvidamos enteramente el desempeño interno. Este énfasis proviene de los orígenes de la optimización en el cálculo. No es, sin embargo, un enfasis natural. Consideremos a un tomador de decisiones, por ejemplo, un hombre de negocios. ¿Cómo juzgamos sus decisiones? ¿Qué criterio usamos para decidir si él ha hecho un buen o mal trabajo? Usualmente decimos que el lo ha hecho bien cuando el hace selecciones adecuadas dentro del tiempo y recursos asignados. Su bondad es juzgada relativa a su competencia. ¿Produce él un mejor comentario? ¿Lo obtiene proporcionando más eficiencia? ¿Con mejor promoción? Nunca juzgamos a un hombre de negocios por un logro perfecto; sólo que sea mejor relativo a otro. Como un resultado, concluimos que la convergencia al mejor no es una cuestión en negocios o en la mayoría de las caminatas de la vida; sólo estamos preocupados por hacerlo mejor relativo a otros. Así, si queremos más herramientas de optimización como los humanos, debemos dejar reordenar las prioridades de optimización. La más importante meta de optimización es el mejoramiento. ¿Podemos obtener algún buen, nivel de "satisfacción" o desempeño rápidamente? El logro del óptimo es menos importante para sistemas complejos. Sería bueno ser perfecto: mientras tanto, podemos solamente esforzarnos para mejorar.

V.1.4 ¿En qué son diferentes los algoritmos genéticos de los métodos tradicionales?

Con el fin de superar los algoritmos genéticos a sus más tradicionales "primos" en la búsqueda por la robustez, los AG's deben diferir en algunas maneras muy fundamentales. Los algoritmos genéticos son diferentes de la más normal optimización y procedimientos de búsqueda en cuatro maneras:

1. Los AG's trabajan con una codificación del conjunto parámetro, no con los parámetros en sí mismos.

2. Los AG's buscan una población de puntos, no un sólo punto.

3. Los AG's utilizan información de recompensa (función objetivo), no derivadas u otro conocimiento auxiliar.

4. Los AG's utilizan reglas de transición probabilísticas, no determinísticas.

Los algoritmos genéticos requieren el conjunto parámetro natural del problema de optimización para ser codificado como una cadena de longitud finita sobre algún alfabeto finito. Como un ejemplo, considérese el problema de optimización puesto en la Figura 36. Deseamos maximizar la función $f(x) = x^2$ en el intervalo entero [0, 31]. Con los métodos más tradicionales estaríamos tentados a juguetear con el parámetro x, dandole vueltas como el sintonizador en un radio, hasta que alcancemos el más alto valor de la función objetivo. Con los GA's, el primer paso de nuestro proceso de optimización es codificar el parámetro x como una cadena de longitud finita. Hay muchas maneras de codificar el parámetro x. En este momento consideraremos un problema de optimización donde la codificación se vuelve un poco más natural.

Considérese el problema de la caja negra de apagadores en la Figura 37. Este problema consiste en un dispositivo de caja negra con cinco entradas o apagadores. Para

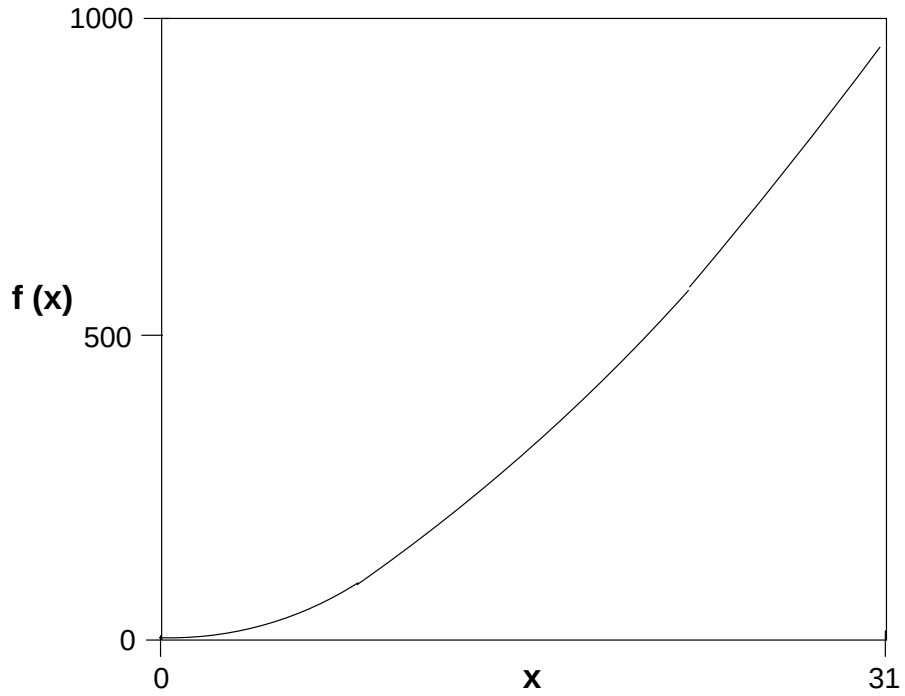


Figura 36: Ejemplo de optimización de una función simple, la función $f(x) = x^2$ sobre el intervalo entero [0, 31].

cada activación de los cinco apagadores, hay una señal de salida f, matemáticamente $f = f(s)$, donde s es una activación particular de los 5 apagadores. El objetivo del problema es activar los apagadores para obtener el máximo valor de f posible. Con otros métodos de optimización debemos trabajar directamente con el conjunto parámetro (las activaciones de los apagadores) y la alternación de los apagadores de una activación a otra usando las reglas de transición de nuestro método particular. Con los algoritmos genéticos, primero codificaremos los apagadores como una cadena de longitud finita. Una codificación simple puede ser generada considerando una cadena de cinco 1's y 0's donde cada uno de los cinco apagadores está representado por un 1 si el apagador está encendido y 0 si está apagado. Con esta codificación, la cadena 11110 codifica la activación donde los primeros cuatro apagadores están encendidos y el quinto apagador apagado. Algunas de las codificaciones no son tan obvias, pero en esta coyuntura sabemos que los algoritmos genéticos utilizan codificación. Más tarde

será aparente que los algoritmos genéticos explotan las similaridades en una forma muy general; como resultado, no están restringidos por las limitaciones de otros métodos (continuidad, existencia de la derivada, unimodalidad, etc.).

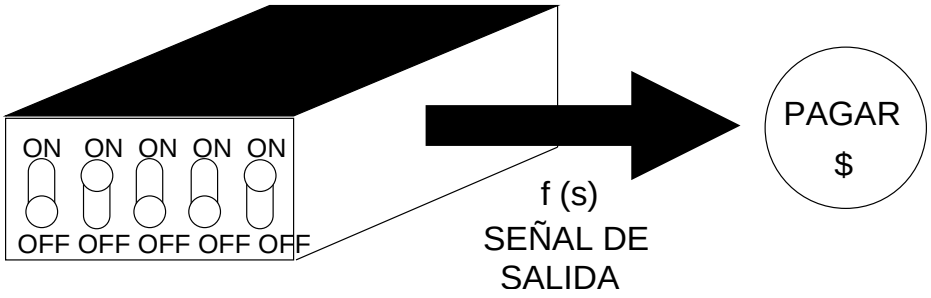


Figura 37: El problema de optimización de la caja negra con cinco apagadores ilustra la idea de una codificación y una medida de pago.

En muchos métodos de optimización, nos movemos con cautela de un solo punto en el espacio de decisión al siguiente, utilizando alguna regla de transición para determinar el siguiente punto. Este método punto a punto es peligroso por que es una receta perfecta para localizar picos falsos en espacios de búsqueda multimodales. En contraste, los AG's trabajan desde una rica base de datos de puntos simultáneamente (una población de cadenas), escalando muchos picos en paralelo; así, la probabilidad de encontrar un falso pico se reduce sobre otros métodos que van punto a punto. Como en un ejemplo, consideremos el problema de optimización de la caja negra (Figura 37) otra vez. Otras técnicas para resolver este problema deben comenzar con un conjunto de activaciones de apagadores, aplicar algunas reglas de transición, y generar un nuevo intento de activación de apagadores. Un algoritmo genético comienza con una población de cadenas y a partir de entonces genera sucesivamente poblaciones de cadenas. Por ejemplo, en el problema de los cinco apagadores, un comienzo aleatorio haciendo sucesivos lanzamientos de moneda (cara = 1, cruz = 0) debe generar la población inicial de tamaño $n = 4$ (pequeño para los algoritmos genéticos estandar):

01101

11000

01000

10011

Después de este comienzo, poblaciones sucesivas son generadas usando el algoritmo genético. Trabajando con una población de diversidad bien adaptada en lugar de un sólo punto, el algoritmo genético adhiere al viejo dicho de que hay seguridad en los números; pronto veremos cómo este gusto paralelo contribuye a la robustez de los algoritmos genéticos.

Muchas técnicas de búsqueda requieren mayor información auxiliar con el fin de trabajar adecuadamente. Por ejemplo, las técnicas de gradiente necesitan derivadas (calculadas analíticamente o numéricamente) con el fin de ser capaces de escalar el pico actual, y otros procedimietos de búsqueda local como las técnicas codiciosas de optimización combinatoria requieren acceso a la mayoría, si no a todos los parámetros tabulares. En contraste, los algoritmos genéticos no tienen necesidad de toda esta información auxiliar: los AG's están ciegos. Para realizar una búsqueda efectiva para encontrar mejores estructuras, solamente se requieren valores recompensa (valores de la función objetivo) asociados con cadenas individuales. Esta característica hace a un AG un método más canónico que muchos esquemas de búsqueda. Después de todo, cada problema de búqueda tiene una métrica (o métricas) relevante para la búsqueda. Sin embargo, diferentes problemas tienen formas diferentes y una cantidad enorme de información. Sólo si rechazamos usar esta información auxiliar podemos esperar desarrollar los esquemas basados en los términos generales que deseamos. Por otro lado, el rechazo a utilizar un conocimiento específico cuando existe puede colocar un límite superior en el desempeño de un algoritmo cuando va cabeza a cabeza con métodos diseñados para ese problema.

Distinto a muchos métodos, los AG's utilizan reglas de transición probabilísticas para guiar su búsqueda. Para personas familiarizadas con los métodos determinísticos esto parece extraño, pero el uso de la probabilidad no sugiere que el método es sólo una simple búsqueda aleatoria; ya que esta no es una decisión de cara o cruz. Los algoritmos genéticos utilizan elecciones aleatorias como una herramienta para guiar una búqueda hacia regiones del espacio de búsqueda con un mejoramiento probable. En efecto la selección juega el rol directriz en la búsqueda.

Tomadas juntas, estas cuatro diferencias —el uso directo de una codificación, búsqueda de una población, ceguera de información auxiliar, y operadores aleatorios— contribuyen a la robustez de los algoritmos genéticos y produce una ventaja resultante sobre otras técnicas más comúnmente empleadas.

V.1.5 Un algoritmo genético simple

Los mecanismos de un algoritmo genético simple son sorprendentemente simples, envolviendo nada más complejo que copiar cadenas e intercambiar cadenas parciales. La explicación de porqué este proceso simple trabaja es mucho más sutil y poderoso. La simplicidad de operación y poder de efecto son dos de las principales atracciones del enfoque del algoritmo genético.

Previamente se apuntó cómo los algoritmos genéticos procesan poblaciones y cadenas. Recordando el problema de los apagadores de la caja negra, recuerdese que la población inicial tenía cuatro cadenas:

01101

11000

01000

10011

También recuerdese que esta población fue elegida aleatoriamente en 20 lanzamientos sucesivos de una moneda imparcial. Ahora debemos definir un conjunto de operaciones simples que tomen esta población inicial y generar poblaciones sucesivas que (esperamos) mejoren con el tiempo.

Un algoritmo genético simple que permite buenos resultados en muchos problemas prácticos está compuesto de tres operadores:

1. Reproducción
2. Cruzamiento
3. Mutación

La reproducción es un proceso en el cual cadenas individuales son copiadas de acuerdo a sus valores de función objetivo, f (los biólogos llaman a esta función de aptitud). Intuitivamente, podemos pensar en la función f como alguna medida de beneficio, utilidad, o bondad que queremos maximizar. Copiando cadenas de acuerdo a sus valores de aptitud significa que las cadenas con más alto valor, tienen una mayor probabilidad de contribuir con uno o más hijos en la siguiente genéración. Este operador, por supuesto, es una versión artificial de la selección natural, una sobrevivencia Darwiniana del más apto entre creaturas cadena. En poblaciones naturales la aptitud se determina por la habilidad de las creaturas a sobrevivir a sus depredadores, la pestilencia, y los otros obstáculos a la edad adulta y subsecuente reproducción. En nuestro escenario artificial inperturbable, la función objetivo es el arbitro final de la vida o muerte de las creaturas cadena.

El operador de reproducción puede ser implementado de una forma algorítmica en un número de maneras. Quizás lo más fácil es crear una ruleta guiada donde cada cadena actual en la población tiene un espacio en la ruleta con el tamaño en proporción

a su aptitud. Supongase que el ejemplo de la población de las cuatro cadenas en el problema de la caja negra tiene valores de función objetivo o de aptitud f como se muestra en la Tabla IV (por ahora aceptamos estos valores como la salida de alguna caja negra desconocida y arbitraria —más tarde examinaremos una función y codificación que genera estos mismos valores).

Tabla IV: Valores de las cadenas y aptitud del problema muestra.

No.	Cadena	Aptitud	% del Total
1	01101	169	14.4
2	11000	576	49.2
3	01000	64	5.5
4	10011	361	30.9

Sumando la aptitud de las cuatro cadenas, se obtiene un total de 1170. El porcentaje de la aptitud total de la población es también mostrado en la Tabla IV. La correspondiente ruleta pesada para esta reproducción de la generación, se muestra en la Figura 38. Para reproducir, simplemente giramos la ruleta pesada así definimos cuatro cadenas. Para el problema del ejemplo, la cadena número 1 tiene un valor de aptitud de 169, el cual representa 14.4 porciento de la aptitud total. Como resultado, a la cadena 1 se da un 14.4 porciento de la ruleta guiada, y en cada giro se presenta la cadena 1 con probabilidad 0.144. Cada vez requerimos otro hijo, un simple giro de la ruleta produce el candidato a reproducción. De esta manera, cadenas más aptas tienen un número más grande de hijos en la generación que sigue. Una vez que las cadenas han sido seleccionadas para reproducción, se hace una replica exacta de la cadena. Esta cadena se introduce en un estanque de apareamiento, una nueva población tentativa, para más acción del operador genético.

Después de la reproducción, el cruzamiento simple (Figura 39) puede proceder en dos pasos. Primero, los miembros de las cadenas reproducidas nuevamente en el

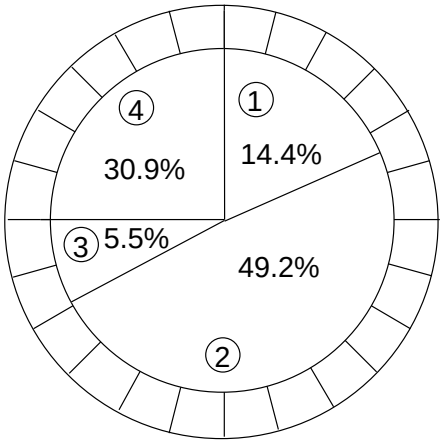


Figura 38: La reproducción simple asigna las cadenas hijas usando una ruleta con ranuras del tamaño de acuerdo a la aptitud.

estanque de apareamiento son apareadas aleatoriamente. Segundo, cada par de cadenas experimentan el cruzamiento como sigue: una posición entera k a lo largo de la cadena es seleccionada uniformemente de forma aleatoria entre 1 y la longitud de la cadena menos 1 $[1, l-1]$. Dos nuevas cadenas son creadas intercambiando todos los caracteres entre las posiciones $k+1$ y l inclusivamente. Por ejemplo, considere las cadenas A_1 y A_2 de nuestro ejemplo de la población inicial:

$$A_1 = 0\ 1\ 1\ 0 \mid 1$$
$$A_2 = 1\ 1\ 0\ 0 \mid 0$$

Supongase que en la elección de un número aleatorio entre 1 y 4, obtenemos una $k = 4$ (como está indicado por el simbolo separador $|$). El cruzamiento resultante produce dos nuevas cadenas donde la prima (') significa que las cadenas son parte de la nueva generación:

$$A'_1 = 0\ 1\ 1\ 0\ 0\ A'_2 = 1\ 1\ 0\ 0\ 1$$

Los mecanismos de reproducción y cruzamiento son sorprendentemente simples, a través de la generación de números aleatorios, copias de cadenas, y algunos intercambios

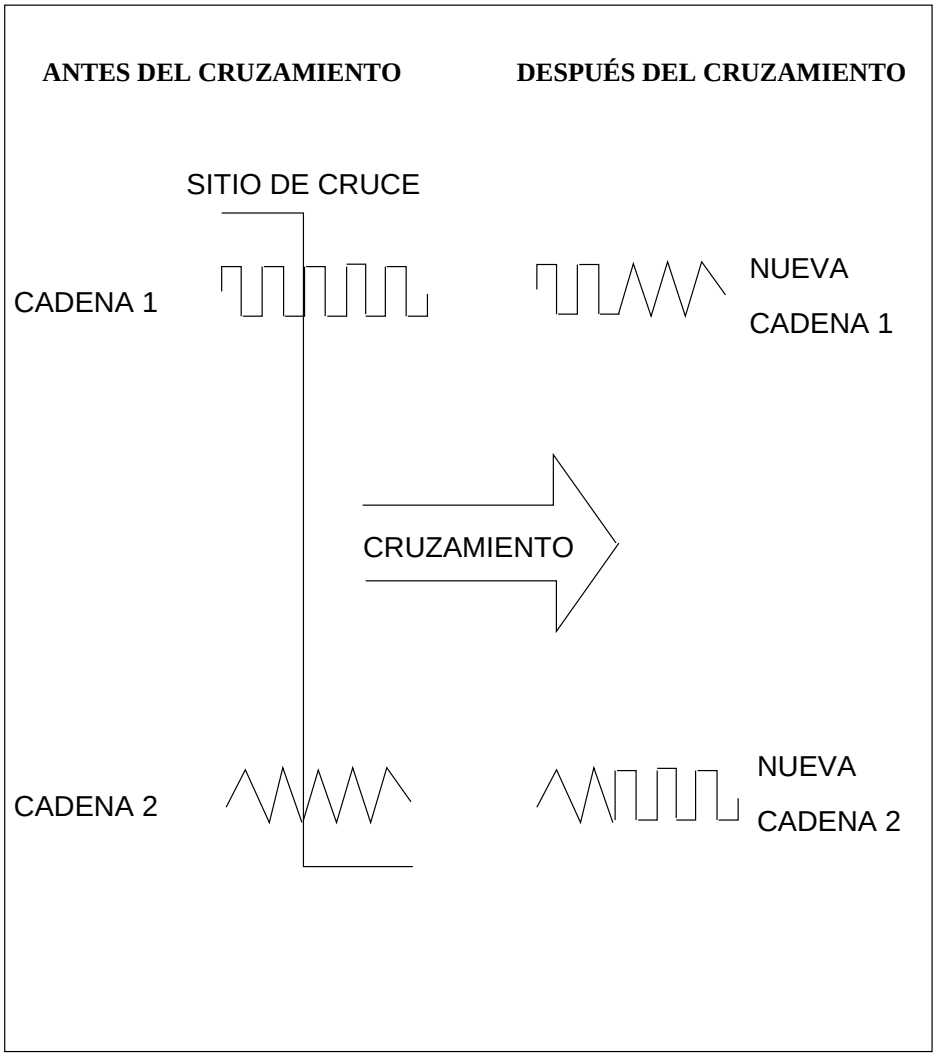


Figura 39: Un cruzamiento esquemático simple muestra la alineación de dos cadenas y el intercambio parcial de información, usando un sitio de cruce elegido aleatoriamente.

de cadena parcial. No obstante, el enfasis combinado de reproducción y el intercambio estructurado de información del cruzamiento, da a los algoritmos genéticos mucho de su poder. Al principio esto parece sorprendente. ¿Cómo pueden dos simples operadores resultar en algo útil? Además producen un rápido y robusto mecanismo de búsqueda. Por otro lado, ¿no parece un poco extraño que el azar debería jugar un rol fundamental en un proceso de búsqueda dirigido? Examinaremos la respuesta parcial a la primera de estas dos preguntas en un momento. La respuesta a la segunda pregunta fue bien reconocida por el matemático Hadamard , 1954:

> Veremos un poco más tarde que la posibilidad de imputar el descubrimiento al puro azar está ya excluido.... Por el contrario, hay una intervención del azar pero también un trabajo necesario de pérdida de conocimiento, lo último implica y no contradiciendo lo anterior.... Es más, es obvio que la invención o descubrimiento, en las matemáticas o en cualquier otro lado, toma lugar combinando las ideas.

Hadamard sugiere que incluso aunque el descubrimiento no es un resultado —no puede ser un resultado— de puro azar, es casi ciertamente guiado por la casualidad dirigida. Además. Hadamard insinua que un rol propio para el azar en un mecanismo de descubrimiento más parecido a los humanos es causa de la yuxtaposición de nociones diferentes. Es interesante que los algoritmos genéticos adopten la mezcla de Hadamard de dirección y azar en una manera que construye eficientemente nuevas soluciones de las mejores soluciones parciales de previos intentos.

Para ver esto, considerece una población de n cadenas (quizás la población de cuatro cadenas para el problema de la caja negra) sobre algún alfabeto apropiado, codificadas de manera que cada una es una *idea* completa o receta para realizar una tarea particular (en este caso, cada cadena es una idea de la activación de los apagadores completa).

Las subcadenas dentro de cada cadena (idea) contiene varias *nociones* de los que es importante o relevante para la tarea. Visto de esta manera, la población contiene no sólo una muestra de n ideas; además, contiene una multitud de nociones y rangos para la realización de la tarea. Los algoritmos genéticos explotan implacablemente esta riqueza de información (1) reproduciendo nociones de alta calidad de acuerdo a su desempeño y (2) cruzar estas nociones con muchas otras nociones de alto desempeño de otras cadenas. Así, la acción de cruzar con previas especulaciones de reproducción sobre las nuevas ideas construidas de bloques (nociones) de construcción de alto desempeño de intentos pasados. En el pasado, a pesar de la definición un tanto difuza de una noción, no se ha limitado una noción a combinaciones lineales simples de características sencillas o pares de características. Los biólogos han reconocido ampliamente que la evolución debe procesar eficientemente la epistasis (no linealidad de la posición) que se presenta en la naturaleza. En una manera similar, procesar nociones de algoritmos genéticos debe efectivamente procesar nociones incluso cuando dependen de sus características en maneras altamente no lineales y complejas.

Intercambiar las nociones para formar nuevas ideas es interesante, si pensamos en términos del proceso de *inovación*. ¿Qué es una idea inovadora? Como Hadamard sugiere, la mayoría usualmente es una yuxtaposición de cosas que han trabajado bien en el pasado. En el mismo sentido, la reproducción y el cruzamiento se combinan para buscar nuevas ideas potencialmente elocuentes. Esta experiencia de enfasis en el cruzamiento es análoga a la interacción humana que muchos de nosotros hemos observado en una muestra comercial o una conferencia científica. En una conferencia especializada, por ejemplo, varios expertos especialistas de alrededor del mundo se reunen para discutir lo último en tecnologia especializada. Después de las seciones de lectura, todos se emparejan alrededor del bar para intercambiar historias especializadas. Los bien conocidos expertos especialistas, por supuesto, están en mayor demanda e

intercambian más ideas, pensamientos, y nociones con sus más cercanos conocidos colegas especialistas. Cuando la muestra termina, la gente especialista regresa a su laboratorios especializados a probar un exceso de inovaciones especializadas. El proceso de reproducción y cruzamiento en un algoritmo genético es este tipo de intercambio. Las nociones con alto desempeño son respectivamente probadas e intercambiadas es la búsqueda de un mejor desempeño.

Si la reproducción de acuerdo a la aptitud combinada con el cruzamiento da a los algoritmos genéticos la mayoría de su poder de procesamiento, ¿Cuál es entonces el proposito del operador de mutación? No es sorprendente, hay mucha confusión acerca del rol de la mutación en genética (ambas natural y artificial). Quizás es el resultado de demasiadas películas, pero cualquiera que sea la causa de la confusión, encontramos que la mutación juega un rol decididamente secundario en la operación de los algoritmos genéticos. La mutación se necesita porque, incluso cuando pensamos que la reproducción y el cruzamiento efectivamente buscan y recombinan nociones existentes, ocacionalmente se pueden volver muy celosos y perder algún material genético potencialmente util (1's ó 0's en localizaciones particulares). En sistemas genéticos artificiales, el operador de mutación protege contra tal irrecuperable pérdida. En un simple AG, la mutación es la ocasional (con probabilidad baja) alteración aleatoria del valor de una posición en la cadena. En la codificación binaria del problema de la caja negra, esto significa simplemente cambiar un 1 a un 0 y vice versa. Por si sola, la mutación es una caminata aleatoria a través del espacio de la cadena. Cuando se utiliza en poca cantidad con la reproducción y el cruzamiento, es una política de seguro contra la pérdida prematura de nociones importantes.

El operador de mutación juega un rol secundario en el AG simple, simplemente notamos que la frecuencia de mutación para obtener buenos resultados en estudios empíricos de algoritmos genéticos está en el orden de una mutación por mil tranferencias

de bit (posición). Las tasas de mutación son similarmente pequeñas (o más pequeñas) en poblaciones naturales, llevandonos a concluir que la mutación es apropiadamente considerada como un mecanismo secundario de la adaptación de un algoritmo genético.

Otros operadores genéticos y planes reproductivos han sido abstraidos del estudio del ejemplo biológico. Sin embargo, los tres examinados en esta sección, reproducción, cruzamiento simple, y mutación, han sido probados como computacionalmente simples y efectivos en atacar un número importante de problemas de optimización.

V.1.6 Algoritmos genéticos trabajando —Una simulación manual

Apliquemos nuestro propio algoritmo genético a un problema de optimización particular paso a paso. Considerese el problema de maximizar la función $f(x) = x^2$, donde x se permite variar entre 0 y 31, una función desplegada anteriormente en la Figura 36. Para utilizar un algoritmo genético se debe primero codificar las variables de decisión de nuestro problema como alguna cadena de longitud finita. Para este problema, codificaremos la variable x simplemente como un entero sin signo binario de longitud 5. Antes de proceder con la simulación, se revisará brevemente la noción de un entero binario. Como criaturas con diez digitos, tenemos un pequeño problema manejando los enteros en base 10 y la aritmética. Por ejemplo, el número de 5 digitos 53,095 puede ser pensado como:

$5 \cdot 10^4 + 3 \cdot 10^3 + 0 \cdot 10^2 + 9 \cdot 10^1 + 5 \cdot 1 = 53,095.$

En base 2 aritmetica, tenemos solo dos digitos para trabajar 0 y 1, y como un ejemplo el número 10011 se decodifica a la base 10:

$1 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^0 = 16 + 2 + 1 = 19.$

Con un entero de cinco bits (digito binario) sin signo podemos obtener números

entre 0 (00000) y 31 (11111). Con una función objetivo bien definida y codificación, simulamos una sola generación de un algoritmo genético con reproducción, cruzamiento, y mutación.

Para comenzar, seleccionamos una población inicial aleatoriamente. Seleccionamos una población de tamaño 4 lanzando una moneda 20 veces. Podemos saltar este paso usando la población inicial creada en esta manera anteriormente para el problema de los apagadores de la caja negra. Viendo esta población, mostrada a la izquierda de la Tabla V, observamos que los valores de x decodificados están presentes con la aptitud o los valores de la función objetivo $f(x)$. Para estár seguros de cómo los valores de aptitud $f(x)$ son calculados de la representación de cadena, miremos la tercer cadena de la población inicial, cadena 01000. Decodificando esta cadena como un entero binario sin signo, notamos que hay un sólo 1 en la posición $2^3 = 8$. Por lo tanto para la cadena 01000 obtenemos $x = 8$. Para calcular la aptitud o la función objetivo simplemente elevamos al cuadrado el valor x y obtenemos el valor de aptitud resultante $f(x) = 64$. Otros valores de x y $f(x)$ pueden ser obtenidos similarmente.

Se puede notar que los valores de aptitud o función objetivo son los mismos que los valores de la caja negra. Esto no es coincidencia, y el problema de optimización de la caja negra estuvo bien representado por la función particular, $f(x)$, y la codificación que estamos utilizando. Por supuesto, el algortimo genético no necesita saber nada de esto; solamente optimizan alguna función arbitraria como alguna función polinomial con codificación binaria directa. Esta discusión simplemente refuerza una de las fortalezas del algoritmo genético: explotando similitudes en las codificaciones, los algoritmos genéticos pueden lidiar efectivamente con una más amplia clase de funciones comparado con la que pueden optimizar otros procedimientos.

Una generación de algoritmos genéticos comienza con la reproducción. Seleccionamos un estanque de apareamiento de la próxima generación girando la ruleta pesada

(mostrada en la Figura 38) cuatro veces. La simulación de este proceso usando una moneda ha resultado en la cadena 1 y la cadena 4 recibiendo una copia en el estanque de apareamiento, la cadena 2 recibiendo dos copias, y la cadena 3 no recibiendo copias, como se muestra en el centro de la Tabla V. Comparando esto con el número esperado de copias ($n \cdot \text{pselec}_i$) hemos obtenido lo que deberiamos esperar: el mejor obtiene más copias, el promedio permanece, y el peor muere.

Tabla V: Algoritmo genético manual.

No. cadena	Población Inicial (aleatoria)	Valor x (Entero sin signo)	$f(x)$ x^2	pselec_i $\frac{f_i}{\sum f}$	Cantidad esperada $\frac{f_i}{\bar{f}}$	Cantidad actual (ruleta)
1	0 1 1 0 1	13	169	0.14	0.58	1
2	1 1 0 0 0	24	576	0.49	1.97	2
3	0 1 0 0 0	8	64	0.06	0.22	0
4	1 0 0 1 1	19	361	0.31	1.23	1
Suma			1170	1.00	4.00	4.0
Promedio			293	0.25	1.00	1.0
Max			576	0.49	1.97	2.0

Tabla VI: Algoritmo genético manual (continuación)

Estanque de apareamiento (punto de cruce)	Pareja (aleatoria)	Sitio de Cruzamiento (aleatorio)	Nueva Población	Valor x	$f(x)$ x^2
0 1 1 0 | 1	2	4	0 1 1 0 0	12	144
1 1 0 0 | 0	1	4	1 1 0 0 1	25	625
1 1 | 0 0 0	4	2	1 1 0 1 1	27	729
1 0 | 0 1 1	3	2	1 0 0 0 0	16	256
					1754
					439
					729

Con un estanque activo de cadenas buscando pareja, el cruzamiento simple procede en dos pasos: (1) las cadenas son apareadas aleatoriamente, haciendo lanzamientos de

moneda para cruzar a las parejas seleccionadas, y (2) las parejas de cadenas conocidas se cruzan, usando lanzamientos de moneda para seleccionar los sitios de cruzamiento. Refiriendonos a la Tabla V, la elección aleatoria de parejas ha seleccionado la segunda cadena en el estanque de apareamiento para ser emparejada con la primera. Con un sitio de cruce de 4, las dos cadenas 01101 y 11000 se cruzan y dan paso a dos nuevas cadenas 01100 y 11001. Las dos cadenas restantes en el estanque se cruzan en el sitio 2; las cadenas resultantes pueden ser revisadas en la Tabla V.

El último operador, la mutación, se realiza sobre la base de bit por bit. Asumimos que la probabilidad de mutación en esta prueba es 0.001. Con 20 posiciones de bit transferidos deberiamos esperar $20 \cdot 0.001 = 0.02$ bits a ser mutados durante una generación. La simulación de este proceso indica que ningun bit fue mutado para este valor de probabilidad. Como un resultado, ninguna posición de bit se cambia de 0 a 1 o viceversa durante esta generación.

Siguiendo la reproducción, el cruzamiento, y la mutación, la nueva población está lista para ser probada. Para hacer esto, simplemente decodificamos las nuevas cadenas creadas por el algoritmo genético simple y se calculan los valores de la función de aptitud de los valores de x decodificados. Los resultados de una sola generación de la simulación son mostrados a la derecha de la Tabla V. Mientras se bosquejan conclusiones concretas de una sola prueba de un proceso estocástico es, a lo mejor, un negocio arriesgado. Comenzamos por ver cómo los algoritmos genéticos combinan nociones de alto desempeño para alcanzar el mejor desempeño. En la Tabla V, note como el desempeño máximo y promedio han mejorado en la nueva población. La aptitud promedio de la población ha mejorado de 293 a 439 en una generación. La aptitud máxima se ha incrementado de 576 a 729 durante ese mismo periodo. Aunque los procesos aleatorios ayudan a estas circunstancias, comenzamos por ver que este mejoramiento no es pura suerte. La mejor cadena de la primera generación (11000) recibe

dos copias por su altura, arriba del desempeño promedio. Cuando esto se combina aleatoriamente con la próxima cadena más alta (10011) y se cruzan en la localización 2 (otra vez aleatoriamente), una de las cadenas resultantes (11011) prueba ser una muy buena elección.

Este evento es una excelente ilustración de ideas y nociones análogas desarrolladas en la sección previa. En este caso, la buena idea resultante es la combinación de dos nociones arriba del promedio, a saber las subcadenas 1 1 - - - y - - - 1 1. Aunque el argumento es todavía algo heurístico, comenzamos por ver cómo los algoritmos genéticos efectuan una búsqueda robusta.

Se ha comparado el algoritmo genético con ciertos procesos de búsqueda humana comunmente llamados inovativos o creativos. Además, la simulación a mano del algoritmo genético simple nos ha dado alguna confianza de que algo interesantisimo está pasando aquí. Aún algo está perdido, ¿Qué está siendo procesado por los algoritmos genéticos y cómo sabemos si el procesamiento nos conducirá a resultados óptimos o cerca del óptimo en un problema particular? Claramente, como los científicos, ingenieros, y manejadores de negocios necesitamos entender el qué y el cómo del desempeño de los algoritmos genéticos.

V.2 Algoritmos genéticos: fundamentos matemáticos

La amplia brocha en la sección anterior pintó cruda, pintura de los algoritmos genéticos y sus mecanismos y poder. Quizás esta brocha traza una apelación a nuestro sentido de descubrimiento y búsqueda humano. Que de alguna manera un procedimiento regular aunque aleatorio puede alcanzar algo de la amplitud y don intuitivo de la búsqueda humana parece muy bueno para ser verdad. Más allá de la apelación intuitiva, es crucial que respaldemos esos sentimientos difusos y especulaciones acerca de los algoritmos

genéticos usando fríos, hechos matemáticos.

Actualmente, hemos empezado ya una definición más rigurosa de los AG's. Cuantitativamente, encontramos que hay enlistado un largo número de similitudes a explotar en una población de cadenas. Intuitivamente, vemos como los algoritmos genéticos explotan en paralelo las muchas similitudes contenidas en bloques de construcción de esquemas de corto, y alto desempeño. En esta sección, hacemos estas observaciones más rigurosas haciendo cuatro cosas. Primero, contamos con los esquemas representados dentro de una población de cadenas y considerando cual crece y cual decrece durante una generación dada. Para hacer esto, consideramos el efecto de reproducción, cruzamiento, y mutación sobre unos esquemas en particular. Este análisis nos lleva al teorema fundamental de los algoritmos genéticos que cuantifica estas tasas de crecimiento y decremento precisamente, apuntando además la forma matemática de este crecimiento. Esta forma está conectada a un problema importante y clásico de la teoría de decisión, el problema del bandido doblemente armado (y su extención, el bandido k-mente armado). La similitud matemática entre la solución óptima (perdida mínima) del bandido doblemente y k-mente armado y la ecuación describiendo el número de pruebas son notables. Contando el número de esquemas que es procesado de manera útil por el algoritmo genético simple, revela una tremenda influencia en el procesado del bloque de construcción. Finalmente, consideramos una pregunta importante: ¿Cómo sabemos que combinar los bloques de construcción nos lleva a un desempeño alto en problemas arbitrarios? La pregunta provoca nuestra consideración de algunas herramientas relativamente nuevas de análisis de algoritmos genéticos: transformar el esquema y el problema engañoso mínimo.

V.2.1 ¿Quién debería vivir y quién debería morir? Teorema fundamental

La operación de los algoritmos genéticos es remarcable sencillamente. Después de todo, comenzamos con un población aleatoria de n cadenas, copiar cadenas con alguna predisposición hacia el mejor, emparejar e intercambiar parcialmente subcadenas, y mutar un valor ocasional de bit para una buena medida. Incluso aunque los algoritmos genéticos manipulan directamente una población de cadenas en esta manera sencilla, en la sección anterior empezamos por reconocer que este procesamiento explicito de cadenas realmente causa el procesamiento implicito de muchos esquemas durante cada generación. Para análizar el crecimiento y decremento de muchos esquemas contenidos en una población, necesitamos alguna notación simple para agregar rigor a la discución. Consideramos la operación de reproducción, cruzamiento, y mutación sobre los esquemas contenidos en la población.

Consideramos cadenas, sin perder generalidad, para ser construidas sobre el alfabeto binario $V = 0, 1$. Como una conveniencia de notación, nos referimos a las cadenas con letras mayusculas y caracteres individuales con letras minusculas con subindices para su posición. Por ejemplo, la cadena de siete bits $A = 0111000$ puede ser representada simbólicamente como sigue:

$A = a_1a_2a_3a_4a_5a_6a_7$

Aquí cada uno de las a_i representan una sola característica binaria o detector (de acuerdo con la analogía natural, algunas veces llamamos a las a_i's *genes*), donde cada característica puede tomar un valor 1 o 0 (algunas veces llamamos a los valores de las a_i *alelos*). En la cadena particular 0111000, a_1 es 0, a_2 es 1, a_3 es 1, etc. Es posible también tener cadenas donde los detectores no están ordenados secuencialmente como en la cadena A. Por ejemplo una cadena A' podría tener el siguiente ordenamiento:

$A' = a_2 a_6 a_4 a_3 a_7 a_1 a_5$

Por ahora, asumimos que la función de las características está determinada por su posición.

La búsqueda genética significativa requiere una población de cadenas, y consideramos una población de cadenas individuales A_j, $j = 1, 2, ..., n$, contenidas en la población $\mathbf{A}(t)$ en el tiempo (o generación) t donde la negrita es utilizada para denotar la población.

Además de la notación para describir poblaciones, cadenas, posiciones de bit, y alelos, necesitamos una notación conveniente para describir los esquemas contenidos en cadenas y poblaciones individuales. Consideremos un esquema H tomado del alfabeto de tres letras $V+ = 0$, 1, *. Como se discutió en la sección pasada, el simbolo adicional, es asterisco o estrella, es un simbolo de no importa o carta amplia el cual se iguala a 0 o a 1 en una posición particular. Por ejemplo, considerece la longitud 7 del esquema $H = *11*0**$. Note que la cadena $A = 0111000$ discutida anteriormente es un ejemplo del esquema H, porque los alelos a_i de la cadena igualan las posiciones del esquema h_i en las posiciones establecidas 2, 3, y 5.

Hay 3^l esquemas definidos similarmente sobre una cadena binaria de longitud l. En general, para alfabetos de cardinalidad k, hay $(k+1)^l$ esquemas. Además, en una población de cadenas con n miembros hay a lo máximo $n \cdot 2^l$ esquemas contenidos en una población porque cada cadena es representativa de 2^l esquemas. Estos argumentos de conteo dan alguna sensación de la magnitud de información siendo procesada por algoritmos genéticos. Sin embargo, para realmente entender los importantes bloques de construcción de soluciones futuras, necesitamos distinguir entre tipos de esquemas.

Todos los esquemas no son creados igual. Algunos son más especificos que otros. Por ejemplo, el esquema 011*1** es una sentencia más definitiva acerca de la similitud importante que el esquema 0******. Además, cierto esquema abarca más de la longitud

de cadena total que otros. Por ejemplo, el esquema 1****1* abarca una porción más larga de la cadena que el esquema 1*1****. Para cuantificar estas ideas, introducimos dos propiedades de esquema: *orden* de esquema y *largo definido*.

El orden de un esquema H, denotado por $o(H)$, es simplemente el número de posiciones determinadas (en un alfabeto binario, el número de 1's y 0's) presentes en la plantilla. En los ejemplos anteriores, el orden del esquema 011*1** es 4 (simbolicamente, $o(011 * 1 * *) = 4$), mientras que el orden del esquema 0****** es 1.

La longitud definida de un esquema H, denotado por $\delta(H)$, es la distancia entre la primera y última posición de cadena. Por ejemplo, el esquema 011*1** tiene una longitud definida $\delta = 4$ porque la última posición especifica es 5 y la primer posición especifica es 1, y la distancia entre ellas es $\delta(H) = 5 - 1 = 4$. En el otro ejemplo (el esquema 0******), la longitud definida es particularmente fácil de calcular. Ya que hay solo una sola posición determinada, la primer y última posiciones específicas son la misma, y la longitud definida $\delta = 0$.

Los esquemas y sus propiedades son interesantes dispositivos notacionales para discutir y clasificar rigurosamente similaridades de cadena. Además de esto, proveen los medios básicos para analizar el efecto red de la reproducción y los operadores genéticos en bloques de construcción contenidos dentro de la población. Consideremos el efecto individual y combinado de la reproducción, cruzamiento, y mutación en esquemas contenidos dentro de una población de cadenas.

El efecto de la reproducción en el número esperado de esquemas en la población es particularmente fácil de determinar. Imaginemos que en un paso dado en el tiempo t hay m ejemplos de un esquema particular H contenido dentro de la población $\mathbf{A}(t)$ donde escribimos $m = m(H, t)$ (existen diferentes cantidades de diferentes esquemas H en diferentes tiempos t). Durante la reproducción, una cadena se copia de acuerdo a su aptitud, o más precisamente una cadena A_i es seleccionada con probabilidad

$p_i = f_i / \sum f_j$. Después de elegir una población no traslapada de tamaño n con remplazo de la población $\mathbf{A}(t)$, esperamos tener $m(H, t+1)$ representantes del esquema H en la población en el tiempo $t+1$ como está dado en la ecuación $m(H, t+1) = m(H, t) \cdot n \cdot f(H) / \sum f_j$, donde f_H es la aptitud promedio de las cadenas representando el esquema H en un tiempo t. Si reconocemos que la aptitud promedio de la población entera puede ser escrita como $\bar{f} = \sum f_j / n$ entonces podemos reescribir la ecuación de crecimiento de esquema reproductivo como sigue:

$$m(H, t+1) = m(H, t) \frac{f(H)}{\bar{f}}. \tag{53}$$

En palabras, un esquema particular crece como la razón de la aptitud promedio del esquema a la aptitud promedio de la población. Puesto de otra manera, los esquemas con valores de aptitud por arriba del promedio de la población recibirán un número mayor de muestras en la próxima generación, mientras los esquemas con valores de aptitud por debajo del promedio de la población recibirán un número menor de muestras. Es interesante observar que este comportamiento esperado se lleva a cabo con cada esquema H contenido en una población particular $\mathbf{A}$ en paralelo. En otras palabras, todos los esquemas en una población crecen o decrecen de acuerdo a sus esquemas promedios bajo la operación de reproducción sola. En un momento, examinamos porque esto debería ser una cosa buena que hacer. Por ahora, simplemente note que muchas cosas van en paralelo con operaciones simples en las n cadenas en la población.

El efecto de reproducción en el número de esquemas es cualitativamente claro; los esquemas de promedio superior crecen y los esquemas de promedio inferior mueren. ¿Podemos aprender algo más acerca de la forma matemática de este crecimiento (decrecimiento) desde la ecuación de diferencia de esquema? Suponga que asumimos un

esquema particular H que permanece sobre el promedio una cantidad $c\bar{f}$ con c una constante. Bajo esta suposición podemos reescribir la ecuación de diferencia de esquema como sigue:

$$m(H, t+1) = m(H, t)\frac{(\bar{f} + c\bar{f})}{\bar{f}} = (1 + c) \cdot m(H, t). \tag{54}$$

Comenzando en $t = 0$ y asumiendo un valor estacionario de c, obtenemos la ecuación

$$m(H, t) = m(H, 0) \cdot (1 + c)^t. \tag{55}$$

Los lectores orientados al negocio reconocerán a esta ecuación como la ecuación de interés compuesto, y los lectores orientados matemáticamente reconocerán una progreción geométrica o el análogo discreto de una forma exponencial. El efecto de la reproducción es ahora cuantitativamente claro; la reproducción asigna un número exponencialmente creciente (decreciente) de pruebas para los esquemas arriba (abajo) del promedio. Desde ahora investigaremos como el cruzamiento y la mutación afectan esta asignación de pruebas.

En alguna medida es extraño que la reproducción pueda asignar números exponencialmente crecientes y decrecientes de esquemas para futuras generaciones en paralelo; muchos, esquemas diferentes son muestreados en paralelo de acuerdo a la misma regla a través del uso de n operaciones simples de reproducción. Por otro lado, la reproducción sola no hace nada para promover la exploración de regiones nuevas del espacio de búsqueda, ya que no son buscados nuevos puntos; si sólo copiamos viejas estructuras sin cambio, entonces ¿como intentaremos algo nuevo? Aquí es donde los pasos de cruzamiento entran. El cruzamiento es un intercambio de información estructurada y aún aleatoria entre cadenas. El cruzamiento crea nuevas estructuras con un mínimo de desbaratamiento para la estrategia de asignación dictada por la reproducción sola.

Estos resultados en proporciones exponencialmente crecientes (o decrecientes) de esquemas en una población sobre muchos de los esquemas contenidos en la población.

Para ver cuales esquemas son afectados por el cruzamiento o cuales no lo son, considere una cadena particular de longitud $l = 7$ y dos esquemas representativos dentro de esa cadena:

$$\begin{aligned} A &= 0\,1\,1\,1\,0\,0\,0 \\ H_1 &= *\,1\,*\,*\,*\,*\,0 \\ H_2 &= *\,*\,*\,1\,0\,*\,* \end{aligned}$$

Claramente los dos esquemas H_1 y H_2 están representados en la cadena **A**, pero para ver el efecto del cruzamiento sobre los esquemas, primero recordemos que el cruzamiento simple procede con la selección aleatoria de una pareja, la selección aleatoria de un sitio de cruzamiento, y el intercambio de subcadenas desde el comienzo de la cadena al sitio de cruzamiento inclusive la subcadena correspondiente de la pareja escogida. Supongase que una cadena **A** ha sido escogida para emparejamiento y cruzamiento. En esta cadena de longitud 7, supongase que movemos una sola parte para elegir el sitio de cruzamiento (hay seis sitios en una cadena de longitud 7). Además supongase que el punto presenta un 3, significando que el corte de cruce tomará lugar entre las posiciones 3 y 4. El efecto de este cruce en nuestros esquemas H_1 y H_2 puede ser visto facilmente en el siguiente ejemplo, donde el sitio de cruzamiento ha sido marcado con el simbolo separador |:

$$\begin{aligned} A &= 0\,1\,1 \mid 1\,0\,0\,0 \\ H_1 &= *\,1\,* \mid *\,*\,*\,0 \\ H_2 &= *\,*\,* \mid 1\,0\,*\,* \end{aligned}$$

A menos que la pareja de la cadena **A** sea idéntica a **A** en las posiciones determinadas del esquema (una posibilidad que ignoramos prudentemente), el esquema H_1 será destruido porque el 1 en la posición 2 y el 0 en la posición 7 serán puestos en diferentes hijos (están en lados opuestos del simbolo separador marcando el punto de cruce, o punto de corte). Es igualmente claro que con el mismo punto de corte (entre

los bits 3 y 4), el esquema H_2 sobrevivirá porque el 1 en la posición 4 y el 0 en la posición 5 será llevado intacto a un solo hijo. Aunque se ha utilizado un punto de corte específico para la ilustración, es claro que el esquema H_1 es menos probable que sobreviva al cruzamiento que el esquema H_2 porque en promedio el punto de corte es más probable que caiga entre las posiciones extremas determinadas. Para cuantificar esta observación, notamos que el esquema H_1 tiene una longitud definida de 5. Si el sitio de cruzamiento se selecciona uniformemente de manera aleatoria entre los $l-1=7-1=6$ posibles sitios, entonces claramente el esquema H_1 es destruido con probabilidad $p_d=\delta(H_1)/(l-1)=5/6$ (sobrevive con probabilidad $p_s=1-p_d=1/6$). Similarmente, el esquema H_2 tiene una longitud definida $\delta(H_2)=1$, y es destruido durante un evento de seis donde el sitio de corte es seleccionado para ocurrir entre las posiciones 4 y 5 tal que $p_d=1/6$ o la probabilidad de sobrevivir es $p_s=1-p_d=5/6$.

Más generalmente, vemos que el límite inferior de la probabilidad de sobrevivir un cruzamiento p_s puede ser calculado para cada esquema. Porque un esquema sobrevive cuando el sitio de cruce cae fuera de la longitud definida. La probabilidad de sobrevivir bajo el cruzamiento simple es $p_s=1-\delta(H)/(l-1)$, ya que el esquema es probable de ser destruido cuando un sitio dentro de la longitud definida es seleccionado de los $l-1$ sitios posibles. Si el cruzamiento es realizado por una elección aleatoria, es decir con probabilidad P_c en un emparejamiento particular, la probabilidad de sobrevivencia puede ser dada por la expresión:

$$p_s \geq 1-p_c \cdot \frac{\delta(H)}{l-1}, \tag{56}$$

la cual se reduce a la expresión anterior cuando $P_c=1.0$.

El efecto combinado de reproducción y cruzamiento puede ser considerado. Como cuando consideramos la reproducción sola, estamos interesados en calcular el número

de un esquema particular H esperado en la próxima generación. Asumiendo la independencia de las operaciones de reproducción y el cruzamiento, obtenemos el estimado:

$$m(H, t+1) \geq m(H, t) \cdot \frac{f(H)}{\bar{f}} \left[1 - p_c \cdot \frac{\delta(H)}{l-1}\right]. \tag{57}$$

Comparando esto a la expresión anterior para la reproducción sola, el efecto combinado del cruzamiento y la reproducción se obtiene multiplicando el número esperado de esquemas para la reproducción sola por la probabilidad de sobrevivir bajo el cruzamiento p_s. Otra vez el efecto de las operaciones es claro. El esquema H crece o decrece dependiendo de un factor de multiplicación. Con ambos cruzamiento y reproducción, ese factor depende de dos cosas: cuando el esquema es por arriba o abajo del promedio de la población y cuando el esquema tiene longitud definida relativamente corta o larga. Claramente, esos esquemas con desempeño observado arriba del promedio y longitudes definidas cortas van a ser muestreados en tasas exponencialmente crecientes.

El último operador a considerar es la mutación. Usando nuestra definición previa, la mutación es la alteración aleatoria de una sola posición con probabilidad p_m. Con el fin para un esquema H de sobrevivir, todas la posiciones especificadas deben sobrevivir ellas mismas. Por lo tanto, ya que un solo alelo sobrevive con probabilidad $(1 - p_m)$, y ya que cada una de las mutaciones es estadísticamente independiente, un esquema particular sobrevive cuando cada uno de las o(H) posiciones determinadas dentro del esquema sobrevive. Multiplicando la probabilidad de sobrevivir $(1 - p_m)$ por si misma $o(H)$ veces, obtenemos la probabilidad de sobrevivir la mutación, $(1 - p_m)^{o(H)}$. Para valores pequeños de $P_m(p_m << 1)$, la probabilidad de sobrevivir del esquema puede ser aproximada por la expresión $1 - o(H) \cdot p_m$. Por lo tanto concluimos que un esquema particular H recibe un número esperado de copias en la próxima generación bajo reproducción, cruzamiento, y mutación como está dado por la siguiente ecuación

(ignorando terminos producto-cruce pequeños):

$$m(H, t+1) \geq m(H, t) \cdot \frac{f(H)}{\bar{f}} \left[1 - p_c \frac{\delta(H)}{l-1} - o(H) p_m\right]. \tag{58}$$

La adición de los cambios de mutación cambia nuestras concluciones previas un poco. Los esquemas cortos, de orden bajo, arriba del promedio reciben pruebas exponencialmente crecientes en generaciones subsecuentes. Esta conclusión es importante, tan importante que se le dio un nombre especial: el *Teorema de Esquema*, o teorema fundamental de los algoritmos genéticos. Aunque los cálculos que nos dirigen a probar el teorema de esquema no son demandantes, la implicaciones del teorema son de gran alcance y muy sutiles. Para ver esto, examinaremos el efecto de los tres operadores de los algoritmos genéticos sobre los esquemas en una población.

V.2.2 Procesado del esquema durante una prueba

En la sección anterior se mostraron los mecanismos de un AG simple a través de un cálculo manual de una sola generación. Regresaremos a ese ejemplo, esta vez observando cómo los AG procesan los esquemas —no cadenas individuales— dentro de la población. El cálculo a mano de la sección anterior se reproduce en la Tabla VI. En adición a la información presentada anteriormente, también mantenemos una cuenta de tres esquemas particulares, a los cuales llamamos H_1, H_2, y H_3, donde $H_1 = 1****$, $H_2 = *10**$, y $H_3 = 1***0$.

Observe el efecto de la reproducción, el cruzamiento, y la mutación sobre el primer esquema, H_1. Durante la fase de reproducción, las cadenas se copian probabilisticamente de acuerdo a sus valores de aptitud. Mirando la primer columna de la Tabla VI, notamos que las cadenas 2 y 4 son ambas representativas del esquema 1****. Después de la reproducción, notamos que tres copias del esquema han sido producidas (cadenas

2, 3, 4 en la columna de apareamiento). ¿Corresponde este número con el valor predicho por el teorema del esquema? Del teorema del esquema esperamos tener $m \cdot f(H)/\bar{f}$ copias. Calculando el esquema promedio $f(H_1)$, obtenemos $(576+361)/2 = 468.5$. Dividiendo esto por el promedio de la población $\bar{f} = 293$ y multiplicando por el número de esquemas H_1 en el tiempo t, $m(H_1, t) = 2$, obtenemos el número esperado de esquemas H_1 en el tiempo $t+1$, $m(H, t+1) = 2 \cdot 468.5/293 = 3.20$. Comparando esto al actual número de esquemas (tres), vemos que tenemos el número correcto de copias. Tomando este paso más adelante, nos damos cuenta que el cruzamiento no puede tener algun efecto más sobre el esquema porque una longitud definida $\delta(H_1) = 0$ evita la interrupción del bit solo. Además, con la taza de mutación establecida en $p_m = 0.001$ esperamos tener $m \cdot p_m = 3 \cdot 0.001 = 0.003$ o ningun bit cambiado dentro de las tres copias del esquema en las tres cadenas. Como un resultado, observamos que para el esquema H_1, obtenemos el número esperado de esquemas incrementando exponencialmente como se predice por el teorema del esquema.

Hasta el momento, todo parece estar muy bien; pero el esquema H_1 con su único bit determinado parece como algun caso especial. ¿Qué pasa con la propagación de importantes similitudes con más largas longitudes definidas? Por ejemplo considerese la propagación del esquema $H_2 = *10**$ y el esquema $H_3 = 1***0$. Siguiendo la reproducción y anterior al cruzamiento la replicación del esquema es correcta. El caso de H_2 comienza con dos ejemplos en la población inicial y termina con dos copias siguiendo la reproducción. Esto concuerda con el número esperado de copias, $m(H_2 = 2 \cdot 320/293) = 2.18$, donde 320 es el promedio del esquema y 293 es la aptitud promedio de la población. El caso de H_3 comienza con un sólo ejemplo (cadena 2) y termina con dos copias siguiendo la reproducción (cadenas 2 y 3 en la columna de copias de cadenas). Estos aciertos con el número esperado de copias $m(H_3) = 1 \cdot 576/293 = 1.97$, donde 576 es la aptitud promedio del esquema y 293 es la aptitud promedio de la población.

Tabla VII: Algoritmo genético manual.

No. cadena	Población Inicial (aleatoria)	Valor x (Entero sin signo)	$f(x)$ x^2	pselec$_i$ $\frac{f_i}{\sum f}$	Cantidad esperada $\frac{f_i}{\bar{f}}$	Cantidad actual (ruleta)
1	0 1 1 0 1	13	169	0.14	0.58	1
2	1 1 0 0 0	24	576	0.49	1.97	2
3	0 1 0 0 0	8	64	0.06	0.22	0
4	1 0 0 1 1	19	361	0.31	1.23	1
Suma			1170	1.00	4.00	4.0
Promedio			293	0.25	1.00	1.0
Max			576	0.49	1.97	2.0

Procesado del esquema

		Antes de la reproducción	
		Cadenas Representativas	Aptitud promedio del esquema $f(H)$
H_1	1 * * * *	2,4	469
H_2	* 1 0 * *	2,3	320
H_3	1 * * * 0	2	2

Tabla VIII: Algoritmo genético manual (continuación)

Estanque de apareamiento (punto de cruce)	Pareja (aleatoria)	Sitio de Cruzamiento (aleatorio)	Nueva Población	Valor x	$f(x)$ x^2
0 1 1 0 | 1	2	4	0 1 1 0 0	12	144
1 1 0 0 | 0	1	4	1 1 0 0 1	25	625
1 1 | 0 0 0	4	2	1 1 0 1 1	27	729
1 0 | 0 1 1	3	2	1 0 0 0 0	16	256
					1754
					439
					729

Procesado del esquema

Después de la reproducción			Después de todas las operaciones		
Cuenta esperada	Cuenta actual	Cadenas representativas	Cuenta esperada	Cuenta actual	Cadenas representativas
3.20	3	2,3,4	3.20	3	2,3,4
2.18	2	2,3	1.64	2	2,3
1.97	2	2,3	0.0	1	4

Las circunstancias siguiendo el cruzamiento son diferentes. Note que para el esquema corto H_2, las dos copias se mantienen incluso aunque el cruzamiento ha ocurrido. Por que la longitud definida es corta, esperamos que el cruzamiento interrumpa el proceso sólo una vez de cuatro ($l - 1 = 5 - 1 = 4$). Como resultado, el esquema H_2 sobrevive con alta probabilidad. El número actual esperado de los esquemas H_2 es así $m(H_2, t + 1) = 2.18 \cdot 0.75 = 164$, y esto se compara bién con la cuenta actual de dos esquemas. H_3 es un esquema de un color diferente. Porque debido al largo de su longitud definida ($\delta(H_3) = 4$), el cruzamiento usualmente destruye este esquema.

El cálculo manual ha confirmado la teoría desarrollada anteriormente en esta sección. A esquemas cortos, de bajo orden son dados números de muestras mayores o menores de forma exponencial dependiendo de las aptitudes promedio de los esquemas.

V.2.3 Algoritmos genéticos y evolución artificial

Si los algoritmos genéticos son una simulación plausible de la evolución biológica es un tópico muy discutido que no abordaremos aquí. Sin embargo, para el propósito de esta tesis se merece preguntar si la evolución artificial sería implementada como un algoritmo genético. La cuestion se vuelve relevante cuando uno mueve el foco de atención de un sistema de optimización a la auto-organización de sistemas autónomos. En el primer caso, la evolución artificial es vista como una técnica de búsqueda elegida por su eficiencia computacional y los algoritmos genéticos alcanzan bien este próposito porque son relativamente una técnica bien entendida y formalizada. En cambio, en el último caso lo que importa es el desarrollo espontáneo de entidades autónomas, tales como creaturas artificiales y robots. En este contexto, el cual podría ser definido como una forma de vida artificial, las nociones de eficiencia computacional, óptimo, e incluso progreso se vuelven menos definidas y secundarias. Aquí el aspecto más importante es la aparición de habilidades complejas desde un proceso de interacción autónoma entre

el agente y su ambiente.

La evolución artificial de sistemas autónomos difiere en muchas formas delicadas, pero importantes, de los algoritmos genéticos tradicionales. Se espera que los sistemas autónomos sobrevivan en ambientes desconocidos y parcialmente impredecibles ideando sus propias metas y encontrando soluciones a retos que pueden surgir. Por lo tanto, es difícil establecer a priori cuales habilidades y logros serán necesarios para obtener mayor aptitud. Entre más detallada y restringida es una función de aptitud, la evolución artificial se vuelve más cercana a una técnica de aprendizaje supervisado y menos espacio se deja para la aparición y autonomía del sistema evolutivo.

Como la evolución artificial se vuelve más manejable por la interacción entre el agente y su ambiente más que por una función de aptitud detallada, la noción de escalamiento de la colina en un espacio de búsqueda estático se vuelve menos útil. Si los cambios de ambiente y el agente no son seleccionados para una tarea predefinida detallada, el curso de la evolución es dictado por los cambios ambientales y adaptaciones locales. Un ejemplo extremo es el caso de la co-evolución competitiva donde agentes competidores son co-evolucionados bajo restricciones de aptitud holgadas. En estas circunstancias el panorama de la aptitud es dinámico y la noción de óptimo cambia en el tiempo.

Un aspecto relacionado es el de evolución incremental. Como hemos mencionado anteriormente, la evolución artificial de sistemas complejos puede fallar desde el principio porque todos los individuos de la genéración inicial reciben cero aptitud (el problema de autosuficiencia). Además, la definición de complejo y óptimo puede cambiar en el tiempo mientras el sistema evoluciona. Estos dos factores requieren una nueva formulación de la evolución artificial en la perspectiva de procesos incrementales. Por ejemplo, Harvey , 1992 ha propuesto SAGA —Algoritmo Genético de Adaptación de Especies—, una forma de evolución artificial caracterizada por genotipos de longitud

variable y una población bastante convergente genéticamente que se mueve en el espacio genético a través de mutaciones graduales, más que por el efecto del cruzamiento.

Variaciones adicionales a los algoritmos genéticos tradicionales son introducidas por inclusión de fenomenos tales como mutación y aprendizaje. Esto es cambios que afectan la aptitud de un individuo durante su propia vida. Los cambios ontogenéticos interactuan y afectan el curso de la evolución, incluso aunque las modificaciones inducidas no son directamente transmitidas por el cromosoma de un individuo. Una vez más, bajo estas circunstancias la relevancia de la asignación exponencial y la recombinación de esquemas con alta aptitud pueden ser mucho menores con respecto a otros factores que afectan la presión selectiva.

Capítulo VI

Evolución de navegación simple en el Pioneer P2-AT

VI.1 Introducción

En este capítulo se describirá la evolución de un comportamiento simple. Cualquier habilidad de navegación requiere el desarrollo de una transformación adecuada de información sensorial a acciones de motor. Las acciones realizadas por el robot dependen de la información sensorial obtenida, que a su vez, dependen de las acciones realizadas en los pasos previos. Este ciclo cerrado de retroalimentación hace difícil diseñar un sistema de control estable para simulaciones realistas en ambientes imprevistos. Desde una perspectiva ingenieril, se podría intentar enlistando todas las posibilidades sensoriales y asociarlas a un conjunto de acciones de motor predefinidas. En el caso de robots autónomos que se espera operen en ambientes parcialmente desconocidos e impredecibles, esta solución no siempre es viable. Como veremos más tarde, es muy difícil hacer a mano un sistema de control incluso para ambientes y tareas de navegación muy simples.

En la próxima sección describiremos con más detalles algunos de los problemas que surgen cuando se hace a mano un comportamiento de navegación simple y mostramos que la evolución artificial puede desarrollar automáticamente controladores para este propósito. Se volverá claro que la evolución artificial construye controladores inteligentes explotando las interacciones entre el robot y el ambiente que serían muy difícil de tomar en cuenta con métodos analíticos tradicionales.

VI.1.1 Movimiento con evasión de obstáculos

La navegación con evasión de obstáculos es una tarea clásica que necesita ser resuelta por la mayoría de la gente trabajando en robótica móvil y que requiere de experiencia. Un robot es puesto en un ambiente con algunos objetos en él y se requiere cubrir la distancia más larga sin golpear los objetos.

Si la morfología del robot (forma, motores y disposición de los sensores) es simétrica respecto a por lo menos un eje con dos ruedas a los costados, se puede resolver con un vehículo de Braitenberg, Braitenberg , 1984. Pero el Pioneer P2-AT no tiene dos ruedas, sino que tiene cuatro con un motor independiente cada rueda. Pero podemos utilizar el concepto del vehículo de Braitenberg para aplicarlo en el Pioneer P2-AT.

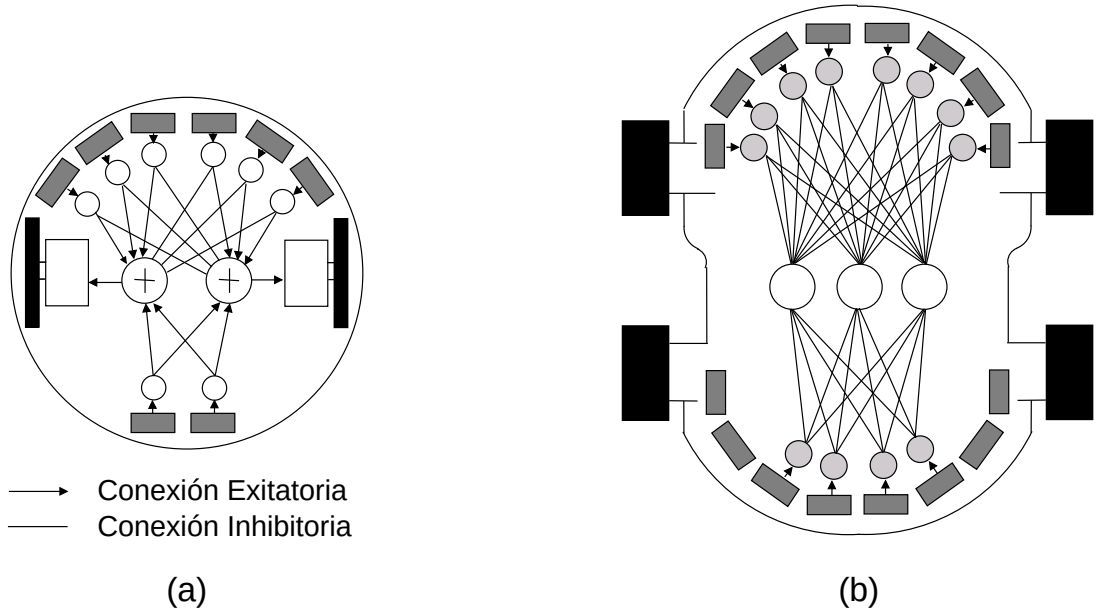


Figura 40: **(a)** Un vehículo de Braitenberg es un robot conceptual cuyas ruedas están directamente ligadas a los sensores a través de conexiones ponderadas. **(b)** Robot Pioneer P2-AT, tiene un motor independiente en cada rueda, se controla mediante instrucciones básicas.

Los vehículos de Braitenberg son robots conceptuales cuyas ruedas estan directamente ligadas a los sensores a través de conexiones ponderadas (con pesos) muy

similares a las de las redes neuronales artificiales. Si la conexión es positiva (exitatoria), la velocidad de rotación de la rueda es proporcional a la activación del sensor. Además, si la conexión es negativa (inhibitoria), la velocidad de rotación de la rueda es inversamente proporcional a la activación del sensor, posiblemente revirtiendo su dirección de rotación. La Figura 40 (a) muestra el diagrama de la instalación de un controlador tipo Braitenberg para el robot miniatura Khepera. Cada rueda está ligada a través de conexiones excitatorias a los sensores en su propio lado y a través de conexiones inhibitorias a los sensores en el lado opuesto. Un valor de compensación aplicado a cada rueda, genera movimiento hacia delante mientras la suma de pesos de señales entrantes dirigen al robot lejos de los objetos.

El robot móvil Pioneer P2-AT utiliza el sistema Saphira como sistema de control a partir del cual es posible enviar mandos básicos y directos de acción al robot en lenguaje Colbert para que este los ejecute; estos mandos son:

Movimiento Meta	**Continuas**
move(int mm)	speed(mm/sec)
turnto(int deg)	rotate(mm/sec)
turn(int deg)	stop

La diferencia principal entre los dos tipos de acciones, desde el punto de vista de Colbert, es si inducen a un movimiento con meta o no. Las acciones en la primer columna, las acciones *movimiento meta*, normalmente causan que el ejecutor Colbert pare hasta que su ejecución termine. Las acciones en la segunda columna, las acciones *Continuas*, regresan imediatamente y continuan la ejecución, aunque sus efectos pueden tardar una cantidad arbitraria de tiempo.

Otra distinción es si el mando causa un movimiento de translación o rotación. Los dos tipos de movimiento son independientes, de manera que un mando move() se puede activar simultáneamente con un mando rotate(). El mando stop que no toma

argumentos, detiene ambos movimientos, translacionales y rotacionales.

move(int mm) Mueve el robot hacia delante (mm positivos) o hacia atras (mm negativos) una distancia de |mm| milímetros.

turnto(int deg) Voltea al robot a apuntar en la dirección deg. Esta dirección está de acuerdo al sistema coordenado global del robot.

turn(int deg) Voltea al robot incrementalmente un ángulo |deg|. Esta dirección es en contra de las manecillas del reloj para grados (deg) positivos, y en favor para grados negativos.

speed(int mmsec) Establece una velocidad para el robot de |mmsec| (en mm/seg). La velocidad es hacia delante si mmsec es positivo, de otra manera es hacia atras. El robot continuará teniendo esta velocidad hasta que un mando de translación nuevo sea ejecutado.

rotate(int degsec) Establece una velocidad rotacional para el robot de |degsec| (en deg/seg). La velocidad es contra las manecillas del reloj si degsec es positivo, de otra manera es en favor de las manecillas. El robot continuará moviendose a esta velocidad hasta que otro mando rotacional sea ejecutado.

stop Detiene todos los movimientos del robot, cancelando cualquier mando de translación o rotación.

Utilizando sólo tres de estos mandos es posible diseñar un esquema de vehículo de Braitenberg para el robot Pioneer P2-AT, ya que este robot tiene cuatro ruedas y no dos, en la Figura 40 (b) se muestra un esquema del Pioneer. Los tres mandos que utilizamos para mover al robot son: speed(mmsec), move(mm), turn(deg), los argumentos de estos mandos dependen de las lecturas de los sonares en un momento

determinado, es decir, la velocidad, la distancia y los grados que el robot utilizará para moverse dependeran de las lecturas que recibe el robot de sus sonares. Estos argumentos se determinan de la misma manera que en un vehículo de Braitenberg conceptual, el cual determina la velocidad de sus dos ruedas tal y como el que se describió anteriormente, sólo que para el Pioneer P2-AT son tres los valores a cálcular.

Debería notarse que incluso este simple diseño requiere un análisis cuidadoso de perfiles de sensores y motores, y decisiones importantes respecto a la dirección de movimiento (hacia delante, hacia atras), su rectitud de trayectoria y su velocidad. Además, los pesos de conexión deben igualar las propiedades de respuesta del sensor. Por ejemplo, si un sensor tiene un perfil de respuesta más bajo comparado con otros sensores, sus conexiones de salida tendrían pesos más fuertes con el fin de obtener una buena trayectoria. El balance de pesos es igualmente importante. Por ejemplo, si las conexiones de salida del sensor más a la izquierda y de un sensor frontal tienen los mismos pesos, el robot tenderá a voltear demasiado cuando un objeto es percibido en el lado izquierdo o muy poco cuando un objeto es percibido justo en frente. Es claro que esto depende de que tan fuertes son los valores elegidos. Las fuerzas de las conexiones dependen también de las velocidades de compensación, las cuales dependen de la taza de actualización de los sensores, de las propiedades de reflexión de los objetos, y del ambiente desordenado. En otras palabras diferentes robots y diferentes ambientes requieren diferentes conjuntos de valores escogidos cuidadosamente.

Es válido ahora preguntar si un enfoque evolutivo podría encontrar una solución para el movimiento recto y evasión de obstáculos sin asumir todo el conocimiento apriori acerca de los sensores, motores y el ambiente. Si tal solución se encuentra por evolución artificial, es también válido preguntar cómo trabaja y cómo se compara con una arquitectura prediseñada tal como la del controlador tipo Braitenberg descrito

anteriormente. Nosotros hemos investigado estas preguntas con un robot Pioneer P2-AT.

Saphira cuenta con un software simulador del robot físico y su ambiente. Esta característica nos permite depurar las aplicaciones convenientemente en una computadora sin usar el robot físico. El simulador tiene modelos de error realistas para los sonares. Incluso su interface de comunicación es la misma que para un robot físico, de modo que no es necesario reprogramar o hacer cualquier cambio especial al cliente para tenerlo corriendo en el simulador o en el robot. El simulador también permite construir modelos en 2-D para ambientes reales o imaginarios, llamados mundos. Los modelos mundo son abstracciones del mundo real, con segmentos lineales representando las superficies verticales de corredores, puertas y objetos en él.

El ambiente en el que se moverá el robot es un espacio rectangular delimitado por 4 paredes definiendo una área de 3.52 x 2.48 mts. (Figura 41). La evolución se lleva a cabo completamente en el simulador, ya que la evolución puede durar días y hacerla en el robot real no sería muy práctico, el Pioner P2-AT cuenta con baterias con duración de hasta 4 horas, su tamaño es de 70 cms de diámetro, por esta y otras razones se decidió hacer la evolución en el simulador y probar después los resultados en el robot real.

VI.1.2 Función de aptitud y comportamiento

La meta fue evolucionar un sistema de control capaz de maximizar el movimiento hacia delante con la más alta velocidad posible mientras evita colisionar con las paredes. La definición de la función de aptitud Φ se basó en cuatro variables medibles desde la plataforma del robot. Esto es útil en robots que evolucionan sin control externo. Tres de estas variables corresponden a los argumentos de los tres mandos que mueven al Pioneer P2-AT.

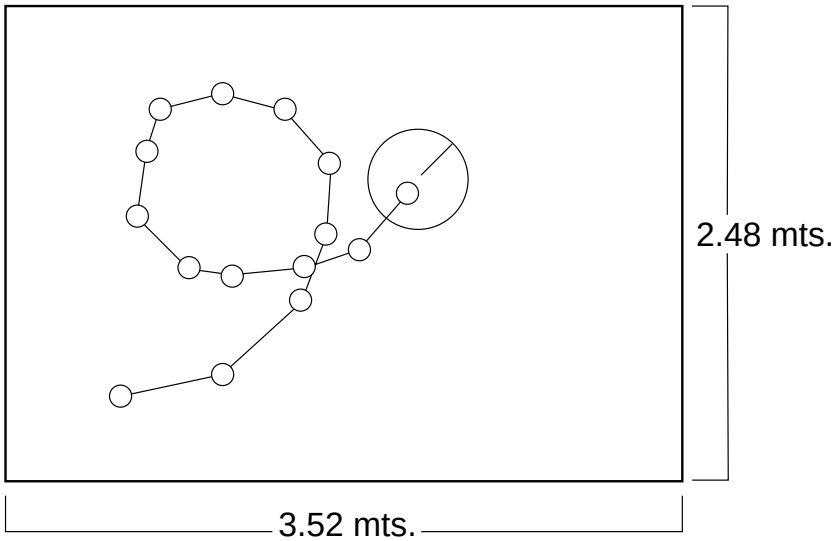


Figura 41: Vista del ambiente en el que se desenvolverá el robot Pioneer P2-AT.

La función de aptitud que utilizamos es:

$$\begin{aligned} \Phi = (1 - (\mid \theta \mid /90)) * (\nu/500) * (\delta/200) * \mu \\ -90 \leq \theta \leq 90 \\ 0 \leq \nu \leq 500 \\ 0 \leq \delta \leq 200 \\ 0 \leq \mu \leq 1 \end{aligned} \tag{59}$$

Donde θ es la cantidad de grados que el robot voltea, ν es la velocidad que el robot toma, δ es la distancia que el robot recorre, y μ es la lectura normalizada del sonar que obtiene la menor distancia a una de las paredes. Esta función se evalúa cuando un nuevo ciclo motor-sensorial se ejecuta y los valores resultantes son sumados y divididos por el número total de ciclos motor-sensoriales durante los cuales el individuo es probado. Estos cuatro componentes alimentan —respectivamente— el desplazamiento recto, la velocidad, la distancia, y la evasión de obstáculos.

El primer componente $(1 - (\mid \theta \mid /90))$ alienta al robot a desplazarse en línea recta. La variable θ es calculada restando al valor generado por el neuro-controlador, en el

rango de [0, 1], la cantidad de 0.5, generando un valor en el rango [-0.5, 0.5] y después convirtiendo ese valor a grados de la siguiente manera:

$$\theta = (o1 * 90)/0.5, \tag{60}$$

donde o1 es la salida del neuro-controlador correspondiente al mando de dirección. θ obtiene valores entre -90 y 90 grados, entre más grande el valor absoluto de θ, el término ($\mid \theta \mid /90$) se acerca más al 1; por lo tanto el término completo ($1 - (\mid \theta \mid /90)$) tiende a 0. Con objeto de maximizar este componente los grados absolutos de θ se deben minimizar. Esto quiere decir que un robot que está detenido totalmente o uno que va a velocidad máxima en línea recta tienen los mismos valores para este componente.

El segundo componente ($\nu/500$) alienta al robot a alcanzar una velocidad mayor de hasta 500 mm/seg. La variable ν es calculada convirtiendo el valor que se obtiene del neuro-controlador en el rango [0, 1] a una velocidad entre 0 y 500 mm/seg, todo esto con la siguiente fórmula:

$$\nu = o2 * 500, \tag{61}$$

donde o2 es la salida del neuro controlador correspondiente al mando de velocidad. Un robot girando sobre si mismo y un robot que va derecho a velocidad máxima generarán el mismo valor alto para este componente. El efecto combinado del primer y segundo componente dará mayor aptitud a los robots moviendose en línea recta a una velocidad máxima.

El tercer componente ($\delta/200$) alienta a su vez al robot a desplazarse una distancia de hasta 200 mm. La variable δ al igual que las anteriores se obtiene convirtiendo su valor de salida correspondiente del neuro-controlador empleando la siguiente conversión:

$$\delta = o3 * 200, \tag{62}$$

donde o3 es la salida del neuro-controlador correspondiente al mando de movimiento. Un robot que se mueva 200 mm por ciclo, a baja velocidad y girando obtendrá el mismo valor de aptitud que uno con alta velocidad y que se mueva en línea recta recorriendo los mismos 200 mm para este componente. Dado que los dos anteriores componentes combinados asignan una mejor aptitud a un robot moviendose en línea recta a una velocidad máxima, entonces combinando los tres componentes debe recorrer una distancia máxima para alcanzar una aptitud mejor.

El cuarto componente μ alienta la evasión de obstáculos. Cada uno de los sonares en el Pioneer P2-AT emiten pulsos ultrasónicos a través del emisor. Cuando chocan con un objeto los impulsos se reflejan y forman una señal de eco que es captada por el receptor. El receptor amplifica la energía de las ondas del eco y genera una señal que es enviada al indicador, constituido por una pantalla en la que se ve el objeto en el que han rebotado las ondas. Los sonares del Pioner P2-AT miden distancias mayores a los tres metros, aunque para este experimento se ajustaron para medir distancias máximas a los tres metros. Las medidas que arrojan los sonares son en milimetros y son normalizadas dividiendolas entre 3000 mm para que entren en el rango de [0, 1]. El valor μ es el valor del sonar con la más baja medida normalizada. Entre más grande el valor de μ quiere decir que el robot está más alejado de la pared. Combinado con los otros componentes, contribuirá a seleccionar robots que se muevan tan rectos como puedan pero evadiendo las paredes en su camino.

VI.1.3 Evolución de neuro-controladores

El sistema de control del robot es una red neuronal artificial con una arquitectura muy parecida al cableado de un vehículo de Braitenberg ilustrado en la Figura 40 (a). Esta consiste en una sola capa de pesos sinápticos de doce sonares a tres unidades de mando. Cada unidad de mando fue implementada como una neurona sigmoidal cuya salida, arreglada en los rangos [0, 1] y [-0.5, 0.5] en el caso de la unidad que corresponde a la dirección (θ) de rotación, que fue utilizada para establecer los argumentos de los mandos que mueven al robot (la dirección de rotación del robot es dada por el signo). La capa de salida tiene conexiones recurrentes a las entradas del neuro-controlador. Cada unidad de salida (unidad de mando) y de entrada tiene un peso sináptico adicional evolucionable desde una unidad de predisposición (bias), la Figura 42 nos muestra la arquitectura del neuro-controlador. La unidad de predisposición es muy importante en esta arquitectura porque provee a las unidades de salida con una activación incluso cuando no hay entrada sensorial.

El entrenamiento de pesos en redes neuronales artificiales (RN) es comunmente formulado como una minimización de una función error. Esta puede calcularse como el error cuadrático entre salidas objetivo y actuales promediadas sobre todos los ejemplos de forma iterativa ajustando los pesos de conexión. La mayoría de los algoritmos de entrenamiento, tales como los de retro propagación y las conjugaciones del gradiente están basados en el descenso de gradiente. Han existido algunas aplicaciones exitosas de retro propagación en varias áreas, pero la retro propagación tiene inconvenientes debido a su uso del descenso de gradiente. A menudo se ve atrapado en un mínimo local de la función de error y es incapaz de encontrar un mínimo global si la función de error es multimodal y/o no diferenciable.

Una manera de corregir los defectos de los algoritmos de entrenamiento basados en el descenso de gradiente es adoptar la evolución de redes neuronales artificiales,

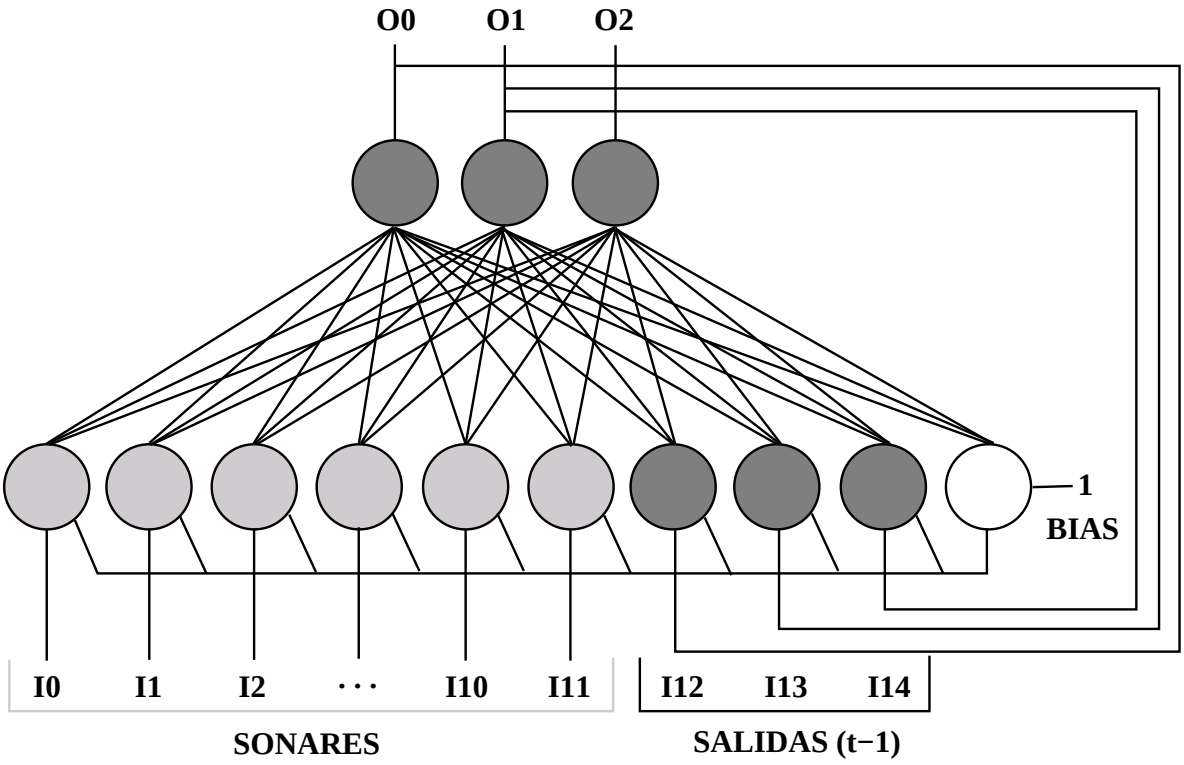


Figura 42: Arquitectura del neuro-controlador, las tres salidas corresponden a los parámetros de las instrucciones: turn(o0), speed(o1), y move(o2).

Yao , 1999, i.e., formular el proceso de entrenamiento con la evolución de pesos de conexión en el ambiente determinado por la arquitectura y la tarea a aprender. Los algoritmos evolutivos (AE) pueden entonces utilizarse efectivamente en la evolución para encontrar un conjunto de pesos de conexiones cercano al óptimo globalmente sin calcular información de gradiente. La aptitud de una RN puede ser definida de acuerdo a diferentes necesidades. Dos importantes factores los cuales a menudo aparecen en la función de aptitud (o error) son el error entre salidas objetivo y actuales y la complejidad de la RN. Diferente al caso de los algoritmos basados en el descenso de gradiente, la función de aptitud (o error) no tiene que ser diferenciable o incluso continua ya que los algoritmos evolutivos no dependen de la información del gradiente. Como se ha podido apreciar, los algoritmos evolutivos pueden tratar con espacios grandes, complejos, no diferenciables, y multimodales, los cuales son el caso típico en el mundo real. Un número considerable de investigaciones han sido dedicadas a la evolución de pesos de conexión.

El enfoque evolutivo para el entrenamiento de pesos en RN's consiste en dos fases principales. La primer fase es decidir la representación de los pesos de conexión, i.e., en la forma de cadenas binarias o no. La segunda fase es el proceso evolutivo simulado por un algoritmo evolutivo, en el cual los operadores de búsqueda tales como cruzamiento y mutación tienen que ser decididos en conjunto con el esquema de representación. Diferentes representaciones y operadores de búsqueda pueden dirigir los resultados a un desempeño de entrenamiento muy diferente.

Representación binaria

El algoritmo genético canónico utiliza cadenas binarias para codificar soluciones alternativas, a menudo calificadas de cromosomas. Algo del trabajo anterior en la evolución de pesos de conexión en RN's siguió este enfoque. En dicho esquema de representación,

cada peso de conexión se representa por un número de bits con cierta longitud. Una RN es codificada por la concatenación de todos los pesos de conexión de la red en el cromosoma.

Las ventajas de la representación binaria caen en su simplicidad y generalidad. Es directa la aplicación del cruzamiento clásico y mutación a cadenas binarias. Existe una pequeña necesidad de diseñar operadores de búsqueda complejos y adaptados.

A menudo se tiene que hacer un sacrificio entre la precisión de la representación y la longitud del cromosoma. Si demasiados bits son usados para representar cada peso o conexión el entrenamiento podría fallar porque algunas combinaciones de pesos de conexión con valores reales no pueden ser aproximadas con suficiente proximidad por sus valores discretos. Por otro lado, si muchos bits son usados, los cromosomas representando grandes RN's se volverán extremadamente largos y la evolución a su vez se volverá muy ineficiente.

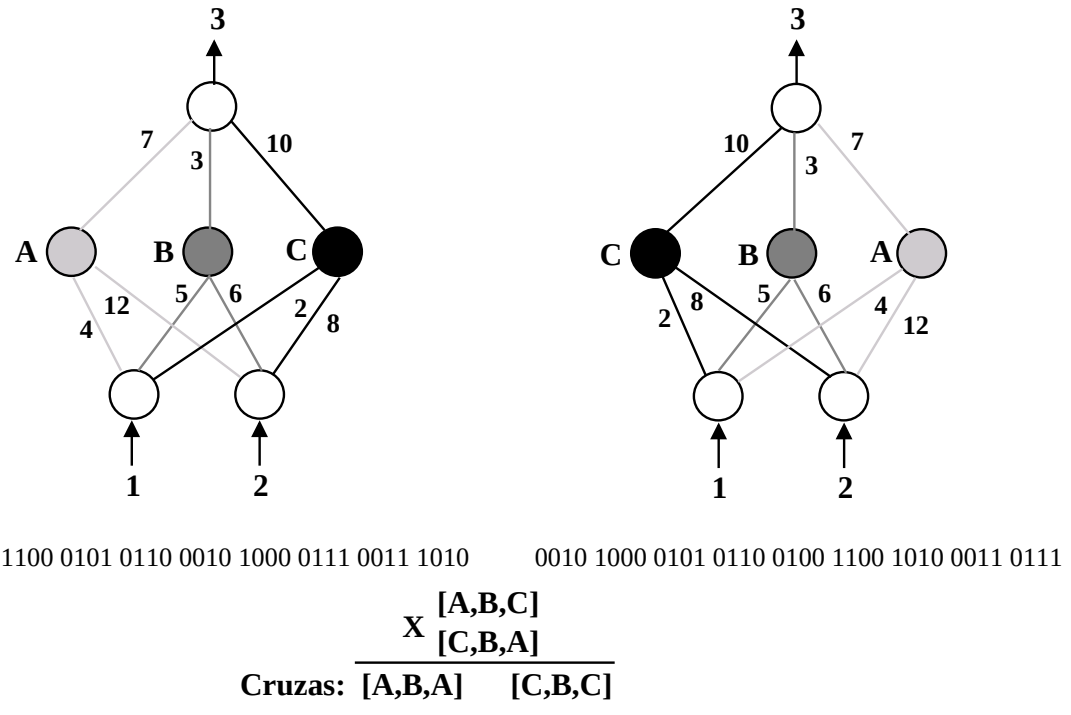


Figura 43: El *problema de permutación.*

Uno de los problemas que enfrenta el entrenamiento evolutivo de RN's es el *problema de permutación*, también conocido como el problema de la competencia de convención, Stanley y Miikkulainen , 2002. Este se produce por la tranformación muchos-a-uno desde la representación (genotipo) a la RN actual (fenotipo) ya que dos RN's que ordenan sus nodos ocultos de forma diferente a sus cromosomas podrían ser equivalentes en su funcionamiento. Por ejemplo, las RN's mostradas en la Figura 43 son equivalentes funcionalmente, pero tienen cromosomas diferentes. En general, cualquier permutación de los nodos ocultos producirá RN's funcionalmente equivalentes con diferentes representaciones de cromosomas. El problema de permutación hace al operador de cruzamiento muy ineficiente y poco efectivo en la producción de buenas cruzas.

La Figura 43 representa el problema para una red simple con tres unidades ocultas. Las dos RN's calculan la misma función incluso aunque sus unidades ocultas aparecen en diferente orden y son representadas por diferentes cromosomas, haciendolas incompatibles para el cruzamiento. La Figura 43 muestra que las dos recombinaciones de punto sencillo están perdiendo uno de los tres principales componentes de cada solución. Las redes representadas son sólo dos de las 6 posibles permutaciones del orden de las unidades ocultas. Las tres unidades ocultas A, B, y C, pueden representar la misma solución general en $3! = 6$ diferentes permutaciones. Cuando una de estas permutaciones se cruza con otra, es muy probable que información crítica se pierda. Por ejemplo, cruzando [A,B,C] y [C,B,A] puede resultar en [C,B,C], una representación que ha perdido una tercera parte de la información que ambos padres tenían. En general, para n unidades ocultas, hay $n!$ soluciones equivalentes funcionalmente.

Representación con números reales

Ha habido algunos debates sobre la cardinalidad del alfabeto del genotipo. Algunos han argumentado que la cardinalidad mínima, i.e., la representación binaria, no podría

ser la mejor. El análisis formal de representaciones no estandares y operadores basados en el concepto de clases equivalentes han producido fundamentos teóricos más solidos, Radcliffe , 1991. Los números reales han sido propuestos para representar los pesos de las conexiones directamente; i.e., un número real por peso de conexión.

Como los pesos de conexión son representados por números reales, cada individuo en una población evolucionando, será representado por un vector real. El cruzamiento y la mutación tradicionales no pueden ser utilizados directamente. Operadores de búsqueda especiales tienen que ser diseñados.

Una manera natural para evolucionar vectores reales sería usar programación evolutiva (PE) o estrategias de evolución (EE) ya que es particularmente idóneo para intentar la optimización continua. Distinto a los AG's, el operador de búsqueda primario en PE y EE es la mutación. Una de las mayores ventajas de utilizar algortimos evolutivos basados en la mutación es que pueden reducir el impacto negativo del problema de permutación. Por lo tanto el proceso evolutivo puede ser más eficiente. Han existido un número de ejemplos exitosos de la aplicación de PE o EE a la evolución de pesos de conexiones de RN.

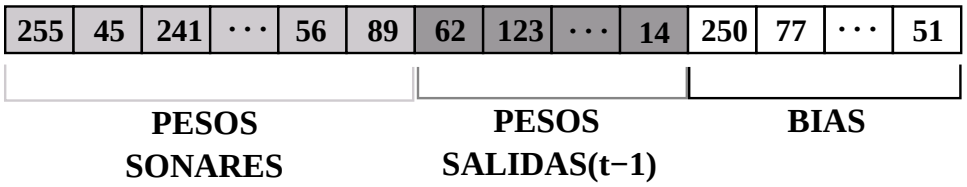


Figura 44: Genotipo del neuro-controlador. Es un vector donde cada elemento codifica un peso de una conexión en la red neuronal.

El genotipo o codificación de nuestros fenotipos de los individuos, que en este caso son neuro-controladores, se muestra de manera gráfica en la Figura 44 y de manera formal a continuación:

$I \in R^n$, $n = 63$

$I = [I_0, I_1, ..., I_{62}]$, $I_i \in R$ para toda i en el rango $0 \leq I_i \leq 255$

El genotipo del individuo es un vector I de dimensión 63, cada elemento I_i del vector pertenece a los reales. Los primeros 36 elementos, I_i para toda i en el rango $0 \leq i \leq 35$, codifican los pesos de las conexiones de las 12 entradas de los sonares y las 3 salidas del neuro-controlador. Los siguientes 9 elementos I_i para toda i en el rango $36 \leq i \leq 44$, codifican los pesos de las conexiones de las 3 entradas retroalimentadas a las 3 salidas del neuro-controlador. Los siguientes 18 últimos elementos I_i para toda i en el rango $45 \leq i \leq 62$, codifican los pesos de las conexiones de la unidad de predisposición a las 15 entradas y a las 3 salidas del neuro-controlador. Los valores de los pesos de las conexiones están en un rango de [0, 255].

VI.1.4 Algoritmo evolutivo

El algoritmo evolutivo se planeó para correr 40 generaciones, con una población de 40 individuos. Cada individuo en la población corresponde a un neuro-controlador cuyo desempeño es probado en el ambiente, cada individuo se prueba cinco veces (épocas). Antes de comenzar cada época el robot se mueve a una posición diferente a la que tiene cuando termina la época anterior. Cada época tiene una duración de 50 ciclos, un ciclo es el periodo de tiempo en el que el robot ejecuta los tres mandos: speed(), move(), y turn(). Al finalizar cada ciclo el algoritmo calcula el desempeño o aptitud Φ del robot durante ese ciclo. Estas aptitudes se acumulan durante toda la época. Al finalizar la época (los 50 ciclos) la aptitud total obtenida se divide por el número de ciclos que duró la época, que en este caso es de 50 ciclos. Si el robot colisiona con una pared durante una época, la prueba se suspende en ese ciclo, se le asigna una aptitud de cero a los ciclos restantes, y se continúa con la época siguiente. Una vez que los 40 individuos de la población han sido probados, y ya se cuenta con sus

aptitudes se escoge a los ocho individuos con mayor aptitud de la población. De estos ocho individuos se desprenderá una nueva población, en la que a cada uno de estos individuos se les aplicará una mutación con una taza de 3% para obtener cuatro hijos de cada individuo seleccionado, lo que nos da un total de 40 individuos nuevos. En la Figura 45 se muestra el pseudo-código del algoritmo evolutivo.

Para cada generación, que toma 2 horas aproximadamente, almacenamos la aptitud promedio de la población y la aptitud del mejor individuo en la población en el archivo statS1.fit en el directorio /usr/local/Saphira/bin/ (Figura 46). Un valor de aptitud de 1.0 podría ser alcanzado sólo por un robot moviendose recto a una velocidad máxima en un espacio abierto. En el ambiente mostrado en la Figura 41, donde algunos de los sensores casi siempre estuvieron activos (con valores menores a 1.0 ya normalizados) y donde muchas vueltas fueron necesarias para navegar, 0.031 fue el máximo valor obtenido por el controlador evolutivo.

Las posiciones y orientaciones iniciales que el robot toma al comienzo de cada época son parámetros importantes para determinar el desempeño del robot en cada época. Por esta razón cada robot tiene 5 oportunidades de probarse en el ambiente. Una vez que la evolución ha terminado, probamos a los tres últimos individuos evolucionados en la mismas condiciones iniciales, es decir, con los mismos parámetros de posición y orientación al inicio de cada época, de esta manera podemos hacer una comparación más objetiva, aunque sigue siendo una apreciación a distancia.

En la comparación los robots parten del punto en la esquina inferior izquierda del ambiente con una orientación hacia el este y comparamos a los tres últimos individuos evolucionados, las trayectorias que describieron estos robots se muestran en la Figura 47.

En la Tabla IX se muestran la distancia recorrida, y velocidad promedio que obtuvieron los 3 individuos durante su prueba que duró 300 ciclos de vida para todos los

```
Algoritmo básico EvoPioneer

Carga archivo de configuración
    Carga parámetros (tamaño de población, épocas, ciclos, padres, hijos, etc.)
Corre evolución
    Genera población aleatoriamente
    Desde  gen =  0  Hasta  Num_gen  Hacer
    {
        ind = 0;
        Desde  ind  Hasta  Num_ind  Hacer
        {
            Inicializa variables de época (aptitud acumulada del ind)
            Obtiene fenotipo
            Desde  epoca = 0 Hasta  Num_epoca  Hacer
            {
                ciclo = 0;
                Cambia de posición y orientación al robot
                Mientras  (ciclo < Num_ciclos) Hacer
                {
                    Toma lecturas de sensores
                    Mueve al robot (speed(), move(), turn())
                    Si choca
                        Termina los ciclos
                    Si no choca
                        Calcula rendimiento (aptitud) y lo acumula
                }
                Guarda la aptitud promediada del ind en un vector
            }
        }
        Extrae los i individuos más aptos del vector (Padres)
        Reproducción
            Desde  1  Hasta  Padres  Hacer
            {
                Desde 1 Hasta  Hijos  Hacer
                {
                    Mutar padre con una taza de mutación
                }
            }
    }
```

Figura 45: Pseudo-código del algoritmo evolutivo utilizado para evolucionar los neuro-controladores.

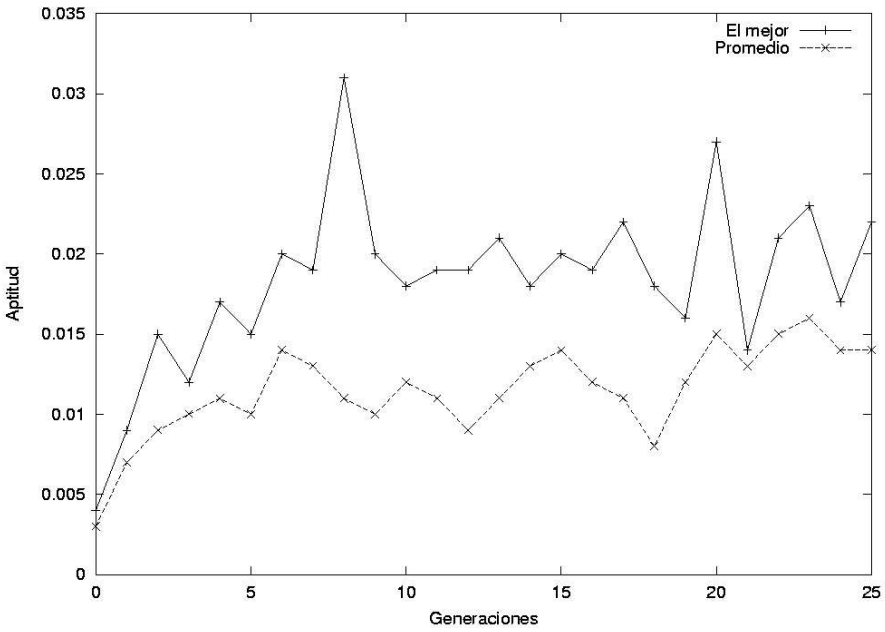


Figura 46: Aptitud promedio de la población y la aptitud del mejor individuo en la población graficada sobre cada generación.

individuos.

Tabla IX: Valores de los tres últimos individuos obtenidos durante la evaluación.

Individuo	Velocidad promedio	Distancia recorrida	Aptitud
23	131 mm/seg	54.508 mts	0.054621
24	143 mm/seg	52.561 mts	0.048724
25	136 mm/seg	56.066 mts	0.056085

En la Figura 47, las trayectorias de todos los individuos son parecidas. El individuo 24 tiene el valor más alto de velocidad promedio, pero es el que más se acerca a los muros, eso afecta su aptitud. El individuo 23 recorre más distancia que el 24 pero a una menor velocidad, y se puede ver que se mantiene más alejado de los muros que el individuo 24. El individuo 25 muestra la menor velocidad promedio pero la mayor distancia recorrida de los tres, además de ser el que más se aleja de los muros durante su recorrido.

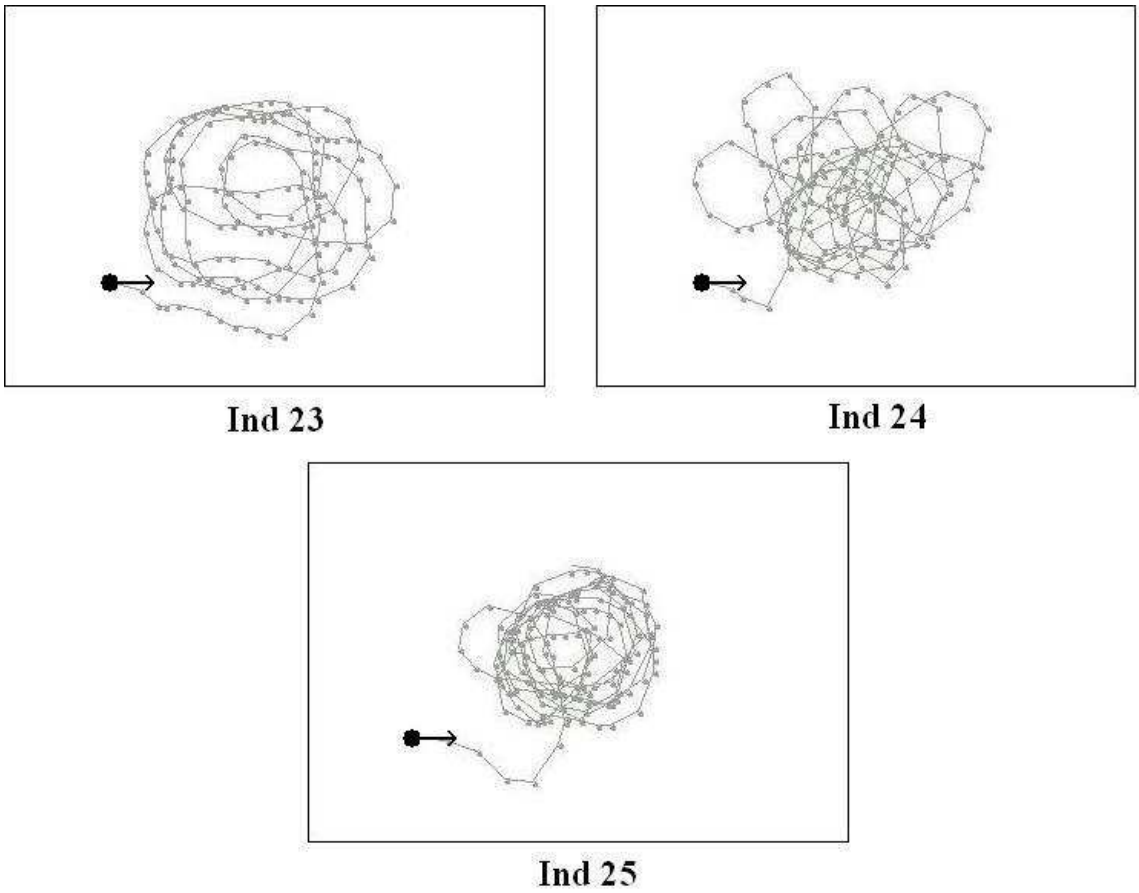


Figura 47: Trayectorias de los 3 últimos individuos evolucionados.

Los individuos tienen la tendencia a desplazarse en círculos, moviendose en el sentido contrario a las manecillas del reloj. En las siguientes comparaciones los robots son posicionados en 4 diferentes lugares dentro del ambiente y con orientaciones iniciales contrarias al sentido en el que los individuos se mueven, a estas situaciones les llamamos situaciones adversas. En las Figuras 48, 49, 50, y 51 se muestran las trayectorias que describen cada uno de los individuos en cuatro situaciones distintas.

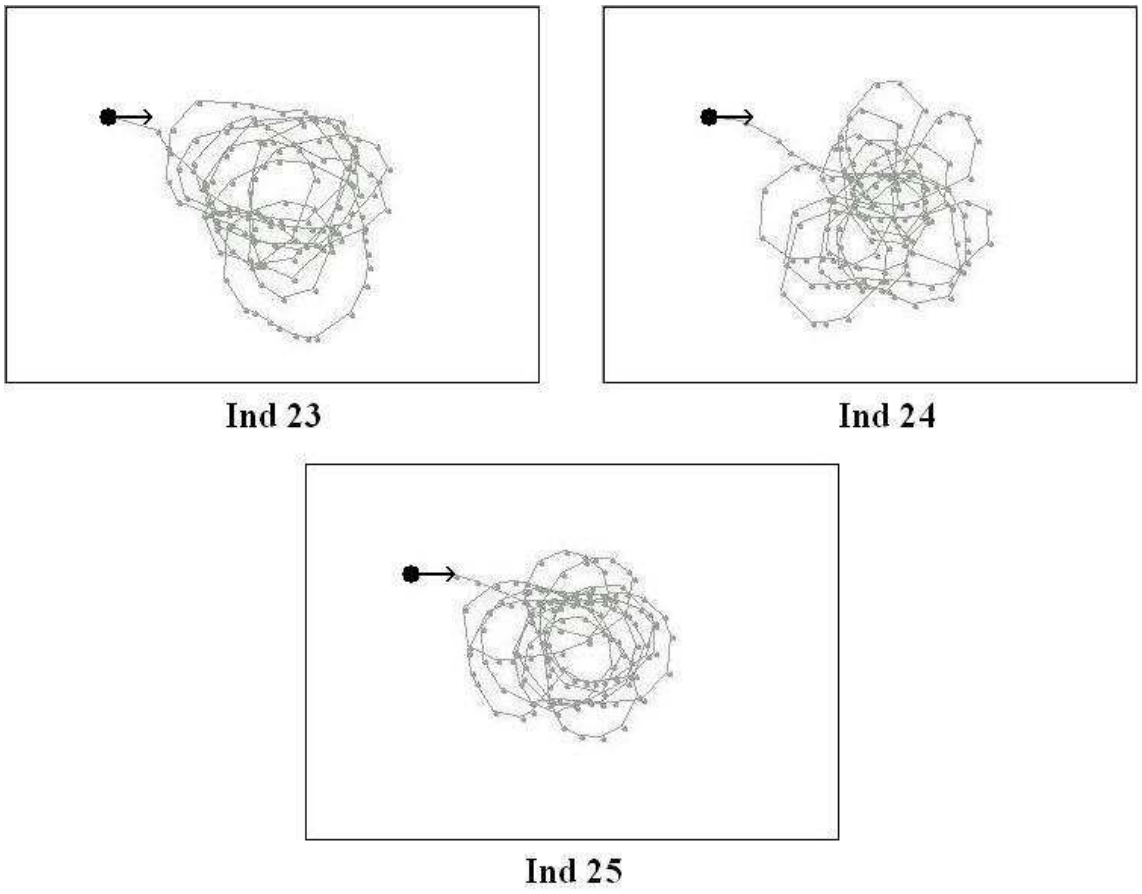


Figura 48: Trayectorias de los 3 últimos individuos evolucionados (Prueba 1).

En las Tablas X, XI, XII, XIII se muestran los valores de velocidad promedio, distancia recorrida y aptitud obtenida por los individuos durante las cuatro diferentes pruebas.

Se puede observar que el único individuo que terminó de forma satisfactoria todas

Tabla X: Valores de los tres últimos individuos obtenidos durante la evaluación (Prueba 1).

Individuo	Velocidad promedio	Distancia recorrida	Aptitud
23	135 mm/seg	54.399 mts	0.05627
24	138 mm/seg	52.652 mts	0.04979
25	133 mm/seg	56.392 mts	0.05830

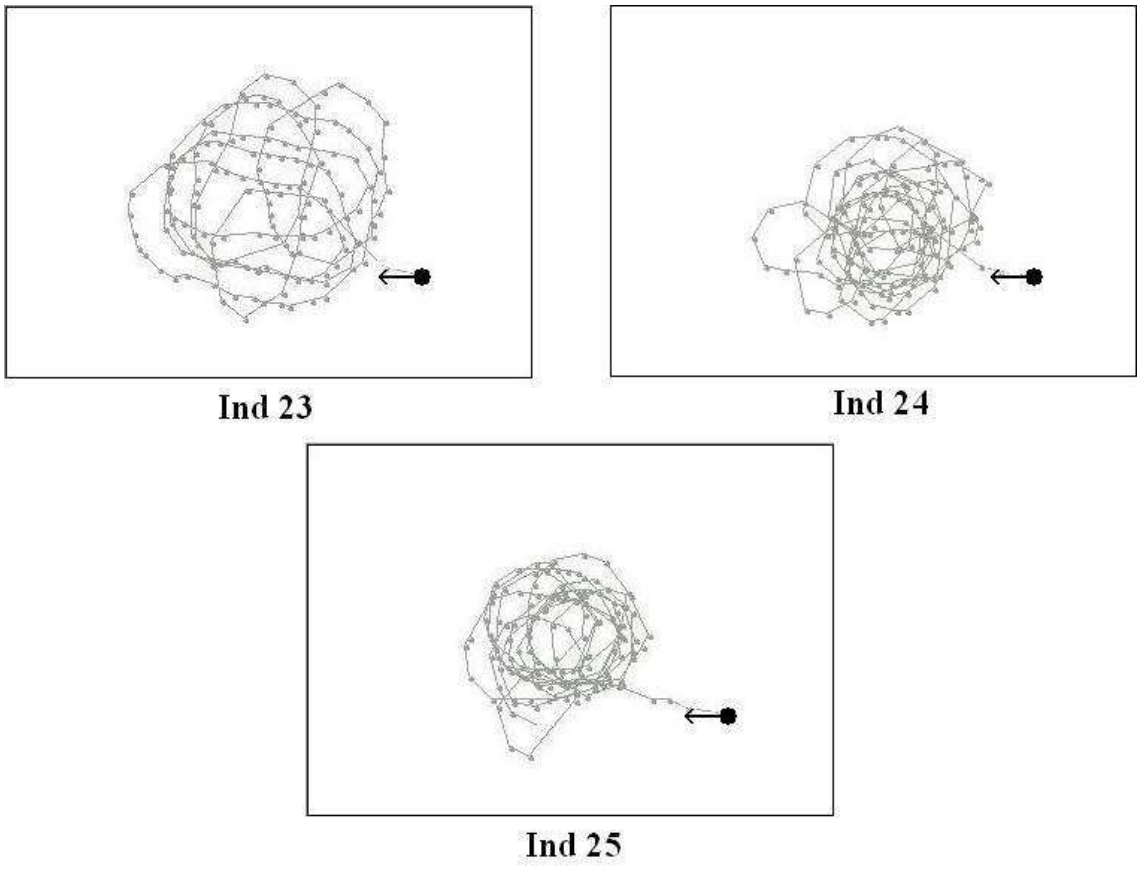


Figura 49: Trayectorias de los 3 últimos individuos evolucionados (Prueba 2).

Tabla XI: Valores de los tres últimos individuos obtenidos durante la evaluación (Prueba 2).

Individuo	Velocidad promedio	Distancia recorrida	Aptitud
23	136 mm/seg	54.335 mts	0.05571
24	138 mm/seg	52.483 mts	0.04864
25	137 mm/seg	56.328 mts	0.05895

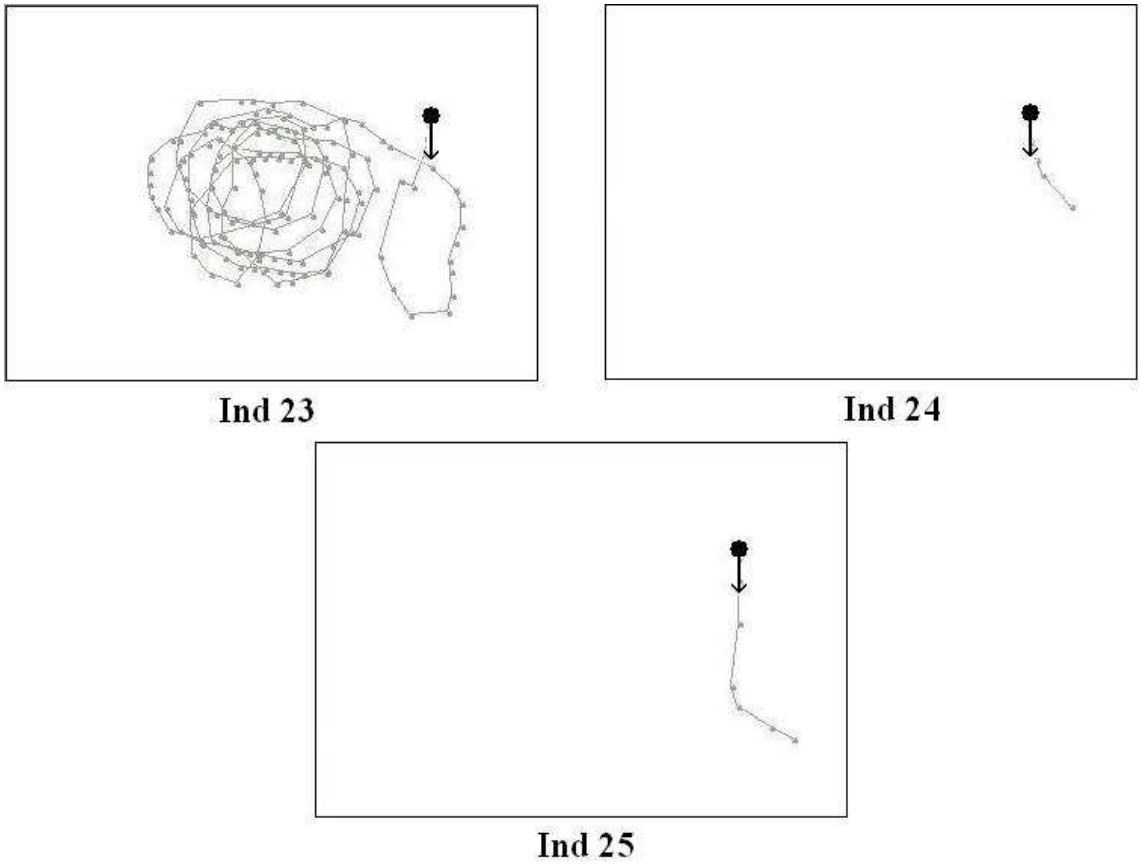


Figura 50: Trayectorias de los 3 últimos individuos evolucionados (Prueba 3).

Tabla XII: Valores de los tres últimos individuos obtenidos durante la evaluación (Prueba 3).

Individuo	Velocidad promedio	Distancia recorrida	Aptitud
23	132 mm/seg	54.255 mts	0.05347
24	218 mm/seg	0.788 mts	0.00075
25	186 mm/seg	1.476 mts	0.00147

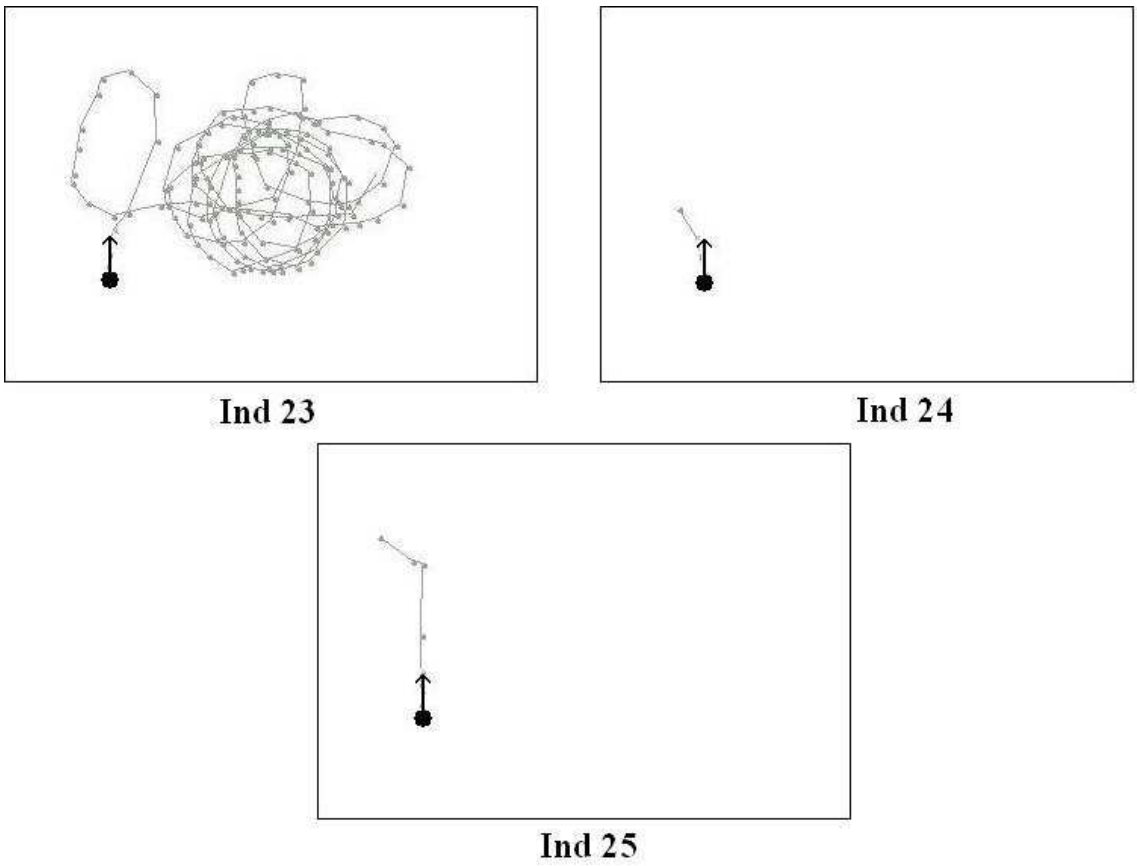


Figura 51: Trayectorias de los 3 últimos individuos evolucionados (Prueba 4).

Tabla XIII: Valores de los tres últimos individuos obtenidos durante la evaluación (Prueba 4).

Individuo	Velocidad promedio	Distancia recorrida	Aptitud
23	138 mm/seg	54.310 mts	0.05688
24	228 mm/seg	0.663 mts	0.00076
25	169 mm/seg	1.452 mts	0.00131

las pruebas fue el número 23. Por lo tanto este individuo es el más apto de los tres para navegar ya que es capaz de resolver de manera exitosa las situaciones adversas a las que se le enfrentó a diferencia de los otros dos.

Capítulo VII

Evolución de navegación y dinámica de recarga con visión en el Pioneer P2-AT

VII.1 Introducción

En el capítulo anterior hemos presentado un conjunto de experimentos de robots reactivos controlados por redes neuronales simples (i.e., perceptrones conectados completamente sin capas internas y sin ninguna forma de organización interna). En este capítulo describiremos la evolución de un comportamiento más complicado que el del capítulo anterior. La navegación con dinámica de recarga consiste en una actividad donde el robot es puesto en un ambiente a navegar con una limitado, pero recargable, suministro de energía. Describiremos también la evolución de las habilidades para localizar un cargador de batería y regresar periodicamente al cargador sin introducir indicaciones explicitas en la función de aptitud.

Una manera de enriquecer el poder computacional de un neuro-controlador es proveerlo con conexiones recurrentes. Este tipo de redes podrían tomar en cuenta estados previamente experimentados sensorialmente y por lo tanto reaccionar de manera diferente a los mismos estados sensoriales. Por lo tanto, estos robots podrían desplegar dinámicas internas.

VII.1.1 Navegación y dinámica de recarga de batería con visión

Para la realización de esta tarea el robot Pioneer P2-AT, equipado con energía limitada —pero recargable—, es puesto a navegar en un ambiente. El ambiente incluye un cargador de batería que se encuentra en un área con rostros en la pared, cuando el robot se encuentra a una distancia de los rostros al mismo tiempo que los observa la batería se recarga. Dado que la evolución se lleva a cabo en el simulador este proceso es simulado también. Aunque la función de aptitud no especifica la localización de la estación de recarga de la batería o el hecho de que el robot debería alcanzarla, el robot evoluciona la habilidad de encontrar y regresar a la estación de recarga mientras explora el área.

El ambiente en el que se moverá el robot es un espacio rectangular delimitado por 5 paredes definiendo una área de 3.52 x 2.48 mts. aproximadamente (Figura 52). El área de recarga también se muestra en la Figura 52, cuando el centro del robot entra a esta área y su orientación le permite ver la pared donde se encuentran las caras, el nivel de su batería se recarga a un nivel de 1.0, este nivel disminuye 0.02 a cada ciclo que pasa. La batería que el robot utiliza es una simulación de una batería ya sea en el robot simulado o en el real, de otra manera sería muy difícil controlar el nivel de descarga por ciclo y su recarga al pasar por el área de recarga ya que el Pioner P2-AT cuenta con baterías con duración de hasta 4 horas.

El robot Pioneer P2-AT esta equipado con un conjunto de 12 sonares; 8 sonares al frente y 4 en la parte posterior. Adicionalmente el robot cuenta con una cámara con un campo de visión de 48.8° (Figura 53). Cada vez que la cámara está viendo las caras se genera una valor igual a 1 de lo contrario, genera un valor igual a 0, este proceso es simulado durante la evolución. Cuenta también con un sensor que mide el

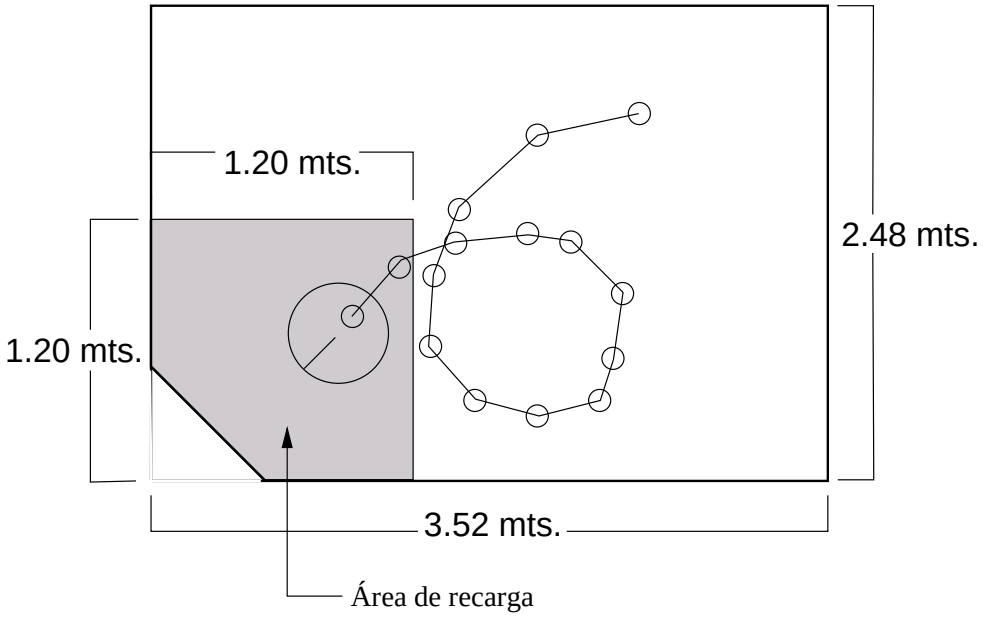


Figura 52: Vista del ambiente en el que se desenvolverá el robot Pioneer P2-AT.

nivel de energía del robot durante su vida. La batería es simulada, como se mencionó anteriormente, y se caracteriza por una tasa de descarga rápida lineal con una duración máxima de 50 ciclos.

VII.1.2 Módulo de simulación de visión

Para simular la visión del robot en el simulador se programó un módulo en lenguaje C, este módulo toma como parámetros la orientación y posición del robot en un momento dado, el campo de visión de la cámara que es de 48.8°, y las paredes que conforman el ambiente.

Lo primero es determinar los límites del campo de visión en un momento dado, dada la orientación del robot en ese momento. Los límites del campo de visión trazan un par de líneas que se cruzan en el centro del robot. Sólo los segmentos del punto de cruce hacia el frente del robot son considerados como se muestra en la Figura 54. Así estas líneas se intersectan con las paredes en el ambiente. Para determinar en donde se

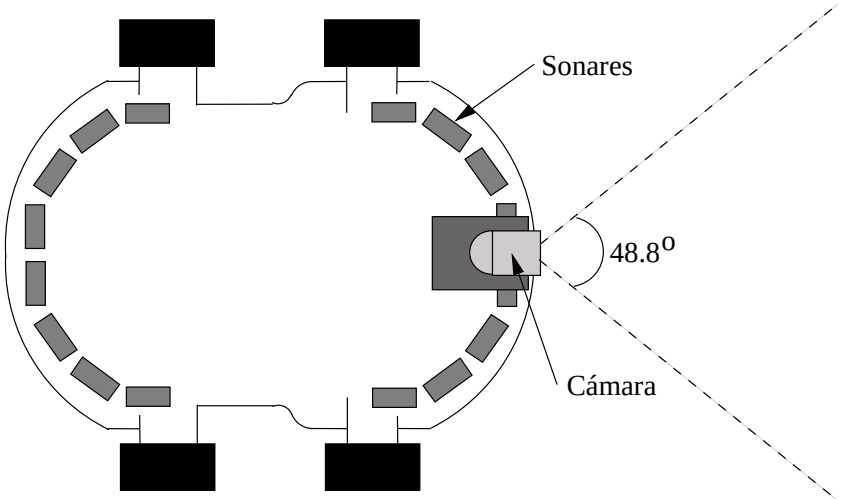


Figura 53: Robot Pioneer P2-AT con cámara integrada.

intersectan las líneas del campo de visión y las paredes se hace revisando ambas líneas por separado y pared por pared de la siguiente manera:

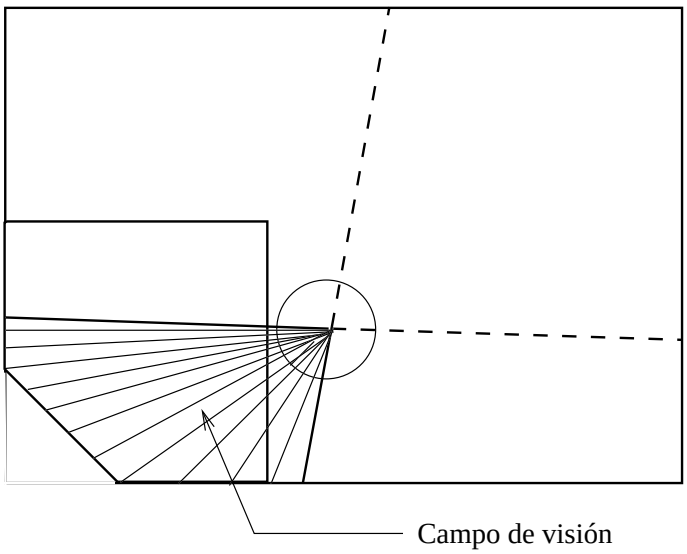


Figura 54: Simulación del campo de visión del robot Pioneer P2-AT.

Para el límite derecho, a la orientación del robot le restamos 24.4°, que representa la mitad del campo de visión, y para el límite izquierdo se los sumamos. Con esto calculamos la pendiente de las líneas determinando la tangente de su ángulo. Además, como estas líneas se cruzan en el centro del robot, también conocemos un punto por

el que pasan las líneas, en donde este punto corresponde a la posición del robot en ese momento. Ahora podemos determinar el punto donde se cruzan los límites del campo de visión y las paredes.

La pared vertical izquierda, es una línea con $x = 0$ y $0 \leq y \leq 248$, el rango donde está definida y representa la longitud de la pared en centímetros. Lo que necesitamos saber es cuál es el punto (x_1, y_1) de la línea límite que se intersecta en esta pared. Para determinar este punto se debe calcular el valor de y_1 cuando $x_1 = 0$ ya que se sabe de antemano que x_1 vale cero a lo largo de toda la pared vertical izquierda. Para esto utilizamos la siguiente fórmula:

$$y_1 = pos_y + m * (x_1 - pos_x), \tag{63}$$

donde x_1 es igual a 0, pos_x y pos_y son las coordenadas de la posición del robot (un punto en la línea) y m es la pendiente de la línea. Para la pared vertical derecha se utiliza la misma ecuación, pero el valor de x_1 es igual a 358. Para ambos casos si el valor obtenido de y_1 no está en el rango [0, 248], entonces la línea no se intersecta en esa pared.

La pared horizontal inferior, es una línea con $y = 0$ y $0 \leq x \leq 352$, el rango donde está definida x representa la longitud de la pared en centímetros. Para determinar el punto donde se intersectan la línea límite y la pared se debe calcular el valor de x_1 cuando $y_1 = 0$. Para esto se utiliza la siguiente fórmula:

$$x_1 = ((y2 - pos_y)/m) + pos_x, \tag{64}$$

donde y_1 es igual a 0, pos_x y pos_y son las coordenadas de la posición del robot (un punto en la línea) y m es la pendiente de la línea. Para la pared horizontal superior se utiliza la misma ecuación, pero el valor de y_1 es igual a 248. Para ambos casos si el

valor obtenido de x_1 no está en el rango [0, 352], entonces la línea no se intersecta en esa pared.

De esta manera se obtuvieron los puntos de intersección con las paredes y los dos límites del campo de visión. Se definió un área de recarga, donde se encontrarían caras que el robot reconoce, el área se encuentra en la esquina inferior izquierda del ambiente y abarca 1200 mm. de la pared vertical izquierda y 1200 mm. de la pared horizontal inferior como se muestra en la Figura 54. Cuando uno de los puntos de intersección se encuentra dentro de esta área definida quiere decir que el robot está viendo una cara y el robot envía una señal que en este caso es un valor 1, de lo contrario envía un 0.

VII.1.3 Función de aptitud y comportamiento

La función de aptitud que utilizamos es una versión de la que empleamos anteriormente, la cual intentamos simplificar para dar más libertad a la evolución. De esta forma el comportamiento esperado no está especificado en su totalidad en la función de aptitud, por lo que se espera que el comportamiento emerja de los cambios en los parámetros de movimiento del robot y en las características del ambiente. Es decir, cuándo el robot descubre la presencia de un lugar donde puede recargar sus baterías y modificar su comportamiento. La función de aptitud que utilizamos es la siguiente:

$$\begin{aligned} \Phi = (1 - (\mid \theta \mid /90) * 0.5) * (\nu/500) * (0.7 + (\delta/200) * 0.3) * \mu \\ -90 \leq \theta \leq 90 \\ 0 \leq \nu \leq 500 \\ 0 \leq \delta \leq 200 \\ 0 \leq \mu \leq 1 \end{aligned} \tag{65}$$

donde θ es la cantidad de grados que el robot voltea, ν es la velocidad que el robot toma, δ es la distancia que el robot recorre, y μ es igual a la lectura normalizada del sonar que obtiene la menor distancia a una de las paredes en el caso de que esta

lectura sea menor que 400 mm. Si es mayor que 400 mm μ vale 0.5. Si es mayor que 600 mm μ vale 0.7, y si es mayor a 1000 mm μ vale 1.0, es decir no afecta a la función de aptitud. Esta función se evalúa cuando un nuevo ciclo motor-sensorial se ejecuta y los valores resultantes se suman y dividen por el número total de ciclos motor-sensoriales durante los cuales el individuo se probó. Estos cuatro componentes alimentan —respectivamente— el desplazamiento recto, la velocidad, la distancia, y la evasión de obstáculos.

La función de aptitud Φ tiene cuatro componentes: el primero maximizado por el desplazamiento recto (sin vueltas), el segundo por la velocidad, el tercero por la distancia, y el cuarto por la evasión de obtáculos. Los valores de peso que se le dieron a cada componente definen su importancia para el comportamiento. El primer componente tiene un peso de 0.5, lo que quiere decir que no es tan importante que el robot trate de seguir una trayectoria recta, si gira no será penalizado tan severamente como en el experimento anterior. El segundo componente tiene un peso de 1, el máximo peso, esto se hizo para acelerar la evolución ya que los individuos lentos toman mucho tiempo y detienen la evolución. El tercer componente tiene un peso de 0.3, muy pequeño ya que no interesa mucho la distancia que recorra el robot. Además si recarga y sigue con vida la distancia que recorrerá será mayor y ya no depende completamente de la función de aptitud. El cuarto componente tiene un peso de 1 ya que es primordial que el robot no choque. Un análisis detallado de esta función se encuentra en el capítulo anterior.

Cabe mencionar además que el robot encuentre un comportamiento que le permita regresar a recargar sus baterías al lugar de recarga depende de la información presente en el ambiente más que de la función de aptitud.

VII.1.4 Arquitectura del neuro-controlador

El sistema de control del robot sigue siendo una red neuronal. Consiste en un perceptron multicapa de unidades sigmoidales. La capa oculta consiste en 5 unidades con conexiones recurrentes a la entrada. La capa de salida consiste en 3 unidades las cuales corresponden a los parámetros de los tres mandos básicos para mover al robot. Además, esta capa tiene conexiones recurrentes a la entrada del neuro-controlador. Existen 12 unidades de entrada que corresponden a cada una de las lecturas de sonar. También tiene una unidad que corresponde a la carga de la batería y otra más para el módulo de visión. Cada unidad de salida (unidad de mando) y de entrada tiene un peso sináptico adicional evolucionable desde una unidad de predisposición (bias); la Figura 55 muestra la arquitectura del neuro-controlador.

Para encontrar el neuro-controlador más apropiado, al igual que en el experimento anterior se utilizó un algoritmo evolutivo.

VII.1.5 Algoritmo evolutivo

El algoritmo evolutivo se planeó para correr 40 generaciones, con una población de 40 individuos. Cada individuo en la población corresponde a un neuro-controlador cuyo desempeño es probado en el ambiente. Cada individuo se prueba cinco veces (épocas). Antes de comenzar cada época el robot se mueve a una posición diferente a la que tiene cuando termina la época anterior, después el robot comienza con una batería completamente cargada la cual se descarga una cantidad determinada cada ciclo de vida; una batería completamente cargada permite al robot moverse por 50 ciclos. Cada época tiene una duración de 50 ciclos, a menos que el robot pase por la zona de recarga, donde la batería es instantaneamente recargada y por consiguiente su vida se prolonga. Con el fin de terminar eventualmente con la vida de los robots

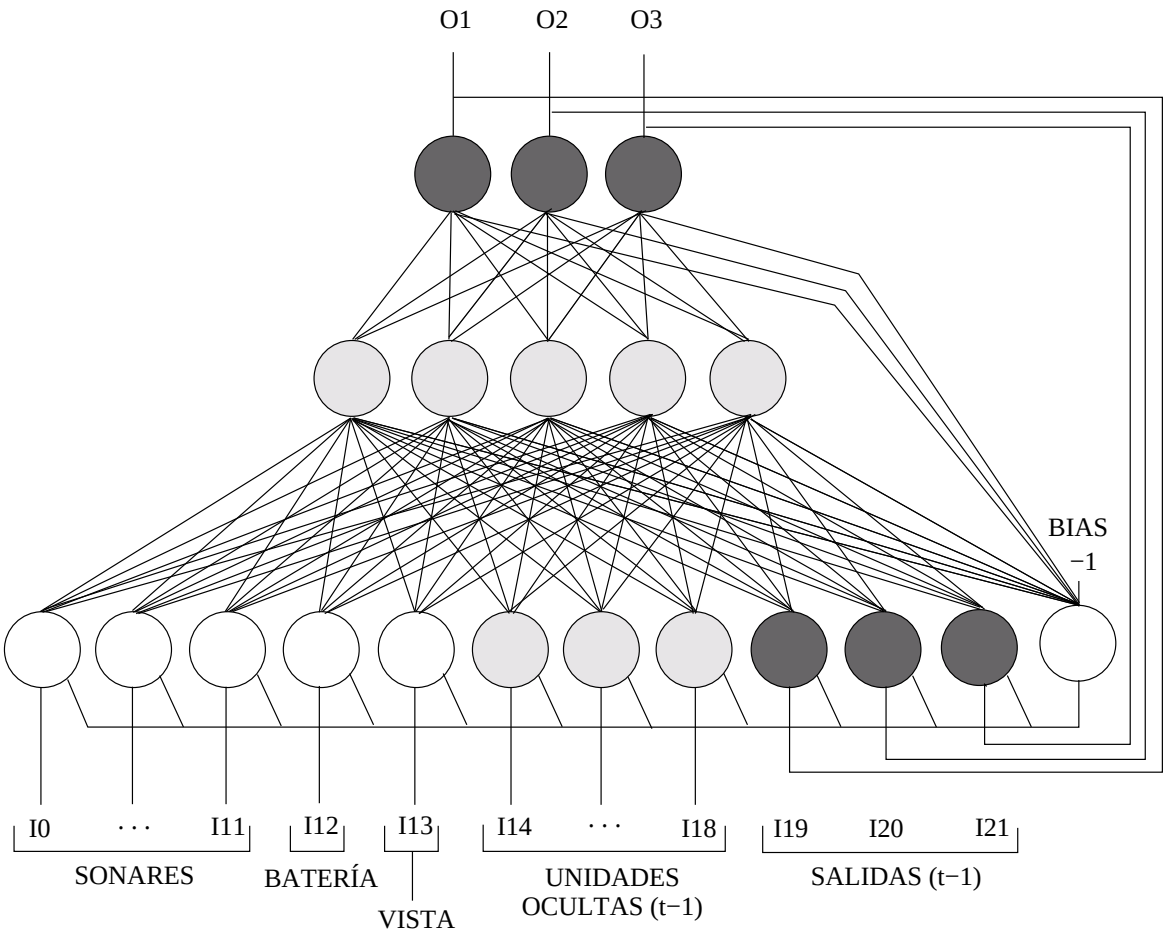


Figura 55: Arquitectura del neuro-controlador evolucionado.

que permanecen en el área de recarga o que pasan regularmente por allí, se permite un límite de diez recargas por época, es decir, después de que el robot haya recargado en diez ocaciones ya no se le permitirá racargarse de nuevo durante esa época. Un ciclo es el periodo de tiempo en el que el robot ejecuta los tres mandos: speed(), move(), y turn(). Al finalizar cada ciclo el algoritmo calcula el desempeño o aptitud Φ del robot durante ese ciclo, estas aptitudes se acumulan durante toda la época. Al finalizar la época (los 50 ciclos) la aptitud total obtenida se divide por el número de ciclos que duró la época, en este caso 50 ciclos. Si el robot choca con una pared durante una época, la prueba se suspende en ese ciclo, y se le asigna una aptitud de cero a los ciclos restantes, entonces se continúa con la época siguiente. Una vez que los 40 individuos de la población han sido probados, y ya se cuenta con sus aptitudes se selecciona a los ocho individuos con mayor aptitud de la población. De estos ocho individuos se desprenderá una nueva población, a cada uno de estos individuos se les aplicará una mutación con una taza de 3% para obtener cinco hijos de cada individuo seleccionado, lo que nos da un total de 40 individuos nuevos. En la Figura 45 en el capítulo anterior se muestra el pseudo-codigo del algoritmo evolutivo.

Para cada generación que toma una duración de 2 horas al principio y aumenta mientras la evolución avanza, almacenamos la aptitud promedio de la población y la aptitud del mejor individuo en la población en el archivo statS1.fit en el directorio /usr/local/Saphira/bin/ (Figura 56). Un valor de aptitud de 1.0 podría ser alcanzado sólo por un robot moviendose de manera recta a una velocidad máxima en un espacio abierto, pero un robot que recargue vivirá más tiempo por lo tanto es más apto aunque esto no se refleje directamente en la función de aptitud. En el ambiente mostrado en la Figura 52, donde algunos de los sensores casi siempre estuvieron activos (con valores menores a 1.0 ya normalizados) y donde muchas vueltas fueron necesarias para navegar, 0.403 fue el máximo valor obtenido por el controlador evolutivo.

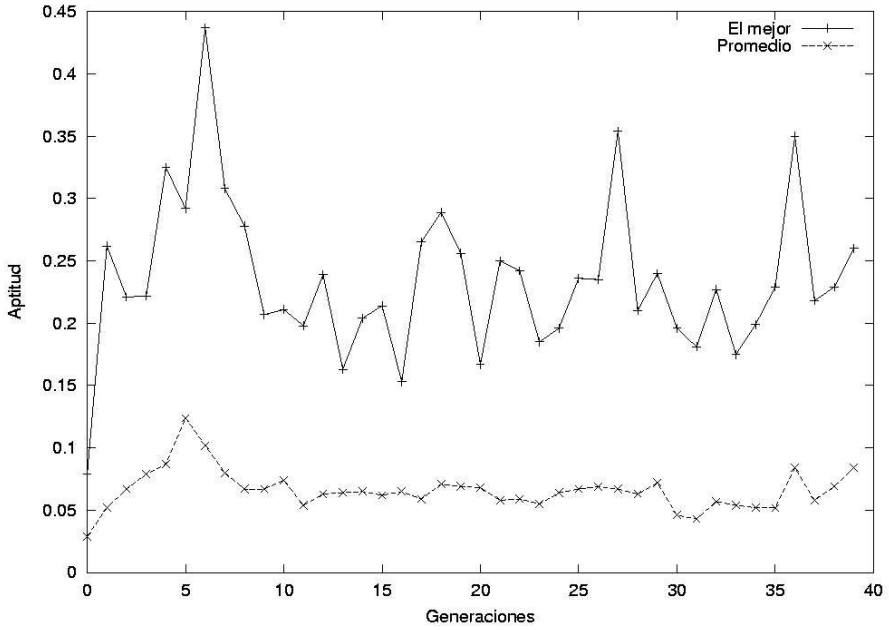


Figura 56: Aptitud promedio de la población y la aptitud del mejor individuo en la población graficada sobre cada generación.

Una vez que la evolución ha terminado, probamos al mejor individuo obtenido durante la evolución en el robot real, en esta evolución el individuo con mayor aptitud es el número 6 (Figura 56). En la prueba el robot parte de la esquina superior derecha en el ambiente con una orientación hacia el oeste, esta posición esta alejada de la zona de recarga ubicada en la esquina contraria (inferior derecha), la trayectoria que describió este robot se muestra en las Figuras 57 y 58, también se muestran las 10 recargas de energía que hizo y las imagenes que capturadas por la cámara del robot en el momento de la recarga.

La trayectoria que sigue el robot lo desplaza visiblemente a la zona de recarga, donde recarga su batería y consigue vivir durante 110 ciclos, más del doble que la duración de la batería (50 ciclos).

Este experimento fue realizado en el robot real Pioneer P2-AT en un ambiente dispuesto en las instalaciones del CICESE. El neuro-controlador obtenido de la evolución

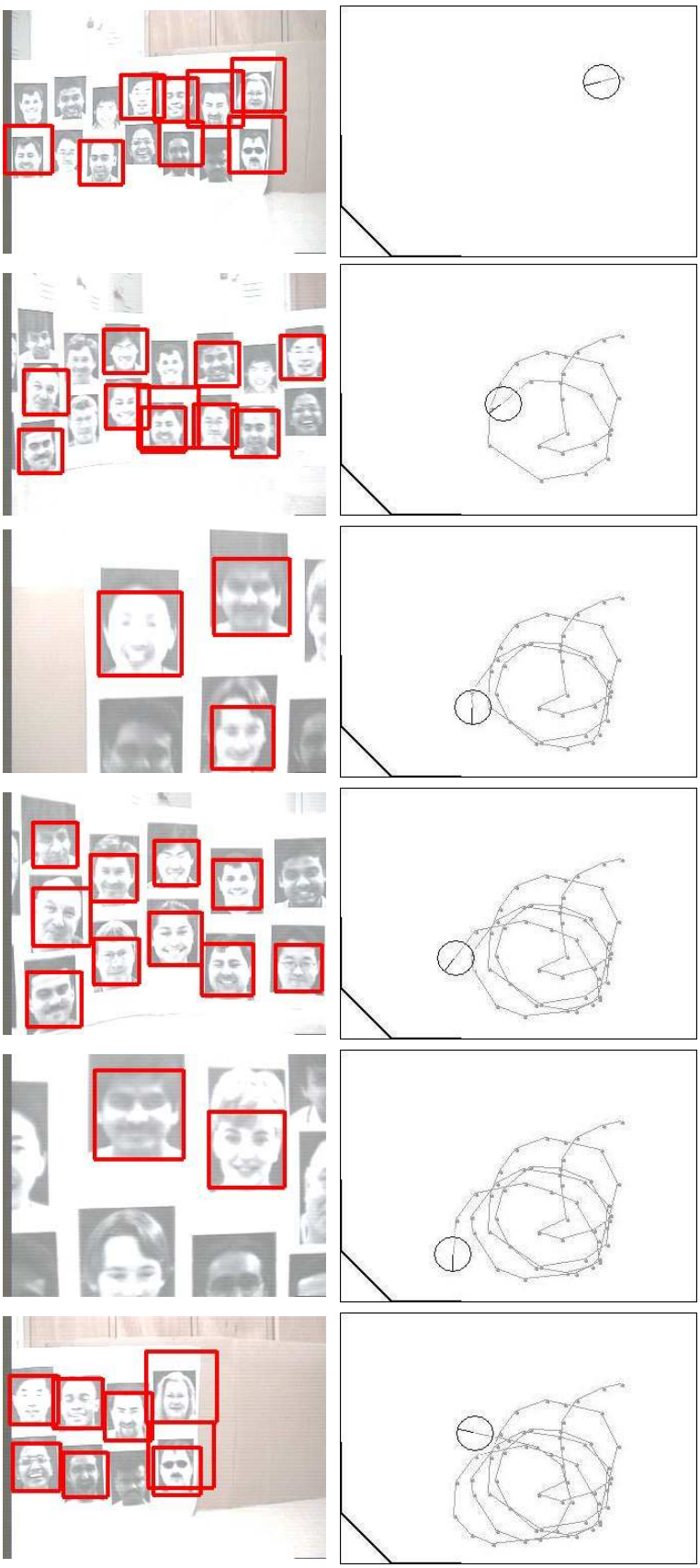


Figura 57: Posición inicial y cinco primeras recargas.

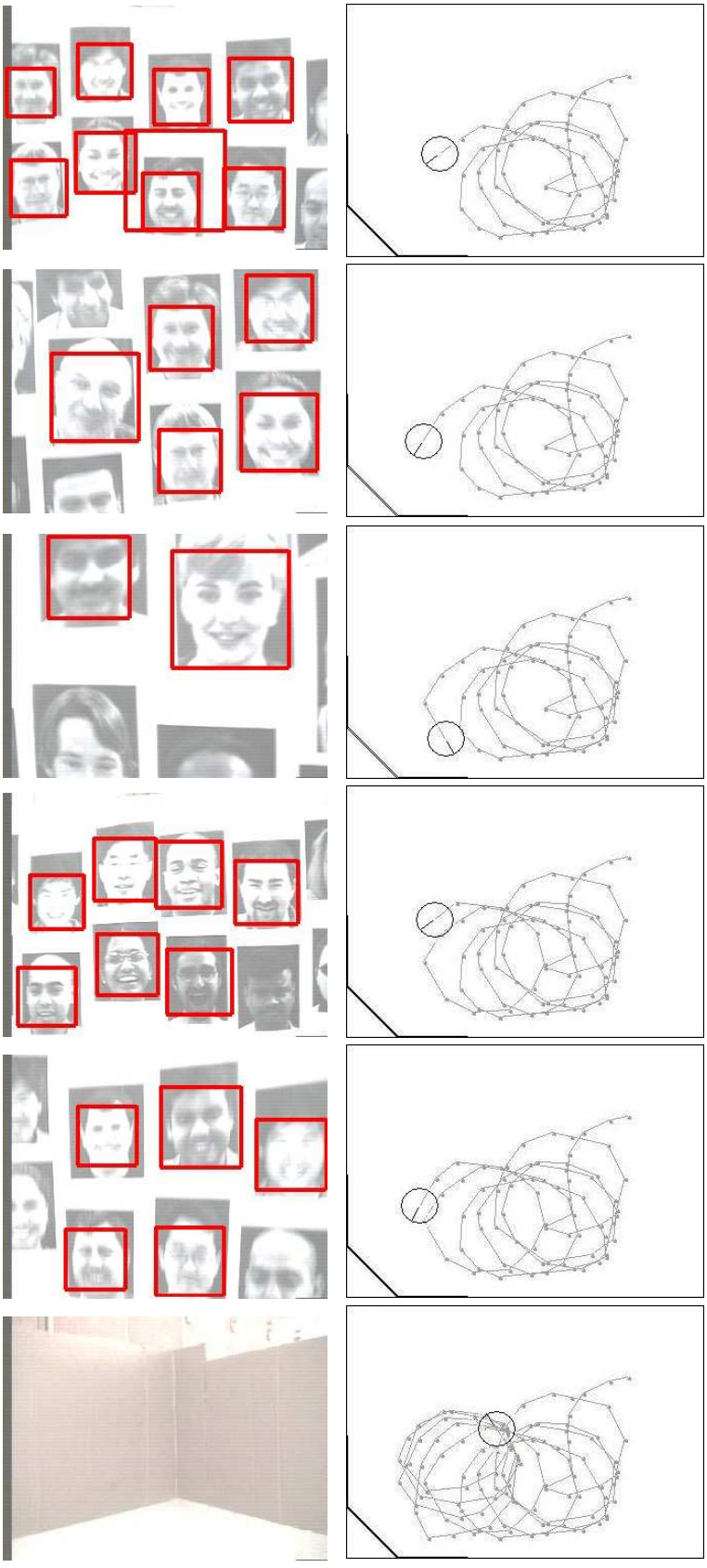


Figura 58: cinco recargas finales y posición final.

en simulación se adaptó completamente al cambio a la realidad, en la mayoría de los experimentos de robótica evolutiva es necesario hacer uso de la evolución incremental y continuar evolucionando en el robot real hasta obtener un individuo aceptable. En nuestro experimento esto no fue necesario, nosotros utilizamos directamente al individuo obtenido mediante la evolución en simulación sobre el robot real. Además si mientras el robot real está navegando el ambiente sufre pequeños cambios el robot responde favorablemente a dichos cambios tomando las acciones necesarias. Todo esto se debe a la robustez del simulador Pioneer que emula de una manera veraz la realidad.

Capítulo VIII

EvoPioneer

VIII.1 Introducción

El EvoPioneer es un programa para correr experimentos de robótica evolutiva. El evopioneer corre sobre el sistema Red Hat 7.3. Los archivos fuente están escritos en lenguaje C, y en lenguaje Colbert, cualquier robot que utilice la arquitectura Saphira puede ser evolucionado por el EvoPioneer aunque fue pensado para evolucionar un robot Pioneer P2-AT.

El EvoPioneer permite evolucionar experimentos en un robot simulado o real. Para correr experimentos evolutivos en el robot real o para probar individuos evolucionados en la simulación en el robot real.

El EvoPioneer está inspirado en el programa Evorobot desarrollado por Stefano Nolfi y Dario Floreano para evolucionar experimentos para un robot Khepera. Esta aplicación permite replicar muchos de los experimentos descritos en Nolfi y Floreano , 2000 y experimentos propios también. Evorobot corre sobre Windows95/98 y WindowsNT. Los archivos fuente están escritos en C y C++. Para comenzar el entendimiento de este programa se dio a la tarea de hacer los cambios necesarios para que corriera sobre Linux, cosa que no fue fácil, se cambiaron todas las intrucciones de interface entre otras cosas pero el resultado final fue muy satisfactorio, estamos hablando como de 12,000 líneas de código que revisar. En las Figuras 59 y 60 se muestra la interface principal resultado de la programación para que corriera en Linux que no es muy diferente a la que se puede ver en windows.

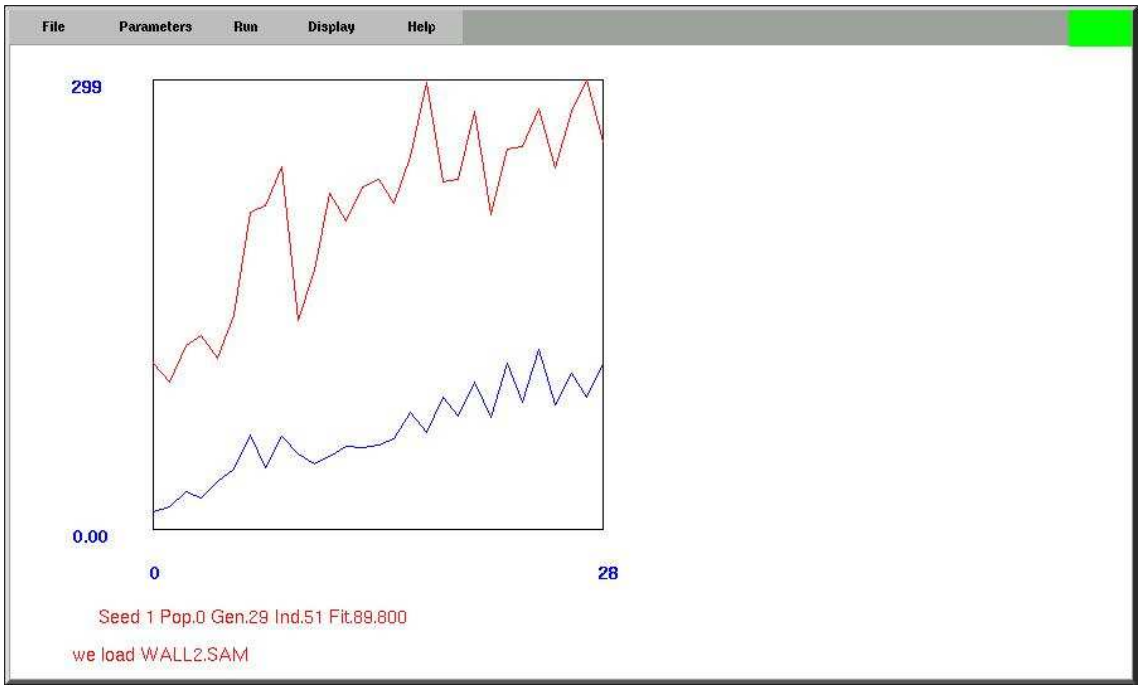


Figura 59: Programa Evorobot en su versión modificada durante el proceso de evolución.

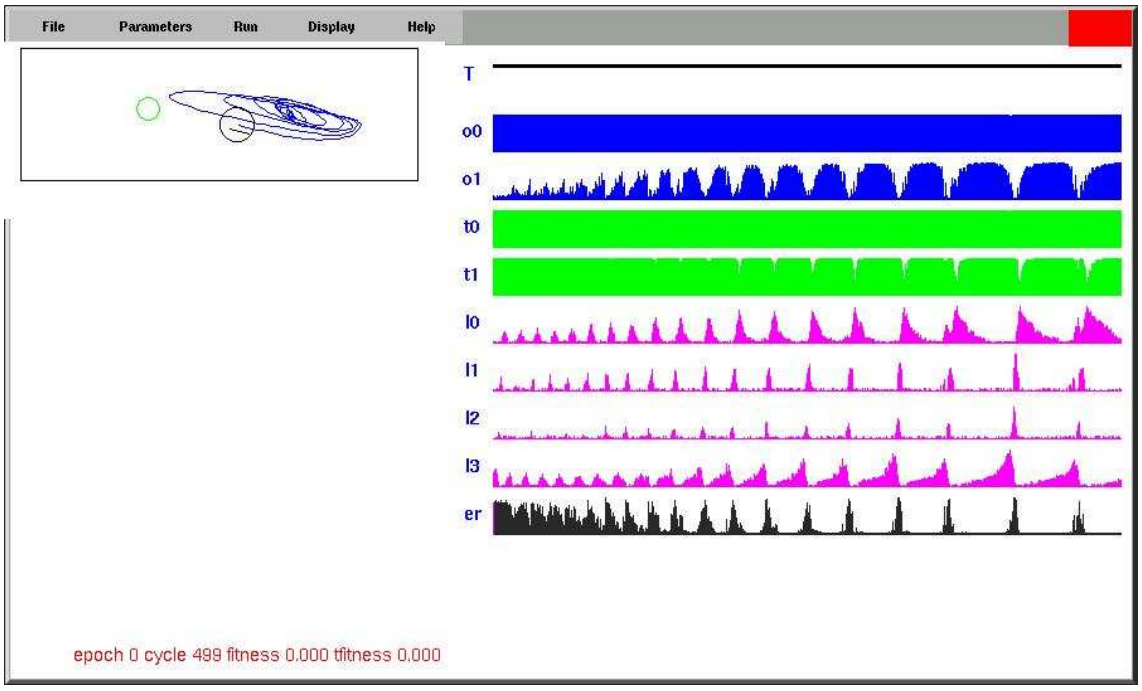


Figura 60: Programa Evorobot en su versión modificada durante el proceso de prueba de un individuo en el simulador.

En este capítulo sólo describiremos al EvoPioneer, su implementación y uso.

VIII.2 Implementación

Se cuenta con un robot PioneerP2-AT con arquitectura Saphira, queremos evolucionar un neuro-controlador para que este robot realice ciertas tareas.

Para manejar actividades complejas de meta-búsqueda, Saphira proveee un método de calendarización de acciones del robot utilizando un lenguaje de control, llamado Colbert. De esta forma, se pueden construir librerías de actividades que realice acciones de manera secuecial del robot en respuesta a condiciones ambientales. Por ejemplo, una actividad típica podría mover al robot en un corredor mientras evade obstáculos.

Los *esquemas de actividad* son los bloques básicos de contrucción de Colbert. Cuando se ejecuta un esquema de actividad, es calendarizado como una micro-tarea con facilidades avanzadas para engendrar actividades hijas y comportamientos, y coordinar acciones entre actividades corriendo concurrentemente.

Los esquemas de actividad están escritos en lenguaje Colbert. El lenguaje tiene un conjunto rico de conceptos de control, y una sintaxis de usuario amigable, similar a la de C, que hace la escritura de actividades más fácil. Debido a que el lenguaje es *interpretado*, es más fácil desarrollar y depurar actividades, porque los errores pueden ser detectados. Una actividad puede ser cambiada en un editor de texto, y entonces ser reinvocada, sin dejar de correr la aplicación.

Aunque Colbert tiene ventajas sobre otros lenguajes de control, Colbert no tiene toda la funcionalidad de un lenguaje de programación como C, por eso es conveniente entender la base de Colbert adhiriendo interfaces a funciones, variables y objetos de C/C++.

Para cargar rutinas nuevas en C/C++ dentro de Colbert, se debe escribir uno o

más programas en C/C++ que contengan nuestras propias funciones, y hacer llamadas a las rutinas de la librería de Saphira. Note que existen muchas funciones que permanecen en Saphira que son funciones en C. Gracias a esta interface entre funciones escritas en C y las actividades en Colbert se extiende mucho su funcionalidad. Nosotros utilizamos esta interface para escribir funciones importantes del algoritmo evolutivo en C, y despues llamarlas desde una actividad en Colbert que conformará el algoritmo evolutivo completo. El 97% del algoritmo evolutivo está escrito en C.

VIII.3 Funcionamiento

El EvoPioneer lo primero que hace es cargar un archivo PIONEE.CF que se encuentra en el directorio Saphira/tutor/evopioneer/. Este archivo es el archivo de configuración y contiene todos los parámetros que pueden ser establecidos por el usuario. Entre estos parámetros están: número de generaciones, épocas, ciclos, individuos padre, individuos hijos, taza de mutación, función de aptitud, y otros sobre la arquitectura de la red neuronal: número de unidades ocultas, recurrencia, entradas, salidas, entre otros.

El archivo de configuración establece e inicializa todos los parámetros que ocupa el algoritmo evolutivo antes de comenzar la evolución. Después de cargar el archivo de configuración se inicializan todas las variables. Ahora el algoritmo evolutivo está listo para empezar.

La evolución comienza y el algoritmo guarda el mejor individuo de cada generación en el archivo B1P0S1.gen, también guarda las aptitudes de todos los individuos de cada generación en el archivo FitPOS1.txt y guarda las mejores aptitudes y las aptitudes promedio por generación en el archivo statS1.fit. Estos tres archivos se almacenan en el directorio Saphira/bin/.

Para ejecutar el algoritmo evolutivo es necesario correr el simulador Pioneer y la

aplicación Saphira, mostrados en la figura 61. Una vez establecida la conexión entre Saphira y el simulador, cargando previamente los archivos de parámetros del robot y del mundo en el simulador, se ejecuta la actividad flloc.act con el mando "load flloc". Esta actividad carga el mundo o ambiente y su función principal es la de mantener localizado al robot en el ambiente. Por último se corre la actividad evopioneer.act, la cual contiene el algoritmo evolutivo, con el mando "load evopioneer", este mando inicializará la evolución.

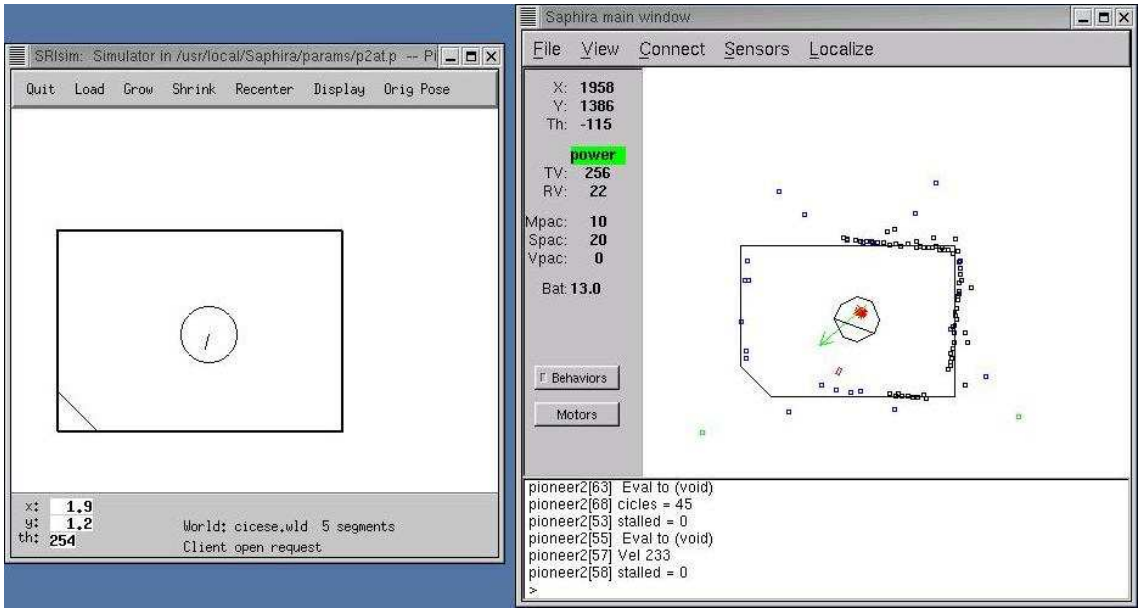


Figura 61: Interfaces principales del simulador Pioneer y de Saphira.

VIII.3.1 Descripción del archivo PIONEE.CF

Cada experimento debe tener su archivo de configuración PIONEE.CF, es decir sus parámetros. Un ejemplo de un archivo de configuración se muestra en la Figura 62 y a continuación se da una descripción detallada de este archivo.

Primer renglón

```
B8 C5
2 m H5 e f n F2
G40 o3
e5 s50
```

Figura 62: Archivo de configuración PIONEE.CF para el segundo experimento de navegación con dinámica de recarga.

B# La letra B seguida del número que establece la cantidad de individuos padre que se utilizarán en la evolución.

C# La letra C seguida del número que establece la cantidad de hijos por cada padre en la evolución. El tamaño de la población no es determinado directamente por el usuario, sino que se obtiene de la multiplicación del número de padres e hijos.

Segundo renglón

2 Indica que se utilizarán las lecturas de 12 sonares para el neuro-controlador, los 8 sonares frontales y los 4 sonares centrales posteriores del Pioneer P2-AT.

m La letra m indica que los neuro-controladores tienen tres unidades de entrada adicionales que almacenan el estado de las tres salidas del controlador en el ciclo previo.

H# La letra H seguida del número de unidades escondidas que se desea tener en el neuro-controlador del robot.

e Esta letra indica que los neuro-controladores tienen unidades de entrada adicionales que almacenan el estado de las unidades ocultas en el ciclo previo.

f Esta letra indica que los neuro-controladores tienen una entrada adicional para la información arrojada por el modulo de visión.

n Los individuos son dotados con un nivel de energía simulado. La energía dura 50 ciclos, el nivel de energía del individuo alimenta una entrada del neuro-controlador.

F# La letra F seguida del número de función de aptitud a usar, hasta ahora hay sólo dos funciones de aptitud determinadas una para cada uno de los experimentos desarrollados en esta investigación.

Tercer renglón

G# La letra G seguida de un número que indica el número de generaciones de la evolución.

o# La letra o seguida de número que indica la taza de mutación en porcentaje.

Cuarto renglón

e# La letra e seguida de un número que indica la cantidad de épocas que vivirá un individuo.

s# La letra s seguida de un número que indica la cantidad de ciclos que durará cada época.

Todos estos parámetros deben estar presentes para que la evolución se lleve a cabo.

Capítulo IX

Concluciones, aportaciones y trabajo futuro

IX.1 Concluciones

Los resultados experimentales demuestran que el enfoque planteado es capaz de evolucionar comportamientos con diferente grado de complejidad. En esta investigación evolucionamos dos comportamientos y con el "EvoPioneer" es posible evolucionar más comportamientos definiendo su función de aptitud y el ambiente en el que se desenvolverá.

El experimento de navegación y dinámica de recarga de batería mediante visión da cuenta de que la evolución está intimamente ligada a las características del ambiente y es capaz de explotarlas para el beneficio de los individuos. El individuo fue capaz de encontrar la zona de recarga y dirigirse hacia ella sin que esto estubiera expresado de manera explísita en la función de aptitud. También demuestra que la aplicación de la visión por computadora permite potencializar el uso de las características en el ambiente que no pueden ser apreciadas por otros medios.

El neuro-controlador obtenido de la evolución en simulación se adaptó completamente al cambio a la realidad, en la mayoría de los experimentos de robótica evolutiva es necesario hacer uso de la evolución incremental y continuar evolucionando en el robot real hasta obtener un individuo aceptable. En nuestro experimento no fue necesario, nosotros utilizamos directamente al individuo obtenido mediante la evolución en simulación sobre el robot real. Además si mientras el robot real está navegando el

ambiente sufre pequeños cambios el robot responde favorablemente a dichos cambios tomando las acciones necesarias. Todo esto se debe a la robustez del simulador Pioneer de Saphira que emula de una forma veraz la realidad.

IX.2 Aportaciones

Aplicamos la robótica evolutiva para obtener comportamientos en un robot grande y complejo como el Pioneer P2-AT, distinto a los trabajos anteriores propuestos por Nolfi y Floreano , 2000. También conseguimos obtener buenos comportamientos en un lapso de tiempo razonable, los resultados obtenidos son dignos de todo el tiempo que tomaron. Finalmente incorporamos el uso de visión por computadora al proceso de evolución.

IX.3 Trabajo futuro

En el futuro se pretende aumentar la complejidad de los comportamientos a evolucionar y utilizar y agregar nuevas técnicas de visión por ordenador a la evolución, por ejemplo, información estéreo, detección de puntos de interés, etc. En lo que se refiere a la simulación podemos agregar información visual para este tipo de evoluciones mediante el uso de un simulador 3D.

Bibliografía

Ackley, D. y Littman, M. 1992. "Interactions between learning and evolution". En: Langton, C., Taylor, C., Farmer, J., y Rasmussen, S. (eds.), "Artificial Life 2". Addison-Wesley, Redwood City, CA, 487–509 p.

Arkin, R. C. 1998. "Behavior-based robotics". MIT Press. Cambridge, MA. 491 p.

Barto, A., Sutton, R., y Anderson, C. 1983. "Neuronlike adaptive elements that can solve difficult learning control problems". IEEE Transactions on Systems, Man, and Cybernetics. 13(5):834-846 p.

Barto, A. G., Sutton, R. S., y Watkins, C. J. C. H. 1990. "Sequential decision problems and neural networks". En: "Advances in neural information processing systems II". Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 686-693 p.

Beightler, C., Phillips, D., y Wilde, D. 1979. "Foundations of optimization". Prentice-Hall. Englewood Cliffs, NJ. 300 p.

Bellman, R. E. 1961. "Adaptive control processes: A guided tour". Princeton University Press. Princeton, NJ. 300 p.

Braitenberg, V. 1984. "Vehicles: Experiments in synthetic psychology". MIT Press. Cambridge, MA. 168 p.

Branch, J. W. y Olague, G. 2001. "La visión por computador. una aproximación al estado del arte". DYNA - Revista de la facultad de minas. 1(133):1-16 p.

Broggi, A. y Cattani, S. 0. "An agent based evolutionary approach to path detection for off-road vehicle guidance". sometido a Elsevier Science, 18 April 2005.

Brooks, R. 1992. "Artificial life and real robots". En: "European Conference on Artificial Life". 3–10 p.

Brooks, R. A. 1986. "A robust layered control system for a mobile robot". IEEE journal of Robotics and Autonomation. 2(1):14-23 p.

Brooks, R. A. 1991a. "How to build complete creatures rather than isolated cognitive architectures". En: Vanlehn, K. (ed.), "Architectures for Intelligence". Erlbaum: Hillsdale, NJ, 225-240 p.

Brooks, R. A. 1991b. "Intelligence without reason.". En: Mylopoulos, J. y Reiter, R. (eds.), "IJCAI". Morgan Kaufmann, 569-595 p.

Carpenter, G. A. y Grossberg, S. 1988. "The art of adaptive pattern recognition by a self-organizing neural network.". IEEE Computer. 21(3):77-88 p.

Colombetti, M., Dorigo, M., y Borghi, G. 1996. "Behavior analysis and training: A methodology for behavior engineering". IEEE Transactions on Systems, Man, and Cybernetics - Part B. 26(3):365-380 p.

Cybenko, G. 1989. "Approximation by superpositions of a sigmoidal function". Mathematics of Control, Signals, and Systems. 2:303-314 p.

Dawkins, R. 1986. "The blind watchmaker". Penguin. London. 254 p.

Dorigo, M. y Colombetti, M. 1994. "Robot shaping: Developing autonomous agents through learning.". Artificial Intelligence. 71(2):321-370 p.

Dudek, G. y Jenkin, M. 2000. "Computational principles of mobile robotics". Cambridge University Press. New York, NY, USA. 280 p.

Floreano, D. y Mondada, F. 1996. "Evolution of Homing Navigation in a Real Mobile Robot". IEEE Transactions on Systems, Man, and Cybernetics - Part B: Cybernetics. 26(3):396–407 p.

Hadamard, J. 1954. "The psychology of invention in the mathematical field". Princeton University Press. Princeton,NJ. 145 p.

Harvey, I. 1992. “The saga cross: The mechanics of recombination for species with variable length genotypes.”. En: Manner, R. y Manderick, B. (eds.), “Parallel Problem Solving from Nature 2”. Elsevier, 271- p.

Harvey, I., Husbands, P., y Cliff, D. Agosto 8-12, 1994. “Seeing the light: artificial evolution, real vision”. En: “SAB94: Proceedings of the third international conference on Simulation of adaptive behavior : from animals to animats 3”, Cambridge, MA, USA. MIT Press, 392–401 p.

Harvey, I., Tuci, E., y Quinn, M. 2005. “Evolutionary robotics: A new scientific tool for studying cognition”. Artificial Life. 11(1-2):79-98 p.

Hebb, D. 1949. “The organization of behavior: A neuropsychological theory”. John Wiley and Sons. New York. 335 p.

Holland, J. H. 1992. “Adaptation in natural and artificial systems”. MIT Press. Cambridge, MA, USA. 211 p.

Jakobi, N. 1997. “Half-baked, ad-hoc and noisy: Minimal simulations for evolutionary robotics”. En: Husbands, P. y Harvey, I. (eds.), “Proceedings of the Fourth European Conference on Artificial Life: ECAL97”. MIT Press, 348-357 p.

Jakobi, N., Husbands, P., y Harvey, I. 1995. “Noise and the reality gap: The use of simulation in evolutionary robotics.”. En Morán *et al.* , 1995, 704-720 p.

Khatib, O. 1986. “Real-time obstacle avoidance for manipulators and mobile robots”. International Journal of Robotics Research. 5(1):90–98 p.

Konolige, K. y Myers, K. L. 1996. “The saphira architecture for autonomous mobile robots”. En: Kortenkamp, D., Bonasso, R. P., y Murphy, R. (eds.), “AI-based Mobile Robots: Case studies of successful robot systems”. MIT Press, 211-242 p.

Lund, H., Miglino, O., Pagliarini, L., Billard, A., y Ijspeert, A. 1998. “Evolutionary robotics — a children’s game”. En: “IEEE 5th International Conference on Evolutionary Computation”.

McCulloch, W. y Pitts, W. 1943. "A logical calculus of the idea immanent in nervous activity". Bulletin of Mathematical Biophysics. 5:115-133 p. Es el arículo original sobre las neuronas de McCulloch-Pitts.

Michie, D. 1961. "Trial and error". En: "Science Survey, part 2". Penguin, 129-145 p.

Michie, D. y Chambers, R. 1968. "Boxes: An experiment in adaptive control". En: Dale, E. y Michie, D. (eds.), "Machine Intelligence 2". Oliver and Boyd, Edinburgh, 137-152 p.

Miglino, O., Lund, H. H., y Nolfi, S. 1995. "Evolving mobile robots in simulated and real environments". Artificial Life. 2(4):417-434 p.

Miller, G. F. y Todd, P. M. 1990. "Exploring adaptive agency iii: Simulating the evolution of habituation and sensitization". En: Schwefel, H.-P. y Männer, R. (eds.), "PPSN". Springer, 307-313 p.

Minsky, M. y Papert, S. 1969. "Perceptrons: An introduction to computational geometry". MIT Press. Cambridge, Mass. 275 p.

Morán, F., Moreno, A., Guervós, J. J. M., y Chacón, P. (eds.) 1995. "Advances in artificial life, third european conference on artificial life, granada, españa, junio 4-6, 1995, proceedings", volume 929 of Lecture Notes in Computer Science. Springer.

Nolfi, S. y Floreano, D. 2000. "Evolutionary robotics: The biology, intelligence, and technology of self-organizing machines". MIT Press. Primera edición. Massachusetts, USA. 320 p.

Nolfi, S. y Parisi, D. 1997. "Learning to adapt to changing environments in evolving neural networks". Adaptive Behaviour. 5(1):75-98 p.

Odetayo, M. O. y McGregor, D. R. 1989. "Genetic algorithm for inducing control rules for a dynamic system.". En: Schaffer, J. D. (ed.), "International Conference on Genetic Algorithms". Morgan Kaufmann, 177-182 p.

Radcliffe, N. J. 1991. "Equivalence class analysis of genetic algorithms". Complex Systems. 5(2):183-205 p.

Rosenblatt, F. 1958. "The perceptron: A probabilistic model for information storage and organization in the brain". Psychological Review. 65:386-408 p.

Rosenblatt, F. 1962. "The principles of neurodynamics". Spartan. New York. 215 p.

Rumelhart, D., Hinton, G., y Williams, R. 1986. "Learning internal representations by back-propagating errors". En: Rumelhart, D. y McClelland, J. (eds.), "Parallel Distributed Processing: Explorations in the Microstructure of Cognition". MIT Press, Cambridge, MA, 318-362 p.

Samuel, A. L. 1959. "Some studies in machine learning using the game of checkers". IBM Journal of Research and Development. 3(3):210-229 p.

Scheier, C. y Pfeifer, R. 1995. "Classification as sensorimotor coordination: A case of study on autonomous agents". En Morán *et al.* , 1995, 657-667 p.

Stanley, K. O. y Miikkulainen, R. 2002. "Evolving neural networks through augmenting topologies". Evolutionary Computation. 10(2):99–127 p.

Sutton, R. 1988. "Learning to predict by the methods of temporal differences". Machine Learning. 3:9-44 p.

Sutton, R. S. 1990. "Reinforcement learning architectures for animats". En: "Proceedings of the First International Conference on Simulation of Adaptive Behavior: From Animals to Animats, Septiembre 24-28, 1990, Cambridge, MA". 288-296 p.

Sutton, R. S. 1991. "Planning by incremental dynamic programming.". En: "ML". 353-357 p.

Wagner, G. P. y Altenberg, L. 1996. "Complex adaptations and the evolution of evolvability". Evolution. 50:967-976 p.

Watkins, C. J. C. H. y Dayan, P. 1992. "Q-learning". Machine Learning. 8(3-4):279-292 p.

Watson, R., Ficici, S., y Pollack, J. 1999. "Embodied evolution: A response to challenges in evolutionary robotics". En: Wyatt, J. L. y Demiris, J. (eds.), "Proceedings of the Eighth European Workshop on Learning Robots". 14–22 p.

Widrow, B., Gupta, N., y Maitra, S. 1973. "Punish/reward: Learning with a critic in adaptive threshold systems". IEEE Transactions on Systems, Man, and Cybernetics. 3:455-465 p.

Widrow, B. y Hoff, M. 1960. "Adaptive switching circuits". IRE WESCON Convention Record. 4:96-104 p.

Yao, X. 1999. "Evolving artificial neural networks". Proceedings of the IEEE. 87(9):1423–1447 p.