UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

MAESTRÍA EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

ROBÓTICA EVOLUTIVA: BÚSQUEDA DE COMPORTAMIENTOS DE NAVEGACIÓN REACTIVA PARA ROBOTS MÓVILES

TESIS

QUE PARA OPTAR POR EL GRADO DE:

MAESTRO EN CIENCIAS (COMPUTACIÓN)

PRESENTA:

ALEJANDRO BERMÚDEZ FIERRO

TUTOR: JESÚS SAVAGE CARMONA, FACULTAD DE INGENIERÍA

México, D.F. Noviembre 2014

ALEJANDRO BERMÚDEZ FIERRO

ROBÓTICA EVOLUTIVA: BÚSQUEDA DE COMPORTAMIENTOS DE NAVEGACIÓN REACTIVA PARA ROBOTS MÓVILES

RESUMEN

En el presente trabajo se describe una metodología basada en algoritmos genéticos y redes neuronales (*neuroevolución*) para crear comportamientos capaces de proporcionarle la habilidad a un robot de a) explorar un ambiente desconocido delimitado y b) navegar hacia un objetivo en un ambiente desconocido, con obstáculos en ambos casos. Se describen distintas arquitecturas de redes neuronales estructuralmente diferentes que son el mecanismo de control central del robot. Se utiliza el algoritmo genético ecléctico como el método de aprendizaje no supervisado para obtener los pesos de las redes neuronales. La función de aptitud para los individuos del algoritmo obtiene medidas de una simulación del robot con la red neuronal representada por la codificación del individuo actual como su mecanismo de control. Estas medidas indican la capacidad de cada individuo para llegar a una meta, evadir obstáculos y explorar el ambiente. Un análisis cuantitativo de las distintas arquitecturas evaluadas en un ambiente representativo de la dificultad de la tarea de navegación se presenta, con el fin de encontrar la arquitectura que muestre el mejor desempeño global, mitigando la tendencia adaptativa de los algoritmos genéticos al posicionar distintas metas en el ambiente (en el caso de la tarea de navegación). De las arquitecturas de redes neuronales evaluadas, una emerge como la más apta en ambas tareas por un margen significativo.

ABSTRACT

This thesis introduces a methodology based on genetic algorithms and neural networks (*neuroevolution*) for creating behaviors capable of providing a robot the ability of a) exploring a delimited unknown environment and b) navigating towards a goal in an unknown environment, with obstacles in both cases. Structurally different neural network architectures acting as the central control mechanism of the robot are presented. The eclectic genetic algorithm is used as the unsupervised learning method to obtain the weights of the neural networks. The fitness function for the individuals in the genetic algorithm is a simulation of the robot with the neural network represented by the bit string of the current individual as its control mechanism, measured by exploration, obstacle evasion and distance to a goal criteria. A qualitative analysis of the different architectures evaluated in an environment representative of the navigation task's difficulty is

presented, aimed at finding the neural net architecture that displays the best global performance, mitigating the adaptive tendency of genetic algorithms by positioning different goals in an environment (in the case of the navigation task). One of the neural network architectures emerges as the fittest in both tasks by a meaningful margin.

Doing science, particularly the synthesis of disparate ideas, is not as arcane as it is often made out to be. Discipline and taste play a vital role, but the activity is familiar to anyone who has made some effort to be creative.

— John Henry Holland

AGRADECIMIENTOS

Quiero agradecer al *Posgrado en Ciencia e Ingeniería de la Computación* por haberme proporcionado la oportunidad de estudiar la maestría en ciencias de la computación en un ambiente académico de la más alta calidad, el cual alimentó con ideas fundamentales la elaboración del presente trabajo. Los profesores de quienes tuve la fortuna de recibir clases y mis compañeros, sabrán apreciar sus valiosas aportaciones. También, quiero agradecer a mi tutor, el doctor *Jesús Savage Carmona* por su apoyo incondicional y su orientación durante estos años. Su excelente humor y temperamento lúdico hicieron del laboratorio de *Bio-robótica* un espacio ideal para realizar la investigación requerida por este trabajo, donde las preocupaciones de la vida cotidiana pasaban a un segundo plano. Finalmente, agradezco al doctor *Sven Wachsmuth* por haber fungido como co-tutor durante una estancia académica en el *Cognitive Interaction Technology - Center of Excellence* (*CITEC*), cuyas observaciones y recomendaciones culminaron en el procesamiento sensorial adecuado para la tarea de navegación hacia una meta. Se contó con el apoyo de una *beca para estudios de maestría* del CONACYT, con la modalidad de *Beca Mixta en el Extranjero* para esta estancia en Bielefeld, Alemania. Los comportamientos de navegación encontrados en esta tesis serán utilizados por el robot *Justina* en tareas para ayudar a adultos mayores y para la atención hospitalaria.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ACRÓNIMOS

ANN Artificial Neural Network

FANN Feed-forward Artificial Neural Network

SRN Simple Recurrent Network

RANN Recurrent Artificial Neural Network

FRANN First-order Recurrent Artificial Neural Network

SRANN Second-order Recurrent Artificial Neural Network

SCN Sequential Cascaded Network

ESCN Extended Sequential Cascaded Network

POMDP Partially Observable Markov Decision Process

LSTM Long Short-Term Memory

CEC Constant Error Carousel

NEAT Neuroevolution of Augmenting Topologies

ESP Enforced Subpopulations

EGA Eclectic Genetic Algorithm

NSGA Nondominatied Sorting Genetic Algorithm

NSGA-II Nondominatied Sorting Genetic Algorithm II

RMH Random Mutation Hill-Climber

CNN Convolution Neural Network

MPCNN Max-Pooling Convolution Neural Networks

PAES Pareto-Archived Evolution Strategy

SPEA Strength Pareto Evolutionary Algorithm

Parte I

LA NAVEGACIÓN EN LA ROBÓTICA

1

INTRODUCCIÓN

1.1 PRESENTACIÓN

Actualmente los robots móviles son campo de investigación en donde existe un enfoque considerable. Casi en cualquier universidad importante se tienen uno o más grupos dedicados al desarrollo de tecnología para robots móviles. Existen varios ejemplos de robots móviles, desde robots que simulan a un insecto volador hasta automóviles equipados con sensores y computadoras que navegan sin piloto en el desierto. Si estos robots se mueven de forma autónoma (sin que un ser humano decida la ruta que seguirá), entonces deben resolver un problema fundamental para el campo de la robótica móvil, la planificación de movimiento.

Concretamente, el problema a resolver es planear movimientos libres de colisiones para cuerpos complejos desde una posición inicial hasta una posición meta entre una colección de objetos estáticos. Extensiones de esta formulación toman en cuenta problemas adicionales causados por limitaciones sensitivas y móviles de robots reales como son la incertidumbre, la retroalimentación y restricciones diferenciales, que complican aun más el desarrollo de planeadores de movimiento automatizados.

Algunos algoritmos modernos han sido considerablemente exitosos en afrontar instancias difíciles del problema geométrico básico y se ha invertido mucho esfuerzo en extender sus capacidades para abordar instancias más complicadas. Estos algoritmos han tenido éxito abundante en áreas distintas a la robótica, como la animación, la realidad virtual y la biología computacional. Existen varios libros y compilaciones que cubren aplicaciones y técnicas modernas de planificación de movimiento [13, 31, 47, 77].

En la formulación de la planificación de movimiento, se trabaja únicamente sobre una representación computacional interna del mundo externo al robot; sin embargo, en la mayoría de los casos, un modelo completo del ambiente en donde interactúa el robot no está disponible y un control perfecto de las estructuras mecánicas, a través de las cuales actúa el robot, nunca es una suposición realista.

El sensado y la estimación son los medios por los cuales compensamos esta falta de información completa. Su rol es proporcionar in-

formación acerca del estado del ambiente y del estado del sistema robótico como una base para el control, las decisiones e interacciones con otros agentes en el ambiente, como los humanos. Una diferenciación importante en el sensado y la estimación es la *"propiocepción"* y *"exterocepción"*. La *propiocepción* busca recuperar el estado del robot mismo, mientras que la *exterocepción* busca recuperar el estado del mundo externo. En la práctica, muchos de los robots utilizan la propiocepción para estimar y controlar su propio estado físico. Por otro lado, recuperar el estado del mundo de los datos sensoriales es un problema más complejo.

En los trabajos iniciales de percepción computacional para la robótica se asumía que era posible recuperar un modelo de propósito general completo del ambiente, y usar ese modelo para tomar decisiones y actuar. Rápidamente se volvió aparente que tal enfoque no es realista. El sensado y la estimación pueden ser considerados como los procesos por los cuales se transforma una cantidad física a una representación computacional que puede ser utilizada en otros algoritmos.

Un sistema de navegación para robots móviles que planifique los movimientos en un ambiente donde inicialmente solo se conoce la meta, deberá descubrir el entorno en el que navega mediante el uso de sus sensores y la información de su posición, y decidir sus movimientos a partir de esta información. El sistema podrá decidir el siguiente movimiento solamente a partir de la información de sensado y posición actual, como lo hacen los sistemas reactivos, o podrá basar su decisión en la secuencia de sensado, posiciones y acciones generada a lo largo del tiempo mediante la interacción del robot móvil con el ambiente, pudiendo crear una representación interna o mapa del ambiente.

En el presente trabajo, se describe la construcción de un sistema para la navegación de ambientes desconocidos y no estructurados, utilizando técnicas de robótica evolutiva, teniendo a una red neuronal como la unidad central de control.

1.1.1 *Objetivos*

Esta investigación se enfoca en el desarrollo de un agente de navegación reactiva para robots móviles en espacios no estructurados y desconocidos.

Los objetivos generales que guiarán la construcción del sistema son los siguientes:

- Explorar el problema de la navegación para robots móviles autónomos.
- Comparar los métodos deliberativos y los reactivos.

- Explorar la utilidad de las redes neuronales recurrentes como sistemas de navegación para robots móviles.
- Contrastar distintas arquitecturas de redes neuronales recurrentes.
- Construir un método de aprendizaje basado en algoritmos genéticos y simulación que encuentre redes neuronales candidatas con capacidades satisfactorias.
- Preprocesar los datos sensoriales para disminuir el espacio de búsqueda para el algoritmo genético, manteniendo la información necesaria para que el robot actúe efectivamente en el ambiente.

1.2 MOTIVACIÓN DEL TRABAJO DE TESIS

En contraste con los robots autónomos móviles, los robots industriales realizan tareas definidas precisamente, utilizando métodos que están bien definidos a bajo nivel. Por ejemplo, un robot industrial usualmente es representado por un modelo dinámico, y la tarea que se espera que realice puede ser lograda por un método o procedimiento bien definido. A menudo, la tarea misma puede ser descrita por un modelo dinámico. Llegar a una descripción matemática de una estrategia de control óptima o casi óptima para la tarea se convierte en un problema de optimización matemática o a veces heurística de procedimientos bien definidos.

La situación es muy distinta para robots autónomos que deben interactuar dinámicamente con ambientes complejos. Mientras la tarea puede permanecer bien definida en el alto nivel, un algoritmo de solución efectivo usualmente no está bien definido. La mayoría de las tareas no triviales para robots autónomos no pueden ser descritas adecuadamente por modelos dinámicos.

Los sistemas de control para robots autónomos son a menudo programados directamente por los investigadores o diseñadores, y pueden llegar a ser sumamente complejos. Los investigadores deben anticipar que habilidades un robot dado necesitará, y después condensarlas en un programa o jerarquía de control. Muchos investigadores en el área de control de robots autónomos dependen de arquitecturas de control complejas para facilitar el diseño del control del robot en general [65, 66].

A medida que aumenta la complejidad de un ambiente y una tarea para un robot autónomo dado, la dificultad de diseñar un sistema de control adecuado a mano se convierte en un factor limitante en el grado de complejidad funcional que puede ser lograda. Una solución potencial a este problema es generar métodos que permitan a los robots aprender cómo realizar tareas complejas automáticamente.

Desarrollar métodos de aprendizaje de máquinas para uso en sistemas robóticos se ha convertido en un enfoque importante para la investigación contemporánea de robots autónomos. Algunos de estos métodos, como la robótica evolutiva, se concentran en el aprendizaje de sistemas de control completos.

Aprender control inteligente para agentes de robots autónomos es, de cierta forma, muy diferente a otras formas de aprendizaje de máquinas u optimización. En particular, a menudo no es posible generar un conjunto de datos de entrenamiento que pueda ser utilizado para entrenar controlador utilizando métodos como el algoritmo de *retropropagación*. Definir estados discretos para ambientes y robots autónomos complejos es también problemático y métodos tradicionales de diferencia temporal tal como *Q-learning* no son fácilmente aplicados a problemas de aprendizaje de control para robots autónomos en ambientes continuos y dinámicos.

La robótica evolutiva aborda el problema del control inteligente aplicando una evolución artificial basada en la población para evolucionar sistemas de control directamente. Este proceso evolutivo representa una forma de aprendizaje de máquinas que no necesariamente requiere conocimiento completo del ambiente, morfología del robot ni dinámica de la tarea.

Las redes neuronales son aptas para ser entrenadas con métodos evolutivos computacionales porque pueden ser representadas por un conjunto conciso de parámetros ajustables. Una amplia gama de redes neuronales puede ser utilizada. Las arquitecturas más comunes son las redes *feed-forward* y recurrentes.

Dado que el problema de la navegación en ambientes desconocidos para robots móviles es altamente dinámico y su solución no ha sido bien definida, se presta a ser trabajado con la metodología planteada por la robótica evolutiva. Evaluando distintas contribuciones al campo de la robótica evolutiva, el presente trabajo pretende generar una metodología para crear sistemas de control robustos para la tarea de la navegación en ambientes desconocidos.

1.3 ORGANIZACIÓN DE LA TESIS

1.3.1 *Parte 1 - La navegación en la robótica*

La primera parte presenta la navegación de un ambiente en la robótica y el problema de la representación de este ambiente dentro del robot para tomar decisiones de acción correspondientes a la dirección del movimiento con el fin de alcanzar un objetivo, y evitar los obstáculos entre el robot y dicho objetivo. Presenta el problema de diseñar sistemas de control para robots a mano y lo contrasta con los métodos de aprendizaje de máquinas, que representan una alternativa viable y con beneficios significativos. Dentro de estos métodos

se encuentra la robótica evolutiva, y esta parte termina describiendo brevemente cómo es que este método aborda el problema del control para robots móviles.

1.3.2 *Parte 2 - Antecedentes*

La parte de antecedentes busca proporcionar los antecedentes necesarios para justificar y crear el contexto de la investigación en el campo de la robótica evolutiva. Se da una revisión general del control en la robótica para ubicar a los sistemas de control generados con métodos evolutivos. Después, se presenta el mecanismo de control más utilizado en el área de la robótica evolutiva, las redes neuronales. Una breve introducción a sus propiedades matemáticas y sus virtudes es proporcionada, presentando al final la relevancia de la inclusión de la recurrencia en la arquitectura de la red. A continuación se describen los algoritmos genéticos que son el mecanismo de optimización central para generar los pesos de las redes neuronales, orientando los algoritmos genéticos a distintos objetivos.

1.3.3 *Parte 3 - Estado del Arte*

Con el fin de mostrar contribuciones relevantes a la robótica evolutiva, se presentan dos algoritmos que han inspirado a distintos grupos de investigación a realizar un número considerable de experimentos nuevos. El primero es el algoritmo NEAT, que incluye la evolución de la topología de la red neuronal a partir de un número mínimo de conexiones y neuronas, intentando aumentar el espacio de búsqueda solo si este aumento resulta también en un aumento en la aptitud del individuo. El segundo algoritmo trata la *búsqueda de novedad*, que presenta una manera radical de búsqueda (abandonar todo objetivo) con el fin de evitar los mínimos locales y explorar el espacio de los comportamientos, favoreciendo a la novedad.

1.3.4 *Parte 4 - La Búsqueda del Mejor Individuo*

Finalmente, se describen todos los componentes del sistema experimental a detalle, incluyendo la simulación, funciones de aptitud, algoritmo genético, construcción del controlador y el preprocesamiento sensorial. También se presentan los resultados experimentales obtenidos a partir de la ejecución del sistema y las conclusiones de todo el proceso. Aunque los resultados obtenidos son alentadores, aún hay varias áreas donde el sistema podría mejorar y se propone trabajo futuro para hacer más eficiente el funcionamiento del algoritmo y refinar los resultados que encuentra.

Parte II

ANTECEDENTES

2

ENFOQUES DE CONTROL EN LA ROBÓTICA

La *robótica situada* trabaja en el dominio de las máquinas con un cuerpo físico en ambientes dinámicamente cambiantes, que a la vez son complejos y difíciles de tratar. Estar *situado* se refiere entonces a existir en un ambiente complejo, y tener un comportamiento afectado en gran medida por este ambiente. En contraste, los robots que existen en ambientes estáticos y no cambiantes se consideran como no situados. Estos ambientes incluyen robots de ensamblado operando en ambientes complejos pero altamente estructurados, fijos, controlables y predecibles. La predictibilidad y estabilidad del ambiente tiene un impacto directo en la complejidad que el robot debe sobreponer para realizar una tarea determinada; por lo tanto los robots situados presentan un reto significativo para el diseñador.

El *control robótico*, también conocido como arquitectura computacional robótica, es el proceso de tomar información acerca del ambiente a través de los sensores del robot, procesándola como un elemento necesario para tomar decisiones para actuar y ejecutar acciones en el ambiente. La complejidad del ambiente tiene un impacto directo en la complejidad del control, que a su vez, está relacionada directamente con la tarea robótica. Mientras que hay un número infinito de maneras de programar al robot, existen cuatro clases fundamentales de métodos de control robótico.

2.1 DELIBERATIVO - PENSAR, DESPUÉS ACTUAR

En el control deliberativo, el robot usa toda la información sensorial disponible, y todo el conocimiento interno almacenado, para razonar acerca de cuales acciones realizar. El sistema de control está usualmente organizando utilizando una descomposición funcional de los procesos de toma de decisiones, los cuales están compuestos por un módulo de procesamiento sensorial, un módulo de modelado, un módulo de planificación, un módulo de evaluación de valores, y un módulo de ejecución [4]. Tal descomposición funcional permite que operaciones complejas se realicen, pero implica una independencia secuencial fuerte entre los modelos de toma de decisiones.

Razonar en sistemas deliberativos se presenta típicamente en la forma de planificación, requiriendo una búsqueda de posibles secuen-

cias de estado-acción y sus consecuencias. Es bien sabido que *la planificación*, un componente importante de la inteligencia artificial, es un proceso computacionalmente complejo. El proceso requiere que el robot realice una secuencia de pasos de sensado, deliberación y acción (por ejemplo, *combinar los datos sensoriales en un mapa del mundo, utilizar el planificador para encontrar una ruta en el mapa y enviar los pasos del plan a las llantas del robot*) [26, 46, 57]. El robot debe construir y después potencialmente evaluar todos los posibles planes hasta que encuentre el que le permita alcanzar su meta, resolver la tarea, o decidir una trayectoria a ser ejecutada. Shakey, uno de los primeros robots móviles que utilizaba Strips, un planificador general, es un ejemplo de este tipo de sistemas aplicado al problema de evitar obstáculos y navegar basado en información visual [61].

La planificación requiere de la existencia de una representación interna y simbólica del mundo, la cual permite al robot ver al futuro y predecir los resultados de acciones posibles en varios estados, con el fin de generar planes. Por lo tanto, el modelo interno se debe mantener preciso y actualizado. Cuando hay suficiente tiempo para generar un plan y el modelo del mundo es preciso, este enfoque permite al robot actuar de manera estratégica, seleccionando la mejor secuencia de acciones para una situación dada. Sin embargo, estar situado en un mundo ruidoso y dinámico provoca que esto sea imposible [9, 71]. Hoy en día casi ningún robot situado es puramente deliberativo. El desarrollo de arquitecturas alternativas fue provocado por la necesidad de acciones más rápidas pero apropiadas en respuesta a los requisitos de ambientes dinámicos y cambiantes del mundo real.

2.2 REACTIVO - NO PENSAR, REACCIONAR

El control reactivo es una técnica para acoplar directamente las entradas sensoriales y las salidas de los actuadores, donde típicamente no inerviene ningún razonamiento [10] para que el robot responda rápidamente a ambientes cambiantes y no estructurados [8]. El control reactivo está inspirado en la noción biológica de *estímulo-respuesta*; no requiere la adquisición ni mantenimiento de modelos del mundo, ya que no depende los procesos de razonamiento complejo del control deliberativo. En vez de la deliberación, métodos relacionados una cantidad mínima de computación, y ninguna representación interna ni conocimiento del mundo son usados típicamente. Los sistemas reactivos logran respuestas en tiempo real construyendo el controlador del robot con una colección de reglas de condición-acción preprogramadas, concurrentes y con un estado interno mínimo (por ejemplo, *si choca, detenerse; si se detiene, retroceder*) [2, 10]. Esto hace al control reactivo especialmente apto para ambientes dinámicos y no estructurados donde tener acceso a un modelo del mundo no es una opción realista. Además, la cantidad mínima de cómputo requerida por los

sistemas reactivos les permite responder inmediatamente a ambientes que cambian rápidamente.

El control reactivo es un método poderoso y efectivo que abunda en la naturaleza; los insectos, que superan a los vertebrados en número, son reactivos en gran medida. Sin embargo, las limitaciones de la reactividad pura incluyen la incapacidad de almacenar información, tener memoria o representaciones del mundo [7], y por lo tanto la incapacidad de aprender y mejorar con el tiempo. El control reactivo intercambia la complejidad del razonamiento por tiempos de reacción rápidos. El análisis formal ha demostrado que, para ambientes y tareas que pueden ser caracterizados a priori, los controladores reactivos pueden ser muy poderosos, y si son estructurados adecuadamente, capaces de rendimiento óptimo en clases particulares de problemas [3, 76]. En otros tipos de ambientes y tareas, donde los modelos internos, la memoria y el aprendizaje son requeridos, el control reactivo no es suficiente.

2.3 HÍBRIDO - PENSAR Y ACTUAR CONCURRENTEMENTE

El control híbrido pretende combinar los mejores aspectos del control reactivo y deliberativo: la respuesta en tiempo real de la reacción, y la racionalidad y optimalidad de la deliberación. Como resultado, los sistemas de control híbrido contienen dos componentes diferentes, las reglas de condición-acción reactivas y las deliberativas, que deben interactuar con el fin de producir salidas coherentes. Esto es complicado porque el componente reactivo lidia con las necesidades inmediatas del robot, como es moverse mientras se evitan obstáculos, y por lo tanto opera en una escala de tiempo muy rápida, utilizando datos sensoriales externos directamente. En contraste, el componente deliberativo utiliza representaciones del mundo internas altamente abstractas y simbólicas, y opera sobre ellas en una escala de tiempo más larga, por ejemplo, para realizar planificación global de rutas o planificar para toma de decisiones de alto nivel.

Mientras las salidas de los dos componentes no entren el conflicto, el sistema no requiere de más coordinación. Sin embargo, las dos partes del sistema deben interactuar si se espera que se beneficien una de la otra. Como consecuencia, el sistema reactivo debe sobreponerse al deliberativo si el mundo presenta un reto inesperado e inmediato. De forma análoga, el componente deliberativo debe informar al reactivo para guiar al robot hacia trayectorias y metas más eficientes y cercanas al óptimo. La interacción de los dos componentes del sistema requiere un componente intermedio, que reconcilie las diferentes representaciones utilizadas por los otros y cualquier conflicto entre sus salidas. La construcción de este componente intermedio es típicamente el mayor reto en el diseño de sistemas híbridos.

Los sistemas híbridos son conocidos también como *arquitecturas de tres capas*, por su estructura, la cual consiste en la capa reactiva (de ejecución), la capa intermedia (de coordinación), y la deliberativa (de organización/planificación), las cuales se organizan de acuerdo con el principio del aumento en precisión de control en las capas de bajo nivel con un decremento en inteligencia [73]. Una gran cantidad de investigación ha sido invertida para diseñar estos componentes y sus interacciones [6, 16, 19, 23, 24, 26, 52, 67]. Las arquitecturas de tres capas buscan obtener lo mejor del control reactivo en la forma de control dinámico, concurrente y responsivo, y lo mejor del control deliberativo, en la forma de acciones globalmente eficientes a lo largo de escalas de tiempo amplias. Sin embargo, hay problemas complejos relacionados con las interfaces entre estos componentes fundamentalmente diferentes y la forma en que su funcionalidad debe ser particionada aún no es entendida por completo [73].

2.4 CONTROL BASADO EN COMPORTAMIENTOS - PENSAR COMO ACTUAR

El control basado en comportamientos emplea un conjunto de módulos distribuidos que interactúan entre sí llamados *comportamientos*, que colectivamente logran el comportamiento deseado a nivel de sistema. Para un observador externo, los comportamientos son patrones de la actividad del robot emergiendo de interacciones entre el robot y su ambiente. Para un programador, los comportamientos son módulos de control que agrupan conjuntos de restricciones para lograr y mantener una meta [5, 54]. Cada comportamiento recibe entradas de los sensores y/o otros comportamientos en el sistema, y proporciona salidas a los actuadores del robot o a otros comportamientos. Por lo tanto, un controlador basado en comportamientos es una red estructurada de comportamientos que interactúan entre sí, sin representación centralizada del mundo. En vez de esta representación, los comportamientos individuales y las redes de comportamiento mantienen cualquier información de estado y modelos.

Sistemas bien diseñados basados en comportamientos se aprovechan de la dinámica de la interacción entre los comportamientos mismos, y entre los comportamientos y el ambiente. Se puede decir que la funcionalidad de los sistemas basados en comportamientos emerge de esas interacciones y por lo tanto no es una propiedad del robot ni del ambiente aislado, sino un resultado de la interacción entre ellos [5]. Al contrario del control reactivo, que utiliza colecciones de reglas reactivas con poca o ninguna representación del estado, el control basado en comportamientos utiliza colecciones de comportamientos, que no tienen tales restricciones; los comportamientos sí tienen estado y pueden ser utilizados para construir representaciones, permitiendo entonces el razonamiento, la planificación y el aprendizaje.

3

REDES NEURONALES ARTIFICIALES

El trabajo en *redes neuronales artificiales* (Artificial Neural Network (ANN)) ha sido motivado desde su creación por la observación de que el cerebro humano realiza cómputo de una manera completamente diferente a la computadora digital convencional. El cerebro es un sistema de procesamiento de información altamente *complejo*, *no lineal* y *paralelo*. Tiene la capacidad de organizar sus componentes estructurales, conocidos como neuronas, con el fin de realizar ciertos procesos computacionales (reconocimiento de patrones, percepción y control motor), muchas veces más rápido que la computadora digital más rápida hoy en día.

Al nacer, el cerebro tiene una estructura bien definida y la habilidad de construir sus propias reglas a través de la *experiencia*. Esta experiencia aumenta con el tiempo, teniendo el desarrollo más dramático en los primeros dos años desde el nacimiento.

Una neurona *en desarrollo* es sinónimo de un cerebro plástico: La *plasticidad* le permite al sistema nervioso en desarrollo adaptarse a su ambiente. Tal como la plasticidad parece ser esencial para el funcionamiento de las neuronas como unidades de procesamiento de información en el cerebro humano, también lo es para redes neuronales compuestas de neuronas artificiales. En su forma más general, una red neuronal es una máquina diseñada para modelar la forma en que el cerebro realiza una tarea en particular o función de interés; la red se implementa usualmente utilizando componentes electrónicos o se simula como software en una computadora digital. Para lograr buen rendimiento, las ANNs emplean una interconexión masiva de componentes simples llamados *neuronas* o *unidades de procesamiento*. A partir de lo anterior se puede ofrecer la siguiente definición de una red neuronal vista como una máquina adaptativa [36]:

RED NEURONAL ARTIFICIAL: procesador distribuido masivamente paralelo compuesto de unidades simples de procesamiento, las cuales tienen una capacidad natural de almacenar conocimiento experimental y de hacer este conocimiento disponible para ser usado. Es similar al cerebro en dos aspectos:

1. El conocimiento es adquirido por la red a partir de su ambiente a través de un proceso de aprendizaje.

2. Las fuerzas de las conexiones interneuronales, conocidas como pesos sinápticos, son utilizadas para almacenar el conocimiento adquirido.

El procedimiento utilizado para realizar este proceso de aprendizaje es llamado un *algoritmo de aprendizaje*, el cual tiene como fin modificar los pesos sinápticos de la red neuronal para obtener el objetivo de diseño deseado.

3.1 BENEFICIOS DE LAS REDES NEURONALES

Una ANN deriva su poder a través de su estructura distribuida masivamente paralela y su habilidad de aprender, y por lo tanto, generalizar. *La generalización* se refiere a la capacidad de la ANN de producir salidas razonables para entradas no encontradas durante el entrenamiento (aprendizaje). Estas dos capacidades de procesamiento de información hacen posible que las ANNs resuelvan problemas complejos. Sin embargo, es importante reconocer que estamos muy lejos de construir una arquitectura computacional que imite a un cerebro humano.

El uso de redes neuronales ofrece las siguientes propiedades y capacidades:

1. *No linealidad*. Una neurona artificial puede ser lineal (función polinomial de grado cero o uno) o no lineal. Una ANN, construida con una interconexión de neuronas no lineales, es en sí no lineal. Además, esta no linealidad es de un tipo especial en el sentido en que está *distribuida* en la red. La no linealidad es una propiedad muy importante, particularmente si el mecanismo físico responsable de la generación de la señal de entrada es inherentemente no lineal.

2. *Mapeo de entrada-salida*. Un paradigma popular del aprendizaje llamado *aprender con un maestro* o *aprendizaje supervisado* involucra la modificación de los pesos sinápticos de una red neuronal aplicando un conjunto de *muestras de entrenamiento* o *ejemplos de la tarea* etiquetados. Cada ejemplo consiste en una *señal de entrada* única y una *respuesta deseada* correspondiente. La ANN es presentada con un ejemplo elegido aleatoriamente del conjunto, y los pesos sinápticos (parámetros libres) de la ANN son modificados para minimizar la diferencia entre la respuesta deseada y la respuesta producida por la ANN a partir de la señal de entrada de acuerdo con un criterio estadístico. El entrenamiento de la red es repetido con muchos ejemplos del conjunto hasta que la red alcanza un estado estable donde no hay más cambios significativos en los pesos sinápticos. De esta manera, la red aprende de los ejemplos construyendo un mapeo de entrada-salida para el problema actual.

3. *Adaptabilidad*. Las ANNs tienen la capacidad inherente de *adaptar* sus pesos sinápticos a cambios en el ambiente que las rodea. En particular, una red neuronal entrenada para operar en ambientes específicos puede ser entrenada nuevamente fácilmente para adaptarse a cambios menores en las condiciones de operación del ambiente. Además, cuando están operando en un ambiente *no estacionario*, una red neuronal puede ser entrenada para cambiar sus pesos sinápticos en tiempo real. Como una regla general, se puede decir que entre más adaptativo sea un sistema, asegurando que en todo momento el sistema se mantenga estable, más robusto será su rendimiento cuando se requiera que el sistema opere en un ambiente no estacionario. Sin embargo, se debe enfatizar que la adaptabilidad no siempre lleva a que un sistema sea robusto, lo opuesto podría ocurrir [33].

4. *Respuesta evidencial*. En el contexto de la clasificación de patrones, una ANN puede ser diseñada para proporcionar información no sólo de cuál patrón en particular se selecciona, sino también de la *confianza* de la decisión tomada. Esta información puede ser usada para rechazar patrones ambiguos, si es que se presentan, mejorando el rendimiento de la clasificación de la red.

5. *Información contextual*. El conocimiento está representado por la estructura misma y el estado de activación de una ANN. Cada neurona en la red está afectada potencialmente por la actividad global de todas las demás neuronas en la red. Consecuentemente, la información contextual es tratada naturalmente por una ANN.

3.2 MODELO DE UNA NEURONA

Una *neurona* es una unidad de procesamiento de información que es fundamental para la operación de una ANN. El diagrama de bloques de la Figura 3.1 muestra el *modelo* de una neurona, el cual forma la base para el diseño de las ANNs. Podemos identificar tres elementos básicos del modelo neuronal:

1. Un conjunto de *sinapsis* o *vínculos*, en donde cada uno es caracterizado por un *peso* o una *fuerza de conexión*. Específicamente, una señal x_i en la entrada de la sinapsis j conectada a una neurona k es multiplicada por el peso w_{kj}. Es importante hacer una nota de la manera en que los subíndices del peso w_{kj} son escritos. El primer subíndice se refiere a la neurona en cuestión y el segundo subíndice se refiere a la entrada de esta sinapsis. A diferencia de una sinapsis en el cerebro, el peso sináptico de una neurona artificial puede estar en un rango que incluye tanto valores positivos como negativos.

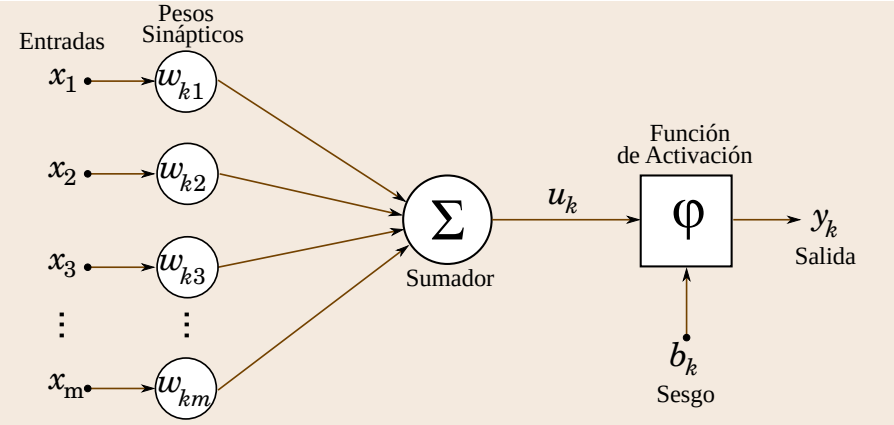


Figura 3.1: Modelo no lineal de una neurona.

2. Un *sumador* para sumar las señales de entrada, ponderadas por las sinapsis respectivas de la neurona; la operación descrita aquí, constituye un *combinador lineal*.

3. Una *función de activación* para limitar el rango de amplitud permisible de la señal de salida de una neurona a un valor finito. Típicamente, el rango de amplitud normalizado de la salida de una neurona es escrito como el intervalo unitario cerrado $[0, 1]$ o alternativamente $[-1, 1]$.

El modelo neuronal en la Figura 3.1 también incluye un *sesgo* (bias), denotado por b_k. El sesgo tiene el efecto de incrementar o decrementar el total de la entrada de la función de activación, dependiendo de si es positivo o negativo, respectivamente.

En términos matemáticos, podemos describir una neurona k escribiendo el siguiente par de ecuaciones:

$$u_k = \sum_{j=1}^{m} w_{kj} x_j \tag{3.1}$$

$$y_k = \varphi(u_k + b_k) \tag{3.2}$$

donde $x_1, x_2, \ldots, x_n$ son las señales de entrada; $w_{k1}, w_{k2}, \ldots, w_{km}$ son los pesos sinápticos de la neurona k; u_k es la *salida del combinador lineal* producido por las señales de entrada; b_k es el sesgo; $\varphi(\cdot)$ es la *función de activación*; y y_k es la señal de salida de la neurona. El uso del sesgo b_k tiene el efecto de aplicar una *transformación afín* a la salida u_k del combinador lineal del modelo en la Figura 3.1.

A menos de que especifique lo contrario, la función de activación que se adoptará en esta tesis para todas las neuronas será la *tangente*

hiperbólica [87], una función *sigmoide* cuya amplitud se encuentra acotada en el intervalo cerrado $[-1, 1]$, lo cual se puede apreciar en la Figura 3.2. La tangente hiperbólica está definida por:

$$\varphi(u) = \tanh(u) = \frac{e^{2u} - 1}{e^{2u} + 1} \tag{3.3}$$

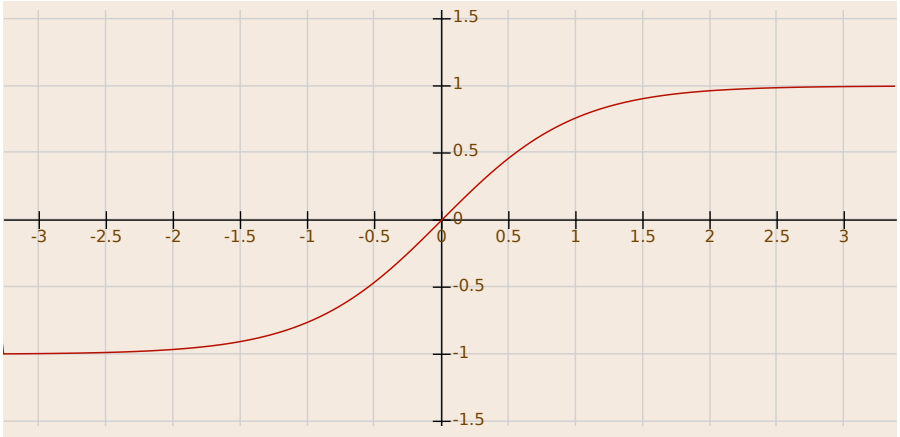


Figura 3.2: La tangente hiperbólica.

3.3 RED NEURONAL ACÍCLICA

En general, podemos identificar dos clases de arquitecturas de ANN: las *redes neuronales acíclicas* (Feed-forward Artificial Neural Network (FANN)) y las *redes neuronales recurrentes* (RANN). Si representamos a las FANNs como un grafo, podemos apreciar que siempre forman un *grafo acíclico dirigido*, lo cual les da su nombre. Para ambas clases, normalmente se crean grupos de neuronas llamados *capas*, en donde todas las neuronas de una capa tienen el mismo vector de entrada, como se puede apreciar en la Figura 3.3.

Una distinción importante entre las ANNs ocurre cuando se tienen una o más *capas ocultas*, cuyos nodos de cómputo son llamados *neuronas ocultas*. La función de las neuronas ocultas es mediar entre las entradas externas y la salida de la ANN de alguna manera que resulte útil. Al agregar una o más capas ocultas, se le permite a la red extraer estadísticas de alto orden. En un sentido más vago, la red adquiere una perspectiva *global* a pesar de su regla de conectividad local debido al conjunto extra de conexiones sinápticas y la dimensión extra de las interacciones neuronales [14]. La habilidad de las neuronas de extraer estadísticas de alto orden es particularmente valioso cuando el tamaño de la capa de entrada es grande.

Los nodos fuente en la capa de entrada proporcionan los elementos del patrón de activación (vector de entrada), el cual constituye las señales de entrada aplicadas a las neuronas (nodos de cómputo) en la segunda capa (la primer capa oculta). Las señales de salida de la

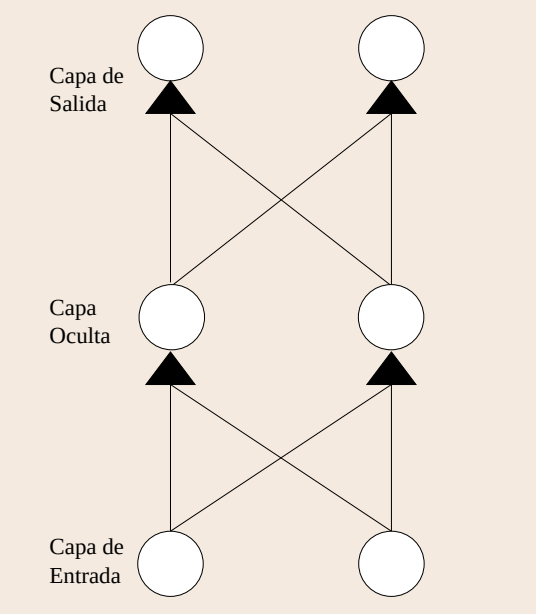


Figura 3.3: Red neuronal acíclica con una capa oculta.

segunda capa son utilizadas como entradas a la tercera capa, continuando de la misma manera para el resto de la red. Típicamente las neuronas en cada capa de la red tienen como sus entradas las señales de salida de la capa anterior solamente. El conjunto de señales de salida de las neuronas en la capa de salida (final) de la red constituyen la respuesta de la red ante un patrón de activación proporcionado por los nodos de entrada de la primera capa. El grafo en la Figura 3.3 ilustra la composición de una FANN multicapa para el caso de una sola capa oculta. Se dice que ANN en la Figura 3.3 es *completamente conexa*, ya que cada nodo en cada capa de la red está conectado a cada nodo en la capa frontal adyacente. Si algunos de los enlaces sinápticos no están presentes en la red, decimos que la red es *parcialmente conexa*.

3.4 REDES NEURONALES RECURRENTES

Una *red neuronal artificial recurrente* (RANN) se distingue de una FANN en que tiene por lo menos un *enlace sináptico retroalimentado*. La presencia de este tipo de enlaces tiene un impacto profundo en la capacidad de aprendizaje de la red y en su rendimiento. Los enlaces de retroalimentación utilizan conexiones particulares compuestas de elementos con *retardo unitario* (denotadas por z^{-1}), que resultan en un comportamiento no lineal y dinámico, asumiendo que la red neuronal contiene unidades no lineales. A diferencia de las FANNs, las RANNs pueden manejar sucesiones de entradas arbitrarias en vez de datos de entrada estáticos. Esto, combinado con la habilidad de memorizar eventos relevantes a lo largo del tiempo, hace que las redes neuronales recurrentes sean más poderosas en términos computacionales que

las FANNs. El conjunto de aplicaciones potenciales es enorme: cualquier tarea que requiera aprender como utilizar la memoria es una tarea potencial para las RANNs.

Dada una FANN multicapa como el elemento de construcción básico, la aplicación de retroalimentación global puede tomar una variedad de formas. Podemos tener retroalimentación de las neuronas de salida de la FANN multicapa a la capa de entrada. Otra forma posible de retroalimentación global es de las neuronas ocultas de la red a la capa de entrada. Cuando la FANN multicapa tiene dos o más capas ocultas, las formas de retroalimentación global se expanden aún más. El punto es que las RANN tienen un repertorio extenso de configuraciones para sus arquitecturas.

Básicamente, hay dos usos funcionales de las RANNs:

- Memorias asociativas
- Redes de mapeo entrada-salida variable en el tiempo

Esta tesis se concentra en las capacidades de mapeo de entrada-salida variable en el tiempo de las RANNs. Para este tipo de aplicación, una RANN responde *temporalmente* a una señal de entrada aplicada externamente. Además, la aplicación de retroalimentación le permite a las RANNs adquirir representaciones de estado, lo cual las hace aptas para para aplicaciones diversas como predicción y modelado no lineal, ecualización adaptativa de canales de comunicación, procesamiento de voz, control de plantas y diagnóstico de motores de automóvil. Esta habilidad de mapear sucesiones de entrada en los reales a salidas también en los reales, haciendo uso de su estado interno para incorporar contexto pasado en su procesamiento actual, las convierte en una herramienta de procesamiento de sucesiones impresionantemente general.

4

ALGORITMOS GENÉTICOS

Los algoritmos genéticos son métodos de optimización numérica inspirados en la *selección natural* y la *genética*. El método es general y capaz de ser aplicado a un rango extremadamente amplio de problemas. Están basados en el concepto de que la *evolución*, la cual, empezando a partir de algo que era poco más que una mezcla aleatoria de químicos, generó toda la biodiversidad que hoy en día vemos alrededor de nosotros, es un paradigma poderoso e increíble para resolver cualquier problema complejo. Concretamente, presentan un método robusto para búsqueda de soluciones de diferentes problemas de optimización prácticos. Un algoritmo genético típico está compuesto por los siguientes componentes:

1. Una población con soluciones candidatas a un problema.
2. Una manera de calcular qué tan bien o qué tan mal una solución generada a partir de un individuo de la población es.
3. Un método para combinar fragmentos de las mejores soluciones para generar otras nuevas y potencialmente mejores.
4. Un operador de mutación para evitar la pérdida permanente de diversidad dentro de las soluciones.

En una búsqueda numérica o problema de optimización, una lista (posiblemente de longitud infinita) de soluciones posibles se busca en orden para localizar la solución que mejor describe el problema actual. Un ejemplo puede ser intentar buscar los mejores valores para un conjunto de parámetros ajustables (o variables) que, cuando se incluyen en un modelo matemático, maximizan el impulso l generado por el ala de un aeroplano. Si hubiera sólo dos de estos parámetros ajustables, a y b, uno podría probar un número grande de combinaciones, calcular el impulso generado por cada combinación y producir una gráfica de la superficie con a, b y l graficados en los ejes x, y y z respectivamente. Tal gráfica sería una representación del espacio de búsqueda del problema. Para problemas más complejos, con más de dos variables, la situación se vuelve más difícil de visualizar. Sin embargo, el concepto de un espacio de búsqueda es válido siempre que alguna medida de distancia entre soluciones pueda ser definida

y a cada solución se le pueda asignar una medida de éxito, o *aptitud*, dentro del problema. Las soluciones con mejor rendimiento ocuparán las *crestas* (o *valles* si es que se está buscando un mínimo óptimo) dentro del espacio de soluciones. Tales espacios de soluciones pueden ser de topografía sorprendentemente compleja. Inclusive para problemas simples, pueden existir numerosas *crestas* de alturas variables, separados entre si por *valles* en múltiples escalas. Un claro ejemplo de esta situación puede ser apreciado en la Figura 4.1, correspondiente al espacio de búsqueda de la función de Hansen cuando $n = 2$:

$$f(2,x) = \sum_{i=0}^{4}(i+1)\cos(ix_0 + i + 1)\sum_{j=0}^{4}(j+1)\cos((j+2)x_1 + j + 1)$$

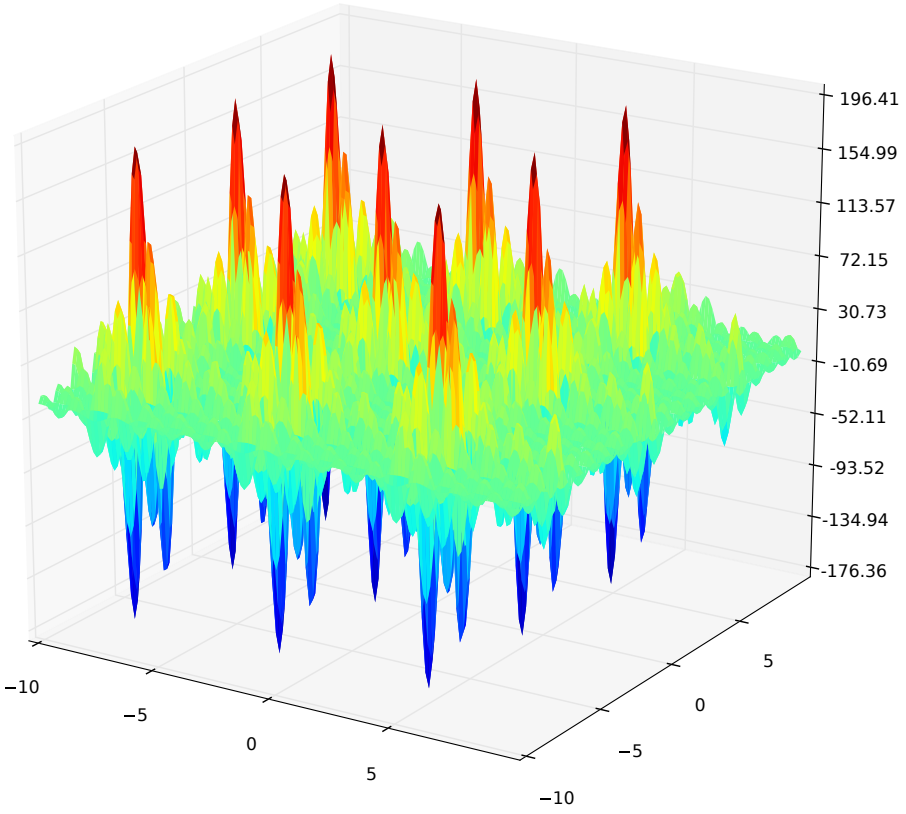


Figura 4.1: Función de Hansen para $n = 2$. Se pueden apreciar gran cantidad de crestas y valles en el espacio de búsqueda generado por la función, al igual que un número aun mayor de mínimos y máximos locales.

A la cresta de mayor altura se le conoce a menudo como el *máximo global* u *óptimo global*, y las crestas de menor altura como *máximo local* u *óptimo local*. Para la mayoría de los problemas de búsqueda la meta es la identificación precisa del óptimo global, pero esto puede no ser siempre cierto. En algunas situaciones, por ejemplo en control de tiempo real, la identificación de *cualquier* punto sobre un cierto valor

de aptitud puede ser aceptable. Para otros problemas, por ejemplo, en diseño de arquitecturas para edificios, la identificación de un conjunto grande de soluciones cercanas al óptimo, pero distantes entre sí en el espacio de soluciones (diseños), puede ser requerido.

En vez de comenzar a partir de un solo punto en el espacio de búsqueda, los algoritmos genéticos se inicializan con una *población* de puntos. Estos son a menudo aleatorios y serán esparcidos a lo largo del espacio de búsqueda. Un algoritmo típico usará después tres operadores, *selección*, *cruza* y *mutación* (elegidos en parte por su analogía con el proceso natural) para dirigir la población (a lo largo de una serie de pasos temporales o *generaciones*) hacia la convergencia en el óptimo global. Típicamente, estos puntos iniciales en el espacio de búsqueda se mantienen en la memoria como codificaciones binarias (o cadenas) de las variables verdaderas, aunque también es posible utilizar codificaciones con los números reales (base 10) directamente, o codificaciones que imiten de alguna forma la estructura natural del problema. Esta población inicial es después procesada por tres operadores principales.

La *selección* intenta aplicar presión sobre la población de una manera similar a la selección natural encontrada en los sistemas biológicos. Los individuos con rendimiento subóptimo son descartados e individuos más aptos tienen mayor probabilidad de propagar la información que contienen a la siguiente generación.

La *cruza* permite a las soluciones intercambiar información de una manera similar a la utilizada por un organismo natural en la reproducción sexual. Un método es escoger pares de individuos promovidos por el operador de selección, escoger aleatoriamente un solo punto dentro de las cadenas binarias e intercambiar toda la información a la derecha de este punto entre los dos individuos.

La *mutación* es utilizada para cambiar aleatoriamente el valor de uno solo de los bits dentro de la cadena binaria de un individuo. La mutación se utiliza en pequeñas cantidades muy a menudo.

Después de que la selección, cruza y mutación han sido aplicadas a la población inicial, una nueva población habrá sido generada y el contador generacional es incrementado en uno. Este proceso de selección, cruza y mutación es continuado hasta que un número fijo de generaciones ha sido alcanzado o algún tipo de criterio de convergencia se ha cumplido.

A primera vista no es para nada evidente que este proceso será capaz de encontrar un óptimo global en algún momento, y mucho menos formar la base de un algoritmo de búsqueda general y altamente efectivo. Sin embargo, la aplicación de esta técnica a numerosos problemas a lo largo de una amplia gama de campos ha mostrado que logra exactamente lo anterior. Posiblemente la prueba máxima de la utilidad de este enfoque descanse en el éxito demostrado de las formas de vida en el planeta Tierra.

Parte III

ESTADO DEL ARTE

5

ALGORITMO NEAT

En enfoques tradicionales de *neuroevolución*, una topología es elegida para las redes que serán evolucionadas antes de que el experimento comience. Usualmente, la topología de la red es una sola capa oculta de neuronas, con cada neurona oculta conectada a cada entrada de la red y cada neurona de salida. La evolución busca el espacio de los pesos de conexión de esta topología completamente conexa permitiendo que las redes con alto rendimiento se reproduzcan. El espacio de pesos es explorado a través de la cruza de vectores de pesos y la mutación de los pesos de la población. Entonces, la meta de la neuroevolución es optimizar los pesos de conexión que determinan la funcionalidad de la red. Sin embargo, los pesos de conexión no son el único aspecto de las redes neuronales que contribuyen a su comportamiento. La topología o *estructura*, de las redes neuronales también afecta su funcionalidad. Modificar la estructura de la red ha sido efectivo como parte del entrenamiento supervisado [25]. La pregunta básica es la siguiente: ¿Puede la evolución de la topología junto con los pesos de la red proporcionar una ventaja sobre evolucionar solo los pesos en una topología fija?

Un argumento persuasivo a favor de hacer evolucionar la topología y los pesos de la red fue presentado por Gruau et al. [34], argumentando que al hacer evolucionar la estructura se ahorra tiempo gastado por los humanos en decidir y experimentar entre diferentes topologías para un problema en particular. Aunque casi todos los sistemas de neuroevolución utilizan una capa oculta completamente conexa, decidir cuantos nodos ocultos son necesarios es un proceso de prueba y error. Gruau et al. soportaron su argumento haciendo evolucionar una topología y los pesos de una ANN que resolvía el problema de balanceo de poste más difícil en su momento. Sin embargo, los resultados posteriores demostraron que la evolución de la topología no era necesaria para resolver el problema. Un método de topología fija llamado Enforced Subpopulations (ESP) [28] fue capaz de resolver el mismo problema cinco veces más rápido con una solución simple, reiniciando el algoritmo con un número aleatorio de neuronas ocultas cada vez que el algoritmo dejaba de mejorar sus soluciones.

El algoritmo NEAT [83] intenta demostrar lo contrario: si se hace de forma correcta, hacer evolucionar una estructura junto con los pesos

de conexión puede mejorar significativamente el rendimiento de la neuroevolución. Está diseñado específicamente para aprovecharse de la estructura de una red como una manera de minimizar la dimensionalidad del espacio de búsqueda de los pesos de conexión. Si la estructura se hace evolucionar de tal forma que las topologías se minimicen y crezcan incrementalmente, ganancias considerables en la velocidad de aprendizaje son obtenidas. NEAT es un algoritmo único porque las estructuras se vuelven incrementalmente más complejas a medida que se van acercando más al óptimo en términos de la tarea, fortaleciendo la analogía entre los algoritmos genéticos y la evolución natural.

La evolución incremental de la estructura de la red presenta varios retos técnicos:

1. ¿Existe una representación genética que permita la cruza de topologías distintas de una forma significativa?
2. ¿Cómo puede una *innovación topológica* que necesita unas cuantas generaciones para ser optimizada ser protegida de tal forma que no desaparezca de la población de forma prematura?
3. ¿Cómo pueden ser minimizadas en tamaño las topologías *a lo largo de la evolución* sin la necesidad de afectar a la función de aptitud con una métrica de complejidad?

5.1 CODIFICACIÓN GENÉTICA

La codificación genética de NEAT está diseñada para permitir alinear a los genes con cierta correspondencia cuando dos genomas se cruzan durante la reproducción. Los genomas son representaciones lineales de la conectividad de la red (Figura 5.1). Cada genoma incluye una lista de *genes de conexión*, cada uno de los cuales se refiere a dos *genes de nodo*. Los genes de nodo proporcionan una lista de entradas, nodos ocultos y salidas que pueden ser conectados. Cada gen de conexión especifica el nodo de entrada, el nodo de salida, el peso de la conexión, si la conexión se expresa o no (bit de habilitación), y un *número de innovación*, que permite encontrar genes correspondientes.

La mutación en NEAT puede cambiar los pesos de conexión y las estructuras de la red. Los pesos de conexión mutan de la misma manera que lo hacen en los sistemas evolutivos tradicionalmente. Las mutaciones estructurales suceden de dos maneras (Figura 5.2). Cada mutación expande el tamaño del genoma agregando genes. En la mutación de *agregar conexión*, un solo gen de conexión nuevo con un peso aleatorio es agregado conectando dos nodos no conectados previamente. En la mutación de *agregar nodo*, una conexión existente es separada y el nuevo nodo es posicionado donde la conexión anterior solía estar. La conexión vieja es deshabilitada y dos conexiones nuevas son agregadas al genoma. La nueva conexión que lleva

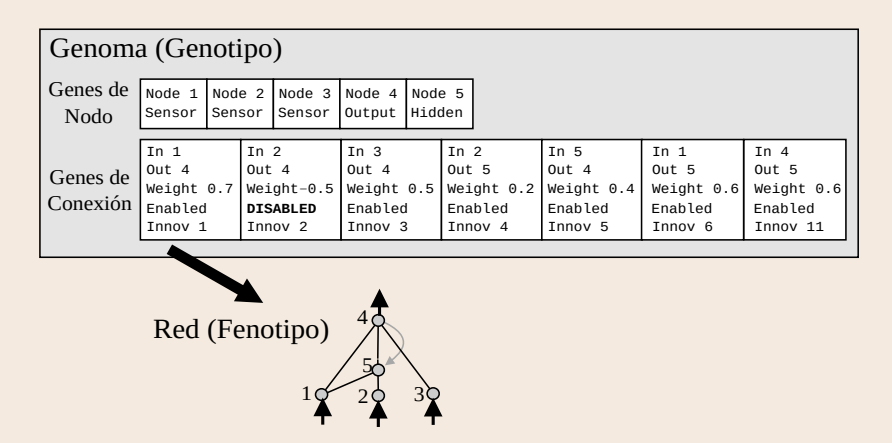


Figura 5.1: Un ejemplo del mapeo de genotipo a fenotipo. Hay tres nodos de entrada, uno oculto, un nodo de salida, y siete definiciones de conexión, una de las cuales es recurrente. El segundo gen es deshabilitado, de tal forma que la conexión que especifica (entre los nodos 2 y 4) no se expresa en el fenotipo.

al nuevo nodo recibe un peso de 1, y la nueva conexión que sale del nuevo nodo recibe el mismo peso que la conexión vieja. Este método de agregar nodos fue elegido con el fin de minimizar el efecto inicial de la mutación. La nueva no linealidad en la conexión cambia la función ligeramente, pero nuevos nodos pueden ser integrados inmediatamente en la red. De esta forma, gracias a la especiación, la red tendrá tiempo de optimizar y hacer uso de su nueva estructura.

A través de la mutación, los genomas en NEAT crecerán gradualmente. Genomas de tamaños variables resultarán, a veces con conexiones diferentes en las mismas posiciones. ¿Cómo puede el algoritmo cruzar genomas de diferente tamaño de forma razonable? La siguiente sección explica como NEAT aborda este problema.

5.2 RASTREAR GENES CON MARCAS HISTÓRICAS

Hay información no explotada en la evolución que nos dice exactamente cuáles genes se alinean con otros genes entre *cualquier* par de individuos en una población topológicamente diversa. Esta información es el origen histórico de cada gen. Dos genes con el mismo origen histórico deben representar la misma estructura (aunque posiblemente con pesos diferentes), ya que ambos están derivados del mismo gen ancestral en algún punto en el pasado. Por lo tanto, todo lo que el sistema necesita hacer para saber cómo alinear los genes es mantener un registro del origen histórico de cada gen en el sistema.

Rastrear el origen histórico requiere muy poca computación. Cada vez que un nuevo gen aparece (a través de la mutación estructural), un *número de innovación global* es incrementado y asignado a ese gen. Los números de innovación representan una cronología de la apari-

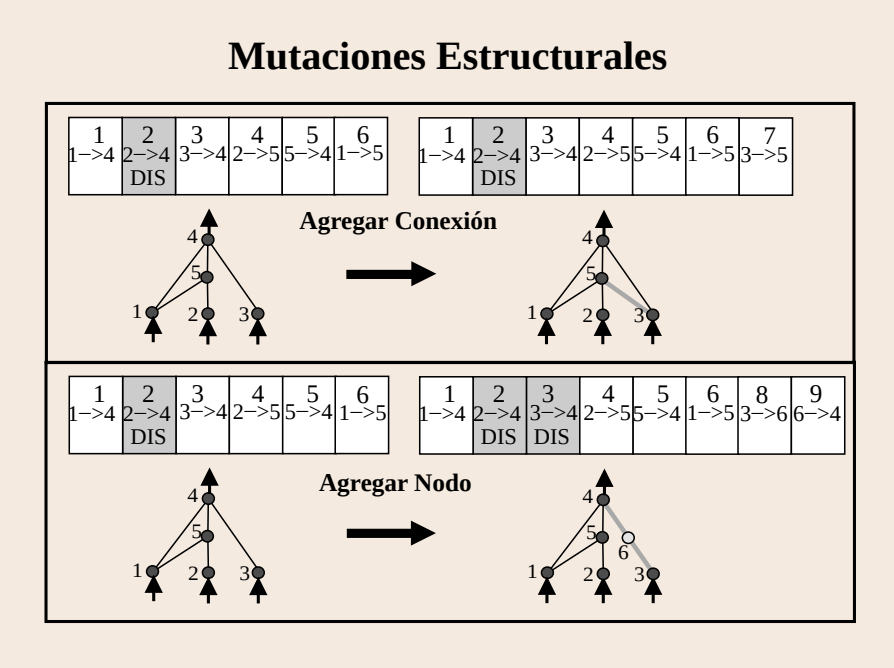


Figura 5.2: Los dos tipos de mutación estructural en NEAT. Ambos tipos, agregar conexión y agregar nodo, están ilustrados con los genes de conexión de una red sobre sus fenotipos correspondiente. El número superior de cada genoma es el *número de innovación* de ese gen. Los números de innovación son marcadores históricos que identifican el ancestro original de cada gen. A nuevos genes se les asignan nuevos números incrementalmente. Cuando se agrega un nuevo nodo, el gen de conexión que es separado se deshabilita y dos nuevos genes de conexión se agregan a al final del genoma. El nuevo nodo estará entre las dos nuevas conexiones. Un nuevo gen de nodo es agregado al genoma también.

ción de cada gen en el sistema. Por ejemplo, digamos las dos mutaciones en la Figura 5.2 ocurrieron una después de la otra en el sistema. Al nuevo gen de conexión creado en la primera mutación se le asigna el número 7 y a los dos nuevos genes de conexión agregados durante la nueva mutación del nodo se les asignan los números 8 y 9. En el futuro, cuando estos genes se reproduzcan, los hijos heredarán los mismos números de innovación en cada gen; los números de innovación nunca se cambian. Entonces, el origen histórico de cada gen en el sistema se conoce a través de la evolución.

Un problema posible es que la misma innovación estructural va a recibir números de innovación distintos en la misma generación si ocurre más de dos veces. Sin embargo, si se mantiene una lista de las innovaciones que ocurrieron en la generación actual, es posible asegurar que cuando la misma estructura emerge más de una vez a través de mutaciones independientes en la misma generación, cada mutación idéntica se le asigna el mismo número de innovación. Por lo tanto, no hay una explosión resultante de números de innovación.

Las marcaciones históricas le dan a NEAT una capacidad nueva muy poderosa. El sistema ahora sabe exactamente cómo embonar los genes (Figura 5.3). Cuando se realiza la cruza, los genes en los dos genomas con los mismos números de innovación se alinean. Estos genes se llaman genes *empatados*. Los genes que no empatan, son *disjuntos* o *exceso*, dependiendo de si ocurren dentro o fuera del rango de los números de innovación del otro padre. Representan la estructura que no está presente en el otro genoma. En la composición de la descendencia de dos individuos, los genes son elegidos aleatoriamente de cualquiera de los dos padres cuando empatan, mientras que todos los genes de exceso o disjuntos del padre más apto siempre se incluyen. De esta manera, las marcaciones históricas le permiten a NEAT realizar cruzas utilizando genomas almacenados en arreglos unidimensionales sin la necesidad de realizar análisis topológico costoso computacionalmente.

Agregando nuevos genes a la población y reproduciendo genomas que representan diferentes estructuras, el sistema puede formar una población de topologías diversas. Sin embargo, resulta que esta población por sí sola no puede mantener las innovaciones topológicas. Esto se debe a que las estructuras más pequeñas se optimizan más rápidamente que las estructuras grandes, y agregar nodos y conexiones usualmente decrementa la aptitud de la red inicialmente. Entonces, las estructuras que fueron aumentadas recientemente tienen pocas esperanzas de sobrevivir más de una generación aunque las innovaciones que representan puedan ser cruciales para resolver la tarea a la larga. La solución es proteger la innovación a través de la especiación de la población, lo cual se explica en la siguiente sección.

5.3 PROTEGIENDO LA INNOVACIÓN A TRAVÉS DE LA ESPECIACIÓN

Separar la población en especies le permite a los organismos competir dentro de sus propios nichos en vez de competir con toda la población. De esta manera, las innovaciones topológicas son protegidas en un nuevo nicho en donde tendrán tiempo de optimizar su estructura a través de la competencia dentro del nicho. La idea es dividir la población en especies tales que las topologías similares estén en la misma especie. Esta tarea parece ser un problema de agrupamiento de topologías. Sin embargo, nuevamente resulta que las marcaciones históricas ofrecen una solución eficiente.

El número de genes de exceso y disjuntos entre un par de genomas es una medida natural de su distancia de compatibilidad. Entre más disjuntos los dos genomas estén, menor será la historia evolutiva que compartan, y por lo tanto serán menos compatibles. Es por eso que podemos medir la distancia de compatibilidad δ de diferentes estructuras en NEAT con una simple combinación lineal del número de

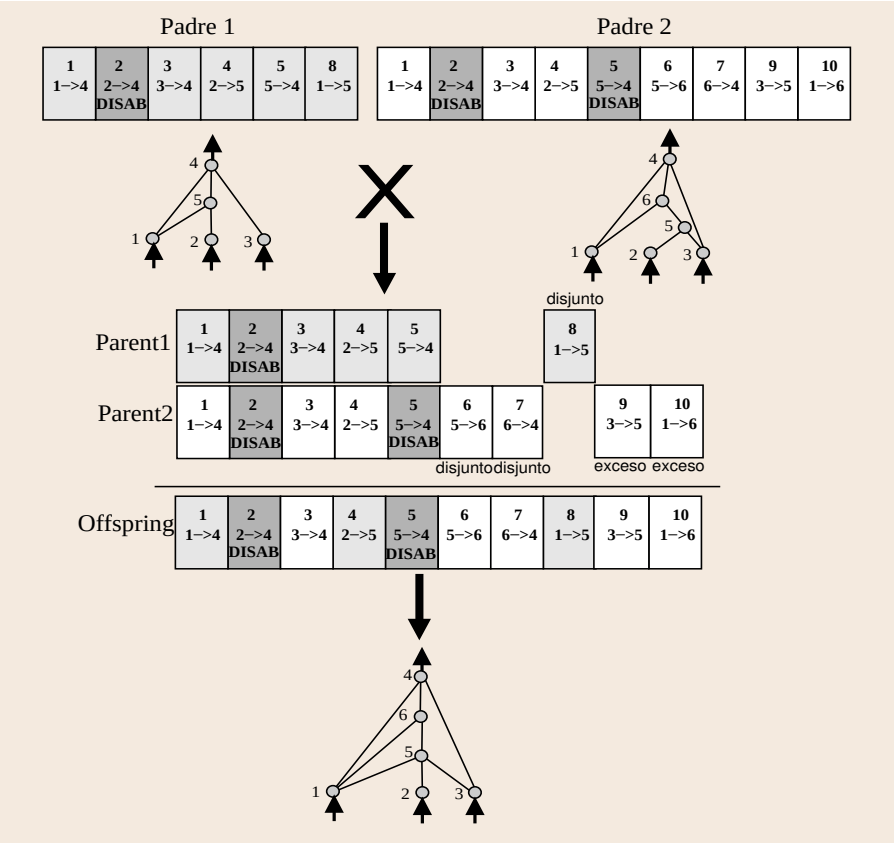


Figura 5.3: Empatamiento de genomas para topologías de red diferentes utilizando números de innovación. Aunque el padre 1 y el 2 se ven diferentes, sus números de innovación nos dicen cuáles genes empatan con cuáles. Sin necesidad de realizar análisis topológico, una nueva estructura que combina las partes que se traslapan de los dos padres al igual que sus partes distintas puede ser creada. Los genes que empatan son heredados aleatoriamente, todos los demás son heredados del padre más apto. En este caso, aptitudes iguales son asumidas, entonces los genes disjuntos y de exceso también son heredados aleatoriamente. Los genes deshabilitados pueden habilitarse nuevamente en generaciones futuras.

genes de exceso E y disjuntos D, al igual que diferencias promedio entre los pesos que empatan $\overline{W}$, incluyendo los genes deshabilitados:

$$\delta = \frac{c_1 E}{N} + \frac{c_2 D}{N} + c_3 \cdot \overline{W} \tag{5.1}$$

Los coeficientes c_1, c_2, y c_3 nos permiten ajustar la importancia de los tres factores, y el factor N, el número de genes en un genoma más grande, normaliza el tamaño del genoma (N puede ser establecido como 1 si los dos genomas son pequeños, por ejemplo, constan de menos de 20 nodos).

La medida de distancia δ nos permite dividir en especies utilizando un umbral de compatibilidad δ_t. Una lista ordenada de especies

es mantenida. En cada generación, los genomas son secuencialmente posicionados en especies. Cada especie existente es representada por un genoma aleatorio dentro de las especies de la *generación previa*. Un genoma g dado en la generación actual es posicionado en las primeras especies en donde g es compatible con el genoma representativo de esa especie. De esta manera, las especies no se traslapan. Si g no es compatible con alguna especie existente, una nueva especie se crea con g como su representante.

NEAT utiliza como método de reproducción *explicit fitness sharing* [27], donde los organismos de la misma especie deben compartir la aptitud de su nicho. Por lo tanto, aunque sus individuos tengan buen rendimiento, el número de individuos de una especie no puede volverse muy grande. Es por eso que es poco probable que una especie domine la población completa, lo cual es crucial para que la evolución con especies funcione. La función de aptitud ajustada f'_i para un organismo i es calculada de acuerdo con su distancia δ con cada organismo j en la población:

$$f'_i = \frac{f_i}{\sum_{j=1}^{n} sh\left(\delta(i,j)\right)} \tag{5.2}$$

la función sh se establece como 0 cuando la distancia $\delta(i,j)$ está sobre el umbral δ_t, de lo contrario, $\delta(i,j)$ se establece como 1 [80]. De esta manera $\sum_{j=1}^{n} sh\left(\delta(i,j)\right)$ se reduce al número de organismos en la misma especie que el organismo i. Esta reducción es natural porque las especies ya se encuentran agrupadas por compatibilidad utilizando el umbral δ_t. A cada especie se le asigna un número potencialmente diferente de descendencia en proporción a la suma de aptitudes ajustadas f'_i de los individuos que las componen. Después de esto, las especies se reproducen eliminando los individuos con peor rendimiento de la población. La población completa es después reemplazada por la descendencia de los individuos remanentes de cada especie.

El efecto resultante deseado de separar la población en especies es proteger la innovación topológica. La meta final del sistema, es realizar la búsqueda de una solución de la manera más eficiente posible. Esta meta es lograda a partir de la minimización de la dimensionalidad del espacio de búsqueda.

5.4 MINIMIZACIÓN DE LA DIMENSIONALIDAD

NEAT favorece la búsqueda de espacios de dimensionalidad mínima comenzando por una población *uniforme* de redes con cero neuronas ocultas (todas las entradas están conectadas a las salidas). Nuevas estructuras se introducen incrementalmente a medida que las mutaciones estructurales ocurren, y sólo las estructuras que son útiles a

través de la evaluación de aptitud sobreviven. En otras palabras, los aumentos estructurales que ocurren en NEAT siempre están justificados. Esta minimización de dimensionalidad le da a NEAT una ventaja de rendimiento comparada con los métodos de arquitectura fija.

6

BÚSQUEDA DE NOVEDAD

El concepto de *función objetivo*, que recompensa moverse más cerca a una meta, es universal en el aprendizaje de máquinas [56]. La intuición detrás de esta idea de la función objetivo, que es aceptada ampliamente, es que la mejor manera de mejorar el rendimiento es recompensar mejorar con respecto al objetivo. En el cómputo evolutivo, esta medida de rendimiento es llamada *función de aptitud*, la cual es una metáfora de la presión para adaptarse en la naturaleza. Aunque es el método estándar de búsqueda, sufre de la patología del *óptimo local* (callejones sin salida en el espacio de búsqueda con respecto a incrementar el valor de la función objetivo). De esta manera, las superficies inducidas por las funciones objetivo son a menudo *engañosas*. El problema es que la función objetivo no necesariamente recompensa el *camino adecuado* en el espacio de búsqueda que finalmente llevan al objetivo global.

En contraste con el énfasis de la optimización de un objetivo en el aprendizaje de máquinas y el cómputo evolutivo, los investigadores en el área de la *vida artificial* a menudo estudian sistemas sin objetivos explícitos, como la *evolución abierta* [12, 53]. Una meta ambiciosa de esta investigación es reproducir la innovación no acotada de la evolución natural. Un enfoque típico es crear un mundo artificial complejo en donde no existe un objetivo final más que la supervivencia y la replicación [1, 12]. Estos modelos asumen que la evolución biológica puede soportar una dinámica abierta que lleva a complejidad incremental y no acotada.

Sin embargo, una teoría que está ganando terreno aunque algo controversial en la biología es que lo que lleva a la complejidad en la evolución natural es una *fuerza pasiva* (no es llevada por la selección) [32]. De hecho, en esta teoría, el camino hacia la complejidad natural en la evolución puede ser *inhibido* por la presión de la selección. Si la presión de la selección es demasiado alta, entonces cualquier desviación de un comportamiento localmente óptimo será descartada por la selección. Entonces, en esta visión de la evolución, en vez de ser un producto lateral de la selección, tal vez la acumulación de la complejidad se explica mejor por una característica distinta de la evolución natural y de los sistemas evolutivos abiertos en general: Estos continuamente producen formas de vida nuevas [82].

Esta perspectiva inspira la idea principal detrás de la búsqueda de novedad [49]: En vez de modelar la evolución natural con la esperanza de que individuos novedosos sean continuamente descubiertos, es posible crear una dinámica abierta buscando la novedad *directamente*. La idea principal es buscar sin ningún objetivo más que el de buscar continuamente comportamientos novedosos en el espacio de búsqueda. Porque hay un número finito de comportamientos en el ambiente, algunos de los cuales deben ser más complejos que otros [32], la fuerza pasiva que lleva al incremento de la complejidad se *acelera* buscando novedad en el comportamiento de los individuos.

La conclusión es que al abstraer el proceso a través del cual la evolución natural descubre la novedad, es posible derivar un algoritmo de búsqueda que opera sin la presión de un objetivo final. La búsqueda de novedad es inmune a los problemas de óptimos locales inherentes en la optimización de objetivos porque ignora el objetivo completamente, sugiriendo la conclusión contraintuitiva de que ignorar el objetivo de esta manera puede a menudo *beneficiar* la búsqueda del objetivo. Aunque la búsqueda de novedad no es para nada una panacea, el punto más importante es que la búsqueda basada en un objetivo (estandar en los algoritmos evolutivos) claramente no siempre funciona bien. La implicación es que mientras parece natural culpar al algoritmo de búsqueda cuando falla en alcanzar un objetivo, el problema puede descansar en la búsqueda del objetivo mismo.

6.1 IMPLEMENTACIÓN

Los algoritmos genéticos son aptos para la búsqueda de novedad porque la población de genomas que es central a tales algoritmos cubre de forma natural una amplia gama de comportamientos. De hecho, rastrear la novedad requiere un cambio muy pequeño a cualquier algoritmo evolutivo además de reemplazar la función de aptitud con una *métrica de novedad*.

La métrica de novedad mide qué tan nuevo es el comportamiento de un individuo, creando una presión constante por hacer algo único. La idea clave es que en vez de recompensar el rendimiento basado en un objetivo, la búsqueda de novedad recompensa la divergencia de los comportamientos pasados. Por lo tanto, la novedad necesita ser *medida*. Hay muchas maneras potenciales para medir la novedad analizando y cuantificando comportamientos para caracterizar sus diferencias. Al igual que la función de aptitud, esta función debe ser ajustada al dominio del problema.

La novedad de un individuo recién generado es calculada con respecto a los *comportamientos* de un *historial* de individuos pasados cuyos comportamientos fueron altamente novedosos cuando fueron originados. La novedad es medida en relación a otros individuos en la evolución; es llevada por una dinámica coevolutiva. Además, si

el algoritmo evolutivo está en un estado estable (un individuo es reemplazado a la vez) entonces la población actual también puede sumplementar el historial representando los puntos visitados más recientemente.

El punto es caracterizar qué tan lejos están los nuevos individuos con respecto al resto de la población y de sus predecesores en el *espacio de comportamientos*. Una buena métrica es calcular qué tan *disperso* es un punto en el espacio de comportamientos de acuerdo con el historial. Las áreas con agrupamientos (clusters) densos de puntos visitados son menos novedosos y por lo tanto obtienen una medida peor.

Una métrica simple de la dispersión en un punto es la distancia promedio a los k vecinos más cercanos de ese punto, donde k es un parámetro fijo que es determinado experimentalmente. Si la distancia promedio a los vecinos de un punto es grande, entonces está en un área dispersa, será una región densa si la distancia promedio es pequeña. La dispersión ρ a un punto x está dada por:

$$\rho(x) = \frac{1}{x}\sum_{i=0}^{k} \mathrm{dist}\,(x, \mu_i) \tag{6.1}$$

donde μ_i es el i-ésimo vecino más cercano de x con respecto a la métrica de distancia $\mathtt{dist}$, que es una métrica dependiente del dominio de la diferencia de comportamiento entre dos individuos en el espacio de búsqueda. El cálculo de los vecinos más cercanos debe considerar individuos de la población actual y del historial permanente de individuos novedosos. Los candidatos de regiones más dispersas de este espacio de comportamientos reciben entonces calificaciones de novedad más altas. Es importante notar que este espacio de comportamientos no puede ser explorado de manera informada; no se conoce *a priori* como entrar a áreas de baja densidad, de la misma manera que no se conoce cómo construir una solución cercana al objetivo. Por lo tanto, moverse en el espacio de comportamientos novedosos requiere exploración.

Si la novedad es suficientemente alta en la ubicación de un nuevo individuo (sobre un umbral $\rho_{\mathtt{min}}$), entonces el individuo se agrega al historial permanente que caracteriza la distribución de soluciones previas en el espacio de comportamientos, similar a enfoques basados en historiales en la coevolución [39]. La generación actual más el archivo dan una muestra comprensiva de dónde ha estado la búsqueda y dónde está actualmente; de esta manera, al intentar maximizar la métrica de novedad, el enfoque de la búsqueda es simplemente hacia lo que es *nuevo*, sin un objetivo explícito. Aunque la búsqueda de novedad no busca un objetivo directamente, saber cuándo para el proceso es necesario. Esto sucede cuando un individuo de la población cumple con un criterio meta.

Parte IV

LA BÚSQUEDA DEL MEJOR INDIVIDUO

7

IMPLEMENTACIÓN EXPERIMENTAL

7.1 PROCEDIMIENTO GENERAL

La robótica evolutiva consiste en repetir ciclos de evaluación y selección con base en la aptitud de los controladores. Estos ciclos son análogos a las generaciones en la evolución natural. Durante cada ciclo o generación, se usan controladores individuales tomados de una población grande de controladores que intentan realizar una tarea durante un período de evaluación. Esto involucra instanciar cada controlador en un robot (real o simulado) y permitirle al robot interactuar con su ambiente (que puede incluir otros agentes) por un período de tiempo. Al terminar este período, el rendimiento de cada controlador robótico es evaluado con base en una función de aptitud (también llamada función objetivo). El algoritmo evolutivo usa información generada por la función de aptitud para seleccionar y propagar los individuos más aptos en la población actual de controladores a la población de la próxima generación. Durante la propagación de individuos entre poblaciones, los controladores son alterados utilizando operadores genéticos estocásticos tal como es la mutación y la cruza para producir los descendientes que conformarán la siguiente generación de controladores. Estos ciclos se repiten por muchas generaciones para entrenar poblaciones de controladores robóticos que realizan una tarea dada, y la evolución se termina cuando cierta condición se cumple.

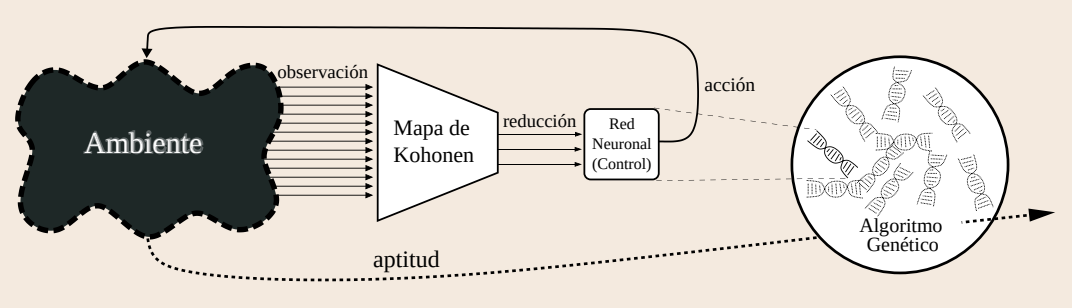


Figura 7.1: Proceso general para la búsqueda de individuos aptos. En esta figura se puede apreciar el rol del mapa de Kohonen para reducir la dimensionalidad de las observaciones del ambiente.

Como se ha mencionado anteriormente, el procedimiento experimental a seguir tiene como fin generar sistemas de control robótico para la navegación de forma automática. La robótica evolutiva describe los pasos generales para lograr este objetivo, pero muchos detalles deben ser proporcionados por el investigador. Con base en la descripción general anterior, podemos extraer los puntos principales del proceso que deben ser detallados:

- Simulación
- Función de aptitud/objetivo
- Algoritmo Genético
- Controlador

Las siguientes secciones se enfocan en dar una descripción de los detalles de implementación y la fundamentación teórica que soporta su elección.

7.2 SIMULACIÓN

El resultado final del proceso de la robótica evolutiva es tener un controlador que pueda ser instanciado en un robot físico. Sin embargo, durante el proceso de la búsqueda de dicho controlador, es difícil tener un robot físico disponible debido a que la evaluación de cada controlador generado por el algoritmo genético puede ser considerablemente tardada y varios controladores subóptimos tienen que ser evaluados primero para llegar a candidatos prometedores, lo cual presenta un riesgo de daño para la plataforma robótica que está siendo usada para el experimento. Otra dificultad de utilizar un robot físico para la búsqueda de controladores es la extracción de las métricas para la evaluación del comportamiento del robot; el investigador tendría que percibir el robot interactuando con un ambiente controlado a través de sensores de contacto o una cámara, y después procesar la señal para hacer sentido de ella. Muy posiblemente sea necesario reiniciar el estado del ambiente de evaluación a una configuración inicial, lo cual puede requerir de la intervención de un humano. A pesar de todas las desventajas técnicas que presenta esta forma de evaluación, existe una importante ventaja, y esta es que los controladores se adaptan directamente a su contenedor final y al ambiente en el que fueron entrenados; todas las limitaciones físicas, y errores de sensado y movimiento ya fueron considerados en la evaluación del controlador. No es necesario hacer un esfuerzo de transferencia como se requiere en la simulación, cuando el proceso evolutivo termina, el controlador final ya está situado en la realidad física del robot.

Una alternativa más conveniente al enfoque anterior es evaluar a los controladores en una simulación del robot interactuando en un

ambiente que representa una simplificación de la realidad, pero captura los aspectos físicos más importantes de ella. El objetivo principal de la simulación es que el comportamiento que es resultado del proceso evolutivo evaluado en la simulación debe suceder de manera idéntica a como lo hace una vez que el controlador responsable del comportamiento a sido instanciado en el robot físico. Esto significa que el robot simulado debe tener los mismos mecanismos de percepción y acción que el robot real. Sin embargo, es bien sabido que los programas que funcionan bien en simulación pueden no funcionar bien en el mundo real por diferencias en el sensado, actuación y las interacciones dinámicas entre el robot y su ambiente.

Este *espacio con la realidad* es aún más evidente en los enfoques adaptativos, como la robótica evolutiva, donde el sistema de control es refinado gradualmente a través de interacciones repetidas entre el robot y el ambiente. Por lo tanto, los robots evolucionarán para ajustarse a las especificidades de la simulación, que difieren del mundo real. Con el fin de evitar que el robot se adapte y dependa de estas especificidades, es posible agregar ruido independiente a los valores de los sensores proporcionados por el modelo y a la posición final del robot calculada por el simulador [58]. Esta técnica es adoptada en el presente trabajo y contemplada en el simulador utilizado para la evolución de los controladores.

Aunque estos problemas descartan cualquier simulación basada en mundos de bloques o cinemática pura, a lo largo de los últimos 15 años las técnicas de simulación han mejorado impresionantemente y han resultado en bibliotecas de software que modelan con suficiente precisión y velocidad propiedades dinámicas como la fricción, colisiones, masa, gravedad, inercia, etc. Estas herramientas de software permiten simular robots de morfologías variables y su ambiente más rápido que en tiempo real en una computadora. Hoy en día, estas simulaciones basadas en física son utilizadas por la mayoría de los investigadores en el área de la robótica evolutiva. El proceso general que llevan a cabo estos simuladores por lo normal ocurre de la siguiente manera:

1. Se crea un espacio vacío en donde los objetos interactuarán y se le asigna una fuerza de gravedad.

2. Los objetos y las interacciones entre ellos se agregan al espacio.

3. Se establece la diferencia de tiempo entre cada paso de simulación. Entre más pequeño sea el paso de simulación mayor calidad tendrá la simulación.

4. Los objetos del espacio se simulan avanzando en el tiempo en incrementos pequeños. Cada paso cambia el estado del espacio.

5. Después de cada paso de simulación, se obtiene nueva información del estado del espacio que puede ser utilizada para repre-

sentar el espacio o alterar alguna propiedad del espacio en el siguiente paso de simulación.

El aspecto de la rapidez es sumamente importante en el contexto de la robótica evolutiva, porque múltiples evaluaciones deben de ser realizadas para determinar la aptitud de los individuos. Para lograr esta rapidez, se optó por utilizar un simulador en dos dimensiones, aunque los simuladores en tercera dimensión pueden representar de manera más real las interacciones físicas de un robot. En el contexto de las tareas de navegación, es posible prescindir de este nivel de detalle ya que las interacciones entre los cuerpos pueden ser representados por una vista aérea del espacio en dos dimensiones.

Se consideraron distintos simuladores de dos dimensiones, considerando principalmente su velocidad, calidad de la simulación, calidad de la documentación, facilidad de desarrollo e integración, y lenguaje de programación. Al final, la biblioteca de simulador que mejor satisfacía estas características (en orden de importancia) fue *Chipmunk2D* [50]. Este simulador de física es usado en cientos de videojuegos en distintas plataformas, pero es particularmente popular en las aplicaciones para celulares debido a su portabilidad y velocidad. Está escrito en C, por lo que es fácil de acoplarlo con el resto de los componentes de la plataforma.

Aunque desarrollar una plataforma de investigación experimental capaz de soportar el entrenamiento evolutivo de robots autónomos sigue siendo una tarea no trivial, muchas de las preocupaciones iniciales y críticas con respecto a la transferencia de robots simulados a reales han sido atendidas. Existen suficientes ejemplos de plataformas de investigación para robótica evolutiva que han demostrado exitosamente la producción de controladores útiles en robots reales [38, 62, 20, 86]. También, hay varios ejemplos de evolución de controladores exitosos en simulación con la transferencia a robots reales [63, 64, 59, 29, 74].

La simulación está basada en las propiedades físicas del robot TurtleBot. Este robot es conveniente porque posee una forma circular, la cual puede ser representada fácilmente como un círculo en el simulador de física. Las especificaciones del robot real son las siguientes:

COMPUTADORA: Asus Netbook

MAX. VELOCIDAD LINEAL: ± 0.5 m/s

MAX. VELOCIDAD ANGULAR: ± 3.0 rad/s

ÁNGULO DE VISIBILIDAD DEL LÁSER: 180°

LECTURAS DEL LÁSER: 510

VELOCIDAD DEL LÁSER: 10hz

DIÁMETRO DEL ROBOT: 335mm

Figura 7.2: El robot TurtleBot, costruido por Willow Garage Inc.

El robot simulado tiene exactamente las mismas características que el robot real excepto por las siguientes:

MAX. VELOCIDAD LINEAL: ± 0.5 m/s

MAX. VELOCIDAD ANGULAR: ± 2.5 rad/s. Decrementado para aumentar la estabilidad del robot físico.

LECTURAS DEL LASER: 180. Decrementado para la velocidad de la simulación.

DIÁMETRO DEL ROBOT: 350mm Incrementado para contemplar las conexiones USB salientes de la computadora.

VELOCIDAD DE SIMULACIÓN: 30 hz

El simulador nos proporciona un mecanismo para detectar las interacciones de contacto entre distintos cuerpos o *colisiones*. Como se puede apreciar en la Figura 7.3, la longitud de ciertos rayos del láser del robot se ve acortada por sus interacciones con otros dos obstáculos que se encuentran unidos. La consulta de esta información es útil para la simulación del láser, ya que nos permite saber el punto de contacto del láser con cualquier otro obstáculo y por lo tanto, la longitud de cada rayo del láser. Esta longitud será alimentada después al control del robot para que este tenga información perceptual acerca del ambiente y genere la siguiente acción.

Otro uso importante de las colisiones en el presente trabajo es la determinación del tiempo de vida del individuo, o bien, el número de pasos que durará la simulación. Un individuo que interactúa de forma negativa con el ambiente, colisionando con obstáculos, representa

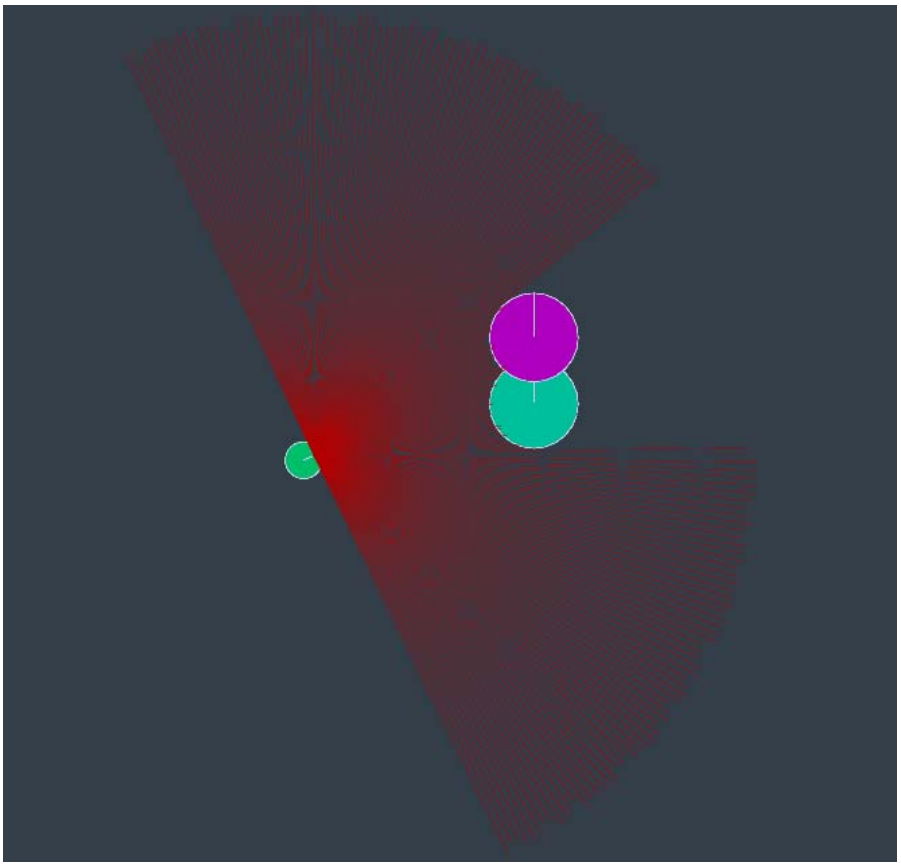


Figura 7.3: Robot simulado en un ambiente básico. Se puede apreciar la simulación del error en la lectura de los lasers y su interacción con el ambiente.

un individuo poco apto en términos de la tarea de navegación. Si este individuo no es deseable o útil en términos de la tarea de navegación, entonces no es necesario dedicarle más ciclos de simulación. El criterio para el decremento del tiempo de simulación es el siguiente: El individuo inicia con 90 segundos de simulación; por cada colisión del cuerpo con los obstáculos del ambiente, el individuo perderá un segundo de simulación. Cabe notar que la detección de colisiones y el decremento del tiempo de simulación ocurren en cada paso de simulación, no en cada segundo. Dado que cada segundo simulado son 30 pasos de simulación, un individuo que colisiona frecuentemente en el ambiente, pierde rápidamente su tiempo de simulación.

7.3 FUNCIONES DE APTITUD

La evolución exitosa de controladores para robots autónomos inteligentes depende, en gran medida, de la formulación de funciones de aptitud adecuadas que sean capaces de seleccionar los comportamientos exitosos sin especificar los detalles de implementación a bajo nivel para esos comportamientos; es la respuesta a la pregunta

de *¿Qué tiene que hacer el robot?*, sin especificar el *¿Cómo lo tiene que hacer el robot?*. La función de aptitud es el centro de una aplicación de cómputo evolutivo. Es responsable de determinar que soluciones dentro de una población son mejores para resolver un problema particular. En la práctica, al intentar evolucionar controladores para robots autónomos capaces de realizar tareas complejas, la función de aptitud es a menudo el factor limitante en la calidad de controladores que se pueden obtener. Este límite es usualmente manifestado por una meseta en la evaluación de aptitud en generaciones subsecuentes, e indica que la función de aptitud no es capaz de detectar diferencias de aptitud entre individuos en la población que está siendo evolucionada.

Para entender cómo es que el comportamiento de un individuo es más apto que el de otro, primero hay que definir precisamente lo que es un comportamiento. Dado un período de simulación de duración T y un conjunto de pasos discretos de tiempo $S = \{t_0, t_1, t_2, \ldots, t_T\}$ podemos definir:

SUCESIÓN PERCEPTUAL: Conjunto ordenado de vectores de información generados por los sensores del robot en un tiempo de la simulación. $P = \{p_0, p_1, p_2, \ldots, p_T\}$

SUCESIÓN DE ACCIONES: Conjunto ordenado de vectores de información que especifican una acción del robot en un tiempo de la simulación. $A = \{a_0, a_1, a_2, \ldots, a_T\}$

Un comportamiento es entonces la sucesión de acciones generada por el controlador robótico dada una sucesión perceptual. Esto no significa que un comportamiento es un mapeo del espacio de vectores perceptuales al espacio de vectores de acción, ya que el siguiente vector perceptual en la sucesión depende de la acción actual del robot. Un comportamiento se exhibe a lo largo de varios pasos de simulación y cobra sentido en términos de la tarea. Por lo tanto, la función de aptitud mide el rendimiento de un comportamiento en términos de la tarea. Esta métrica es independiente de cómo se producen las acciones a partir de la información sensible por el control interno del robot, es decir, esta métrica debería servir para medir el rendimiento de *cualquier* arquitectura de control. Entre mayor sea la independencia de la función de aptitud con respecto a la implementación, mayores serán las oportunidades de que el algoritmo genético encuentre comportamientos sumamente aptos por sí solo.

La acción del robot en cualquier momento toma sentido no sólo a partir de la información perceptual actual, sino de la sucesión de percepciones y acciones anteriores.

Como se mencionó anteriormente, dos tareas de navegación deben ser caracterizadas por la función objetivo:

EXPLORACIÓN: El robot debe explorar su ambiente, cubriendo la mayor cantidad de área posible. Esta meta puede ser medida por la cantidad de celdas visitadas en una cuadrícula.

NAVEGACIÓN A UNA META: Iniciando desde una posición inicial, el robot debe alcanzar una posición objetivo en su ambiente (*goal homming*).

7.3.1 *Exploración*

La tarea que se desea lograr es la de explorar un ambiente desconocido por el robot, intentando cubrir la mayor cantidad de área posible en este mundo, evitando tener colisiones con distintos obstáculos. El robot se encuentra en una búsqueda constante de espacio nuevo (no alcanzado físicamente por el robot anteriormente).

Las capacidades necesarias o útiles que pueden ser exhibidas por la arquitectura de control para que el robot logre realizar esta tarea son:

- Noción de dirección y orientación.
- Noción de posición actual.
- Noción de velocidad lineal y angular actual.
- Detección de obstáculos y noción de su posición.
- Reconocimiento de espacios navegables.
- Reconocimiento de áreas ya visitadas y desconocidas (Memoria a largo plazo).

La función de aptitud correspondiente a esta tarea es:

$$f(I) = -o - \frac{1}{1+c} \tag{7.1}$$

donde o es la cantidad de celdas visitadas por el centro del robot de una malla donde las celdas cuadradas tienen un tamaño del diámetro del robot (Figura 7.4), y c es el número de pasos de la simulación donde el robot está colisionando con algún obstáculo del ambiente al final de la simulación.

7.3.2 *Navegación hacia una meta*

Las capacidades necesarias o útiles que pueden ser exhibidas por la arquitectura de control para que el robot logre realizar esta tarea son:

- Noción de dirección y orientación.
- Noción de posición actual.
- Noción de velocidad lineal y angular actual.

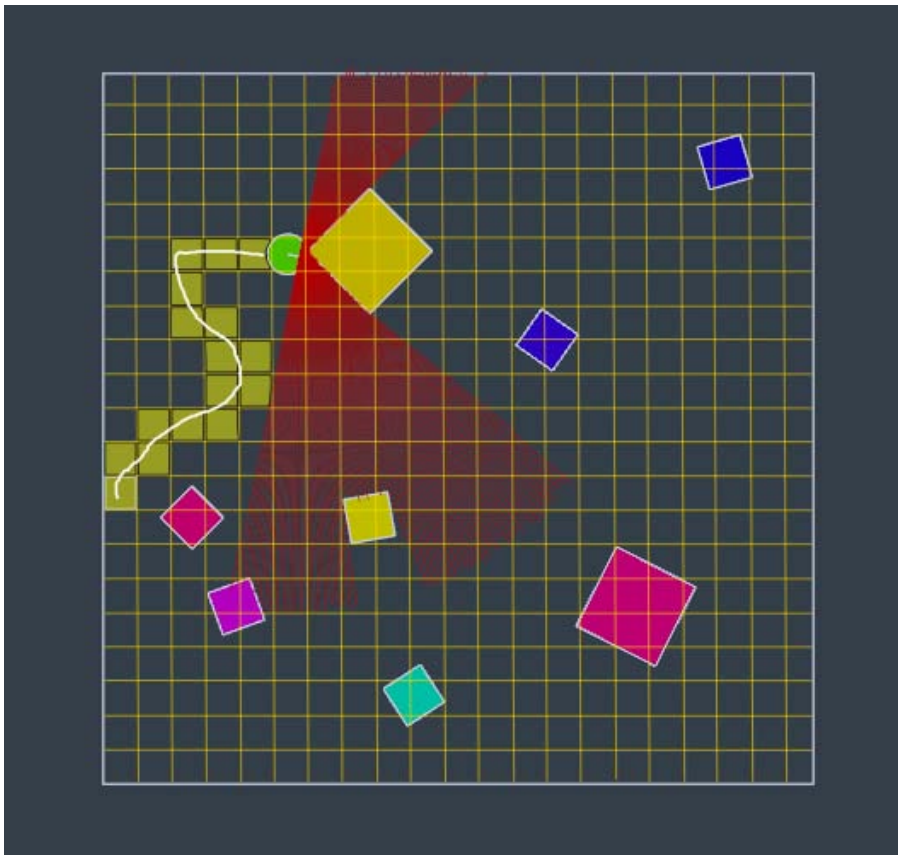


Figura 7.4: Malla del espacio ocupado por el robot a lo largo de la simulación. Cada celda nueva visitada por el robot a lo largo de la simulación aumenta su aptitud.

- Detección de obstáculos y noción de su posición.
- Alineamiento de la dirección de movimiento con la dirección de la meta.
- Capacidad de abandonar la meta para evadir un obstáculo.
- Negociación efectiva entre alcanzar la meta actual y evadir a un obstáculo.

La función de aptitud correspondiente a esta tarea es:

$$f(I) = d - rg \tag{7.2}$$

donde d es la distancia a la meta actual, r es una constante que representa una recompensa por alcanzar una meta y g es el número de metas alcanzadas.

7.3.3 *Tipos de funciones de aptitud*

Como se describe en [60], las funciones de aptitud pueden ser clasificadas por el grado de conocimiento *a priori* que reflejan. La jus-

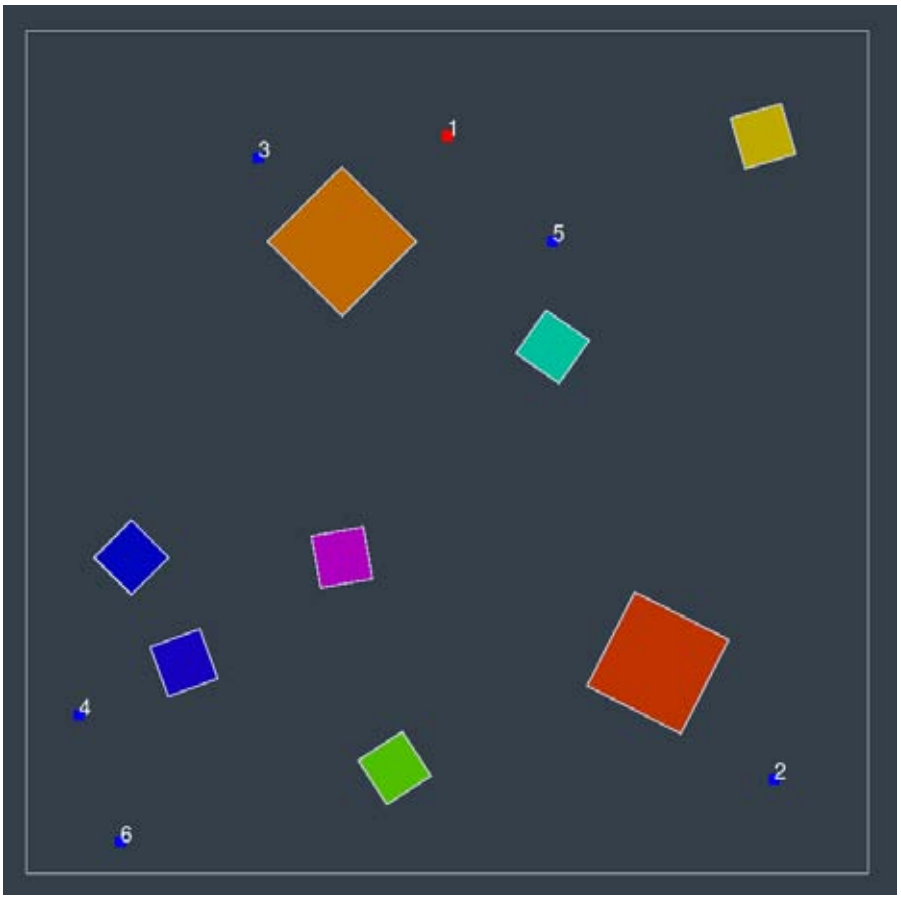


Figura 7.5: Ambiente con 6 metas. El robot tendrá que navegar en orden de ascendencia hacia cada meta antes de que termine el tiempo que tiene disponible, intentando minimizar el número de colisiones que tiene con los obstáculos presentes en el ambiente. Sólo una meta puede *emitir* su señal al robot en cualquier momento de la simulación. Cuando el robot alcanza una meta, la siguiente meta en la sucesión se convierte en el objetivo actual del robot y produce la nueva sucesión de entradas de navegación (con excepción de los lasers) del control del robot, hasta el momento que es alcanzada y cambia nuevamente.

tificación para utilizar este conocimiento *a priori* como base para la clasificación y organización de la investigación es que refleja el nivel de aprendizaje novedoso que ha sido logrado. Hay claramente otros medios por los cuales los diseñadores introducen su propio conocimiento *a priori* de la solución de una tarea en el diseño de los sistemas experimentales pensados para estudiar la evolución y el aprendizaje en robots autónomos. Estos incluyen la selección de sensores y actuadores apropiados, diseño de ambientes de entrenamiento, y la elección de condiciones iniciales. Aunque estas otras formas de conocimiento *a priori* introducido son también importantes, es la función de aptitud que contiene el conocimiento de la solución de una tarea de manera más explícita.

Los tipos más contrastantes de funciones de aptitud son las *funciones de aptitud con datos de entrenamiento* y las *funciones de aptitud acumulativas*, que incorporan la mayor y menor cantidad de conocimiento *a priori* respectivamente.

Las funciones de aptitud con datos de entrenamiento son utilizadas en métodos de entrenamiento basados en descenso de gradiente, tal como la retropropagación de error para el entrenamiento de redes neuronales, y varios métodos numéricos y de ajuste de curvas. En estas funciones la aptitud se maximiza siempre que el sistema en cuestión produce un error de salida mínimo cuando se le presenta un conjunto dado de entradas asociado con un conjunto conocido de salidas óptimo. Para un problema dado, un conjunto de datos de entrenamiento debe incluir suficientes ejemplos de tal forma que el sistema de aprendizaje pueda extrapolar una ley de control generalizable y válida. Por lo tanto, un conjunto de datos de entrenamiento ideal contiene conocimiento de *todas* las características del problema de control en cuestión. El uso principal de estas funciones de aptitud en aprendizaje de control autónomo es en el área de aprendizaje mimético, en donde un sistema robótico aprende a imitar el comportamiento generado por un humano u otro entrenador. Consecuentemente, el comportamiento que el robot exhibirá está completamente determinado y básicamente buscará duplicar un conjunto conocido *a priori* de entradas y salidas del sistema.

Por el contrario, las funciones de aptitud acumulativas evalúan solamente el éxito o fracaso al intentar completar una tarea, ignorando completamente cómo es que la tarea fue completada. Este tipo de función reduce la influencia del sesgo que el humano introduce al sistema evolutivo acumulando la evaluación del beneficio o déficit de todos los comportamientos del robot en un solo término de evaluación. Estas funciones favorecen la evolución de comportamientos inteligentes y novedosos, y evitan caer en la optimización de estrategias diseñadas por humanos para el controlador. Las funciones de aptitud planteadas en el presente trabajo buscan ser de este tipo.

7.4 ALGORITMO GENÉTICO

El algoritmo genético es posiblemente el componente más importante de la robótica evolutiva. Es esencial para la generación evolutiva de individuos aptos en el dominio de la tarea, ya que se encarga de realizar la búsqueda de individuos prometedores en el espacio de controladores posibles guiado por un principio de optimalidad. Concretamente, esta búsqueda se realiza evaluando distintas configuraciones de los pesos de una arquitectura de red neuronal (elegida *a priori*) que permanece *fija* a lo largo de toda la ejecución del algoritmo genético. En realidad, el algoritmo genético estará buscando en el espacio de los pesos de una red neuronal.

Existen distintos algoritmos genéticos con diferencias estructurales importantes que radican principalmente en el uso de los operadores genéticos y de la codificación de los individuos. Afortunadamente, un estudio previo se ha encargado de comparar el rendimiento de estos algoritmos en el espacio de todos los problemas numéricos [44]. El algoritmo que presentó el mejor rendimiento de acuerdo con un muestreo estadístico de posibles funciones fue el *Algoritmo Ecléctico (Apéndice D)* [45]. El rendimiento de un algoritmo genético es medido generalmente por la calidad de las soluciones encontradas y el número de generaciones necesarias para encontrarlas. Es por esto que la búsqueda de controladores se realizará con este algoritmo.

Experimentos recientes apuntan a que las técnicas que presentan algún mecanismo para preservar la diversidad de la población a nivel del comportamiento a menudo encuentran soluciones de mucho mejor calidad que el algoritmo genético clásico. De hecho, existen enfoques en donde los objetivos de la tarea se abandonan por completo y la única métrica de aptitud que existe es la novedad de los comportamientos, de tal manera que si el comportamiento que se presenta es más novedoso, entonces es más apto (Capítulo 6). El algoritmo ecléctico presenta un mecanismo que implícitamente intenta mantener la diversidad fenotípica de la población a través de la selección de individuos con aptitudes muy distintas entre sí.

Se ha demostrado previamente [11] que la consideración conjunta de distintos objetivos en la evolución de los individuos proporcionan mejores resultados en ciertas instancias que realizar una combinación lineal (o de monomios) de las distintas consideraciones que se buscan expresar a través de minimización (o maximización) de métricas específicas. Esto parece ser cierto para la optimización de objetivos múltiples en general.

¿Cómo es que una tarea robótica puede ser expresada como un problema de optimización múltiple? ¿Cuál es la utilidad de expresarlo de esta manera? Tomemos como ejemplo la tarea de la navegación hacia un objetivo. En principio, queremos que el robot navegue en dirección al objetivo y lo alcance. Al mismo tiempo, podríamos también querer que este robot logre realizar movimientos minimizando el número de colisiones que tiene con los obstáculos en su ambiente. Lo que sucede en términos del algoritmo de optimización es que se busca una solución que al mismo tiempo maximice la efectividad del robot para alcanzar objetivos y minimice las colisiones del robot del ambiente. El enfoque tradicional intentaba relacionar los dos objetivos dentro de alguna función matemática. Esto presenta varios problemas; a menudo las unidades en las que se expresan los distintos objetivos no tienen una relación directa y por lo tanto no son directamente comparables. Puede suceder también que las unidades se encuentren en escalas muy distintas, es posible que un objetivo se mida en miles y otro en milésimas de alguna unidad. Tal vez el problema más serio es

que el implementador debe darle una ponderación a cada uno de los objetivos, la cual expresará preferencia por alguno de ellos. En el caso del presente trabajo, uno tendría que responder a la pregunta: ¿En qué medida es preferible que el robot alcance su objetivo comparado con la evasión de colisiones, o viceversa? En una implementación multiobjetivo, los distintos objetivos se consideran de forma aislada con respecto al resto, es decir, los individuos son comparados en cada uno de los objetivos para determinar su ordenamiento.

Lamentablemente, el algoritmo ecléctico no está diseñado para trabajar con objetivos múltiples, por lo que otro algoritmo debe ser considerado. El algoritmo multiobjetivo mejor conocido, gracias a su rendimiento y relativa facilidad de implementación, es Nondominatied Sorting Genetic Algorithm II (NSGA-II) [17]. Cabe resaltar que, tanto NSGA-II como el algoritmo ecléctico se distinguen por tener un mecanismo de preservación de diversidad como un elemento central del proceso de búsqueda. Ambos reportan excelente rendimiento y su programación resulta sencilla al compararse con otras opciones. La pregunta que uno podría hacerse ahora es, ¿Pueden estos dos algoritmos combinarse? Pues bien, si uno analiza la estructura de NSGA-II podrá entonces darse cuenta de que este algoritmo está dedicado casi por completo a crear un ordenamiento de individuos con múltiples objetivos basado en la dominancia de Pareto y en la diversidad con respecto a sus vecinos, es decir, convierte un problema de múltiples objetivos en un problema con un solo objetivo, el de lograr un ordenamiento óptimo. Después de realizar este ordenamiento, el algoritmo prosigue como un algoritmo genético básico con selección de torneo. Es en este momento donde podemos hacer la sustitución de estos operadores tradicionales por los del algoritmo ecléctico y tener la característica particular de una élite del tamaño de la población. Esto resulta en la ejecución del algoritmo ecléctico teniendo como función objetivo directa el ordenamiento de NSGA-II. La combinación de estos dos algoritmos es utilizada para la evolución de los controladores en el presente trabajo.

7.5 CONTROLADOR NEURONAL

7.5.1 *Preprocesamiento sensorial*

La selección cuidadosa y el preprocesamiento correcto de la información obtenida del ambiente es esencial para que el controlador realice una deliberación correcta. Esta información deberá contener los elementos necesarios para componer una acción sensata en términos de los objetivos de evasión de obstáculos, exploración y navegación hacia una meta.

Como se mencionó anteriormente, tanto el sensor simulado como el real son capaces de entregar por lo menos 180 lecturas de distancia,

las cuales proporcionarán la información de entrada al controlador para poder deliberar acerca del ambiente en la vecindad del robot, lo cual es esencial para la percepción y evasión de los obstáculos. Sin embargo, podemos asumir que estas lecturas tienen un nivel bastante alto de redundancia y que la información relevante para las decisiones que debe tomar el controlador es sólo una fracción del total. Además, el espacio de búsqueda del algoritmo genético crece considerablemente por cada nuevo valor de entrada en la capa de entrada de al red neuronal. Como se describe en [84], se emplea un método de cuantificación vectorial llamado *red neuronal de Kohonen* [41] para reducir la dimensionalidad de esta información, y a través de esta reducción, disminuir también los pesos que existen entre la capa de entrada y la siguiente capa en la red neuronal de control. Esta reducción de los pesos hace que el espacio de búsqueda del algoritmo genético se reduzca drásticamente, facilitando la obtención de comportamientos altamente aptos para todo el proceso, y también hace que la información sensorial utilizada para el rastreo del objetivo tenga una cantidad de entradas en la red neuronal igual a las utilizadas para proporcionar la información de los sensores de distancia. La red neuronal de Kohonen utilizada en el presente trabajo consiste en un enrejado l-dimensional con m nodos en cada dimensión, donde $l = 3$ y $m = 10$.

El proceso, ilustrado en la Figura 7.6, consiste en tomar un vector de lecturas de distancia como entrada a la red y encontrar la unidad del cubo que más se active, la unidad *ganadora*. La dirección de esta unidad ganadora es la salida de la red neuronal de Kohonen. Esta metodología hace una reducción de $\mathbb{R}^{20}$ a $\mathbb{N}^3$. La ventaja de esta estrategia es que la estructura topológica original del espacio de entrada está bien preservada en el espacio de salida, lo cual asegura la generalidad de la transformación porque el vector de salida es una función del vector de entrada que varía suavemente. Aunque las lecturas reales exhiben su distribución estocástica, la red de Kohonen es capaz de agrupar esta información ruidosa de distancia en un mapa que preserva la topología y es auto-organizado.

El conjunto de vectores de entrenamiento se obtiene directamente de las lecturas de láser del simulador. El robot simulado se maneja alrededor del ambiente, acumulando los distintos vectores de lecturas percibidos a través de su sensor simulado. Para cada vector de lecturas, se realiza una reducción del número total de lecturas de 180 a 20. Esta reducción funciona simplemente tomando la lectura más pequeña de cada uno de los segmentos angulares de 9°, ya que es la más cercana al robot. El conjunto de vectores de entrenamiento inicial consta de 8,400 vectores reducidos; sin embargo, existe mucha redundancia en la información que contiene; varias lecturas del láser se encuentran en el mismo espacio sensorial y pueden ocasionar que el agrupamiento no se realice adecuadamente en el mapa.

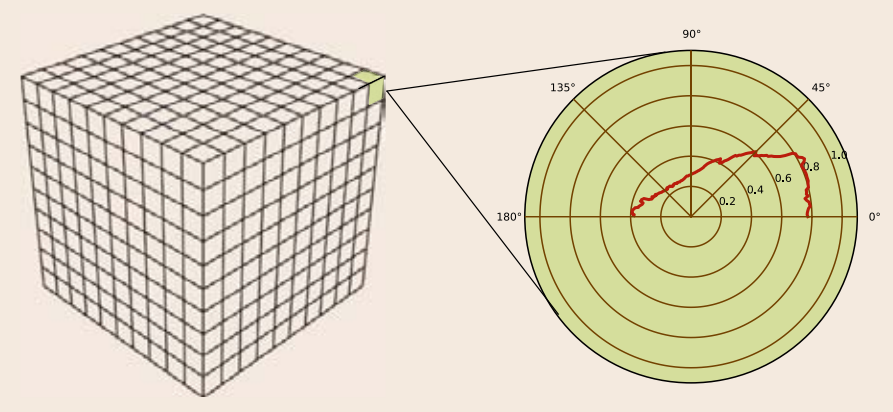


Figura 7.6: Procedimiento de reducción del espacio sensorial a través de un mapa de Kohonen cúbico. Se encuentra el vector más cercano a los distintos centroides del cubo, y se utilizan las coordenadas del centroide como entradas para la red neuronal del robot.

Para mitigar este problema, podemos generar un subconjunto de vectores sensoriales en donde cada vector sensorial guarde una distancia euclidiana de por lo menos un umbral h del resto de los vectores en el subconjunto. Estableciendo $h = 0.3$ se logra una reducción de $8,500$ a aproximadamente $5,400$ vectores, o bien del $36\,\%$.

En cuanto a las entradas que proporcionan la información de la meta actual del robot, se considera la orientación y la distancia actuales del robot con respecto a la meta. Sea $g = \{g_x, g_y\}$ la coordenada cartesiana de la meta actual y $p = \{p_x, p_y\}$ la coordenada cartesiana de la posición actual del robot, podemos procesar las entradas perceptuales de la meta de la siguiente manera:

ORIENTACIÓN: Sea $F = \{F_x, F_y\}$, $|F| = 1$ el vector unitario de la orientación frontal del robot y $D = \{g_x - p_x, g_y - p_y\}$, podemos obtener el ángulo O_θ que existe entre F y D despejándolo del producto punto de F y D:

$$F \cdot D = F_x D_x + F_y D_y$$

$$F \cdot D = |F||D|\cos(O_\theta)$$

$$O_\theta = \arccos\left(\frac{F_x D_x + F_y D_y}{|F||D|}\right) \tag{7.3}$$

Finalmente, debido a que si sólo se alimentara el valor del ángulo a la red neuronal, existiría un cambio numérico muy abrupto

entre 359° y 0°, los cuales representan orientaciones muy similares. Es por eso que se calcula el vector unitario O de O_θ y se alimentan dos entradas a la red neuronal, O_x y O_y. Debido a que se trata de un vector unitario, los valores pueden ser introducidos directamente a la red neuronal por encontrarse en el intervalo adecuado.

DISTANCIA: Representa simplemente la distancia que existe entre p y g. Sin embargo, es necesario acotar los valores posibles entre el intervalo $[-1, 1]$ para que sean procesados por la red neuronal. Esto implica que existen ciertos valores grandes de la distancia para los cuales no va ser posible tener una entrada en la red neuronal. Es necesario entonces establecer un umbral de sensibilidad o de visibilidad, buscando que la entrada se active para valores de distancia que se encuentren sólo dentro de este umbral. Podemos lograr *aplastar* el valor de la distancia (potencialmente infinita) en el intervalo $[0, 1]$ y mantener un umbral de sensibilidad de aproximadamente tres metros con la siguiente función (Figura 7.7):

$$s = 1 - \frac{1}{1 + e^{-3\,\mathrm{dist}+5}} \tag{7.4}$$

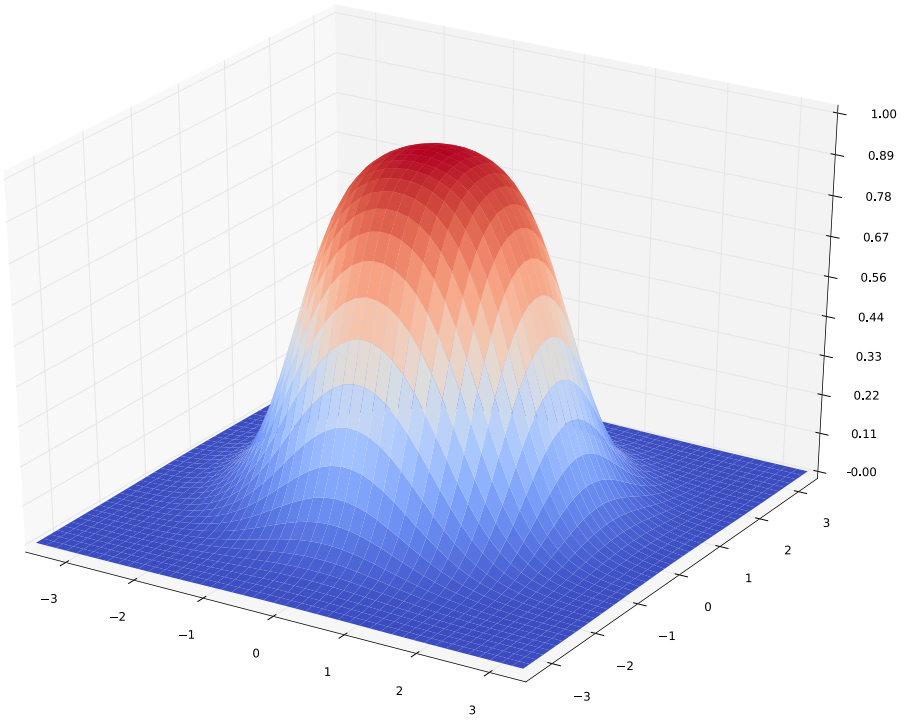


Figura 7.7: Función de estímulo por cercanía a la meta. Los ejes x y y representan los componentes de la distancia entre el robot y su meta.

7.5.2 *Procesamiento neuronal*

Una vez que se tienen los valores de entrada correspondientes acotados y escalados en el intervalo $[-1, 1]$, la red neuronal comienza su trabajo. El objetivo de la red neuronal es generar a partir de los valores de entrada correspondientes a su tarea actual los valores de control que generarán las acciones del robot. Concretamente, todas las redes neuronales tendrán dos salidas en el intervalo $[-1, 1]$ que corresponden a la velocidad lineal y agular del robot, las cuales serán escaladas a los límites establecidos anteriormente (7.3) antes de ser aplicadas.

Las entradas para la tarea de *exploración* son las tres coordenadas de la percepción actual de los lasers después de ser procesadas por el mapa de Kohonen (Figura 7.8). Las entradas para la tarea de *navegación* tienen los tres valores adicionales de orientación (Ecuación 7.3) y distancia a la meta (Figura 7.9):

k_x: Componente x de la ubicación de la neurona en el mapa de Kohonen.

k_y: Componente y de la ubicación de la neurona en el mapa de Kohonen.

k_z: Componente z de la ubicación de la neurona en el mapa de Kohonen.

o_x: Componente x del vector unitario de orientación hacia la menta.

o_y: Componente y del vector unitario de orientación hacia la menta.

s: Estímulo por cercanía a la meta (Ecuación 7.4).

Figura 7.8: Entradas y salidas de la red neuronal para la tarea de *exploración*.

7.5.3 *Arquitecturas de redes neuronales*

Las distintas arquitecturas de redes neuronales que serán utilizadas con el fin de evaluar su rendimiento en las dos tareas

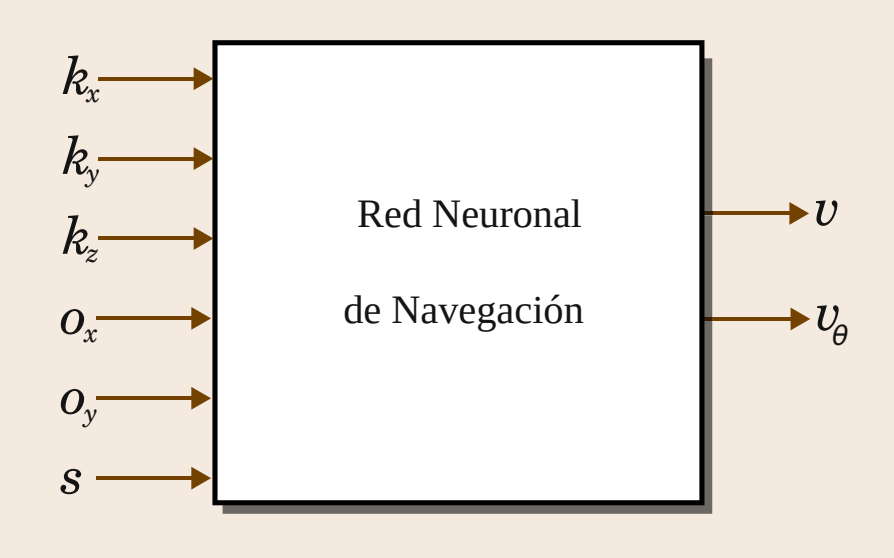


Figura 7.9: Entradas y salidas de la red neuronal para la tarea de *navegación*.

planteadas anteriormente presentan diferencias estructurales importantes que las llevarán a realizar cómputo interno de una manera particular. Se optó por evaluar las siguientes arquitecturas:

- Red Neuronal Recurrente Simple (SRN). Los detalles de esta red se encuentran en el Apéndice A.
- Red Secuencial en Cascada con Unidad de Decisión (ESCN). Los detalles de esta red se encuentran en el Apéndice B.
- Long Short-Term Memory (LSTM). Los detalles de esta red se encuentran en el Apéndice C.

La vasta mayoría de redes neuronales recurrentes utilizadas para aprendizaje y control de robots y agentes autónomos [88] hacen uso de *retroalimentación de primer orden*. Esto significa que ciertos valores de activación son retroalimentados y usados como entradas extra a algunas de las neuronas (típicamente en la capa de entrada) en un paso de tiempo subsecuente (normalmente el siguiente). La *retroalimentación de segundo orden*, por otro lado, modula o adapta los pesos de conexión y los sesgos. A diferencia de otras áreas, como el reconocimiento de lenguajes formales, hay muy pocos casos donde las *redes neuronales de segundo orden* (Apéndice B) han sido usadas como arquitecturas para el control de robots. Se debe señalar que las redes neuronales de primer y segundo orden son equivalentes computacionalmente [79, 78]. Esto significa que cada tarea que puede ser resuelta por un a red neuronal de segundo orden también podrá ser resuelta por una red de primer orden. Sin embargo, la equivalencia computacional de las arquitecturas de redes neu-

ronales no dice mucho acerca de qué tan apropiadas son para resolver tareas particulares en la práctica. Lo anterior será investigado experimentalmente en esta tesis.

8

PRUEBAS Y RESULTADOS

8.1 RENDIMIENTO DE LAS ARQUITECTURAS

Al comparar los resultados del uso del algoritmo genético ecléctico contra los resultados de NSGA-II con operadores genéticos del algoritmo ecléctico se encontró que el uso de NSGA-II generaba individuos con capacidades de navegación relevantes de manera mucho más lenta que el algoritmo genético ecléctico, en todas las instancias de prueba. Probablemente esto se deba a que en realidad los objetivos de navegación y el de minimización de colisiones ya están relacionados en el simulador. Un individuo que tiene muchas colisiones con el ambiente reduce el tiempo de vida que tiene disponible en el ambiente de simulación, y entre menor tiempo de simulación tenga, menor será su oportunidad de generar un buen puntaje en la tarea de navegación actual. Un individuo que tenga sólo un segundo de simulación no podrá alcanzar ninguno de sus objetivos y tampoco podrá explorar espacios nuevos en su ambiente, no le da tiempo. Los individuos que tienen pocas colisiones con el ambiente tienen la oportunidad de lograr mejor puntaje en ambas tareas.

Debido a que las simulaciones son sumamente costosas computacionalmente, el primer paso experimental es determinar cuál de las arquitecturas es más prometedora para cada tarea. Podemos muestrear estadísticamente su rendimiento eligiendo un tamaño de población y un número de generaciones que consuman un tiempo razonable por cada ejecución del EGA. También es necesario establecer un número de neuronas proporcional para cada una de las arquitecturas que serán evaluadas, y que este número no presente un espacio de búsqueda muy grande para el EGA. Los siguientes parámetros se establecieron para esta ronda de experimentos:

POBLACIÓN: 100 individuos

GENERACIONES: 100

NEURONAS EN LA CAPA OCULTA: 4

NEURONAS DE CONTEXTO/MEMORIA: 2

INTERVALO DE VALORES POSIBLES PARA LOS PESOS: $(-2, 2)$ para LSTM y ESCN, $(-1, 1)$ para SRN.

BITS POR PESO: 8 bits en representación de punto fijo. 1 bit de signo, 1 bit entero y 6 bits decimales.

EJECUCIONES DE EGA POR CADA ANN: 20

Resultados experimentales iniciales mostaron que intentar entrenar una SRN con pesos en el intervalo $(-2, 2)$ generaban individuos inestables inicialmente, y el algoritmo tomaba tiempo considerable en encontrar individuos aptos. El cambio al intervalo $(-1, 1)$ favorece la estabilidad de la red neuronal y rinde mejores resultados en un lapso menor de tiempo.

8.1.1 *Resultados de navegación*

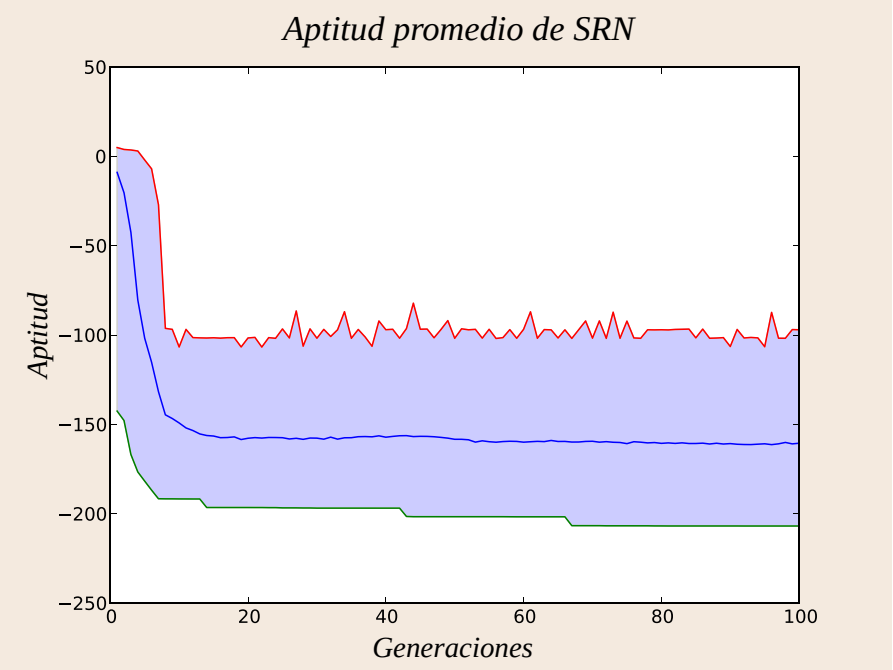


Figura 8.1: Rendimiento promediado de 20 ejecuciones del EGA del mejor individuo (verde), el peor individuo (rojo) y el promedio global de la población (azul), para SRN en la tarea de navegación.

Como se puede apreciar al comparar las gráficas de aptitud de las distintas redes neuronales (Figura 8.1, Figura 8.3 y Figura 8.2), en promedio ESCN alcanza valores de aptitud más altos al final de 100 generaciones de evolución. Es importante recordar que EGA está buscando *minimizar* los valores de aptitud de los individuos, es decir, entre menor sea el valor de evaluación, más apto resulta el individuo evaluado. También parece tener una diversidad mayor en términos del comportamiento que exhibe la población, a pesar de que comparte el mismo objetivo que las otras redes. En segundo lugar en rendimiento le pertenece a LSTM, y el tercero a SRN. Una observación interesante es que

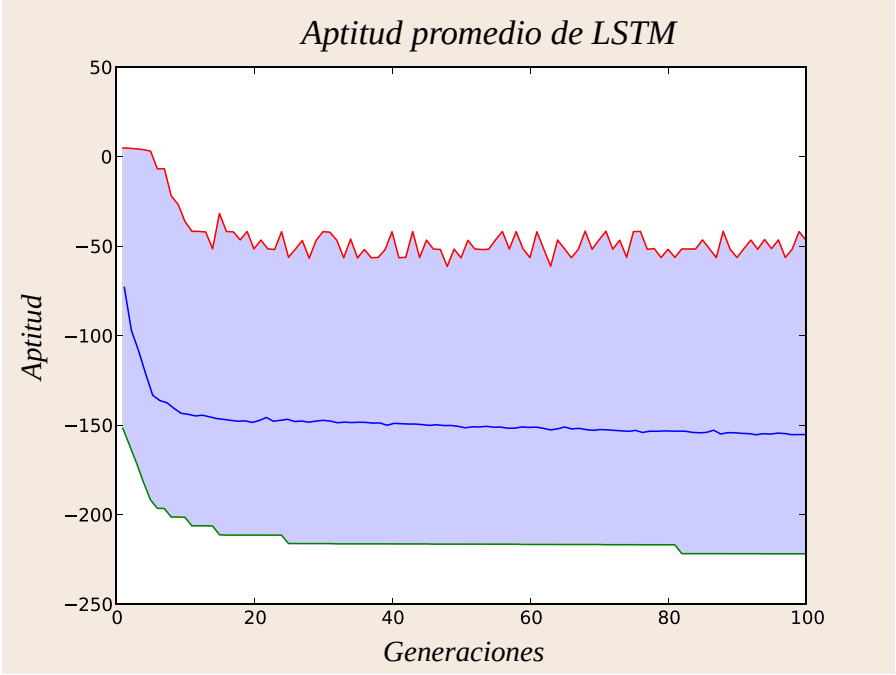


Figura 8.2: Rendimiento promedio de 20 ejecuciones del EGA del mejor individuo (verde), el peor individuo (rojo) y el promedio global de la población (azul), para LSTM en la tarea de navegación.

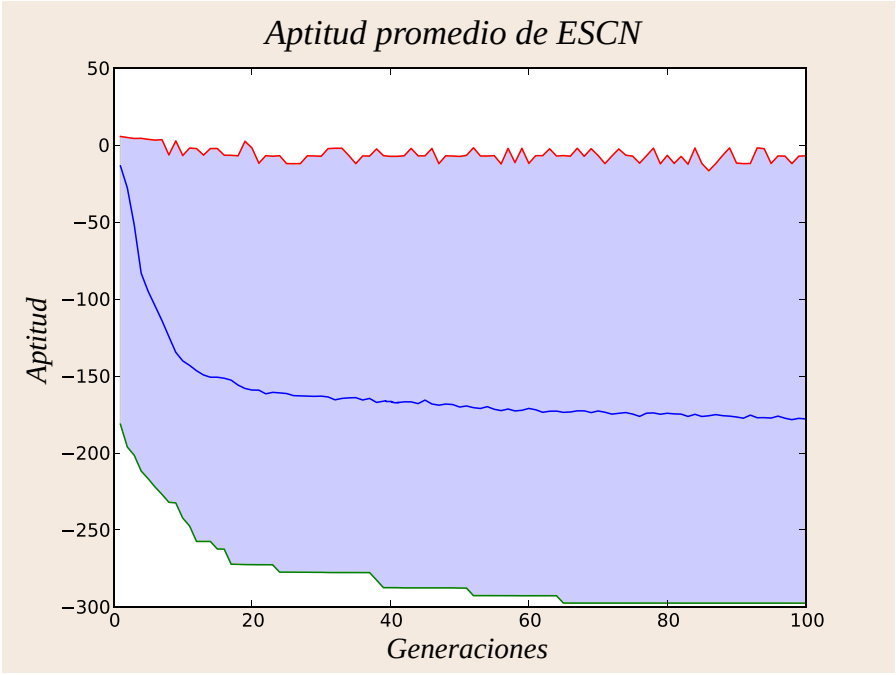


Figura 8.3: Rendimiento promedio de 20 ejecuciones del EGA del mejor individuo (verde), el peor individuo (rojo) y el promedio global de la población (azul), para ESCN en la tarea de navegación.

la diferencia entre el individuo más apto y el menos apto en cada generación es proporcional a la aptitud de esa arquitectura. Esto pue-

de ser validado visualmente, observando que la gráfica de aptitud de ESCN tiene un área azul mayor a las otras, y la gráfica de LSTM tiene un área azul mayor a la de SRN.

Arquitectura	Aptitud	Metas alcanzadas
SRN	−299.619	3
ESCN	−394.78	4
LSTM	−394.509	4

Tabla 8.1: Los mejores valores de aptitud obtenidos durante las 20 simulaciones por cada red neuronal para la tarea de navegación. ESCN y LSTM presentan el mismo potencial para resolver la tarea logrando generar un individuo que alcanza 4 objetivos sucesivamente en el ambiente, mientras que SRN logra un individuo que sólo alcanza 3.

Lamentablemente, ninguno de los tres individuos más aptos de las diferentes arquitecturas demostró tener un rendimiento satisfactorio para la tarea de navegación, pero el más apto para ella parece ser el de ESCN, por lo que una ejecución más extensa en cuanto a ploblación y generaciones de EGA se realizará para esta arquitectura, con una población de *2,000 individuos* durante *300 generaciones*. Con el fin de refinar los resultados de la búsqueda lo más posible, al final de la ejecución de EGA, utilizaremos el algoritmo Random Mutation Hill-Climber (RMH), (Apéndice E), pero en vez de tener un individuo inicializado aleatoriamente al principio del algoritmo, utilizaremos el mejor indiviuo encontrado por la ejecución anterior de EGA.

Como se puede observar en la Figura 8.4, el individuo más apto se encuentra relativamente pronto en la ejecución del algoritmo. Sin embargo, este individuo alcanza una meta más que el mejor individuo encontrado anteriormente. Afortunadamente, la ejecución posterior de RMH logra hacer una sola mejora en la aptitud de este individuo después de 104,711 iteraciones de RMH, equivalentes a aproximadamente 52 generaciones de EGA. Esta mejora aumenta la aptitud de −497.453 a −594.227 y significa que el individuo generado logra alcanzar el último objetivo en el ambiente. Esto se confirma al realizar la simulación en el ambiente gráfico del programa, trazando la ruta seguida para lograr la tarea, como se muestra en la figura Figura 8.5. Un resultado de este tipo es interesante (por lo menos para el autor), ya que el código genético de este individuo en particular tiene una longitud de 1,128 bits, y por lo tanto el tamaño del espacio de búsqueda en donde operan EGA y RMH es de 2^{1128}, un número de combinaciones posibles mayor al estimado de átomos en todo el universo observable.

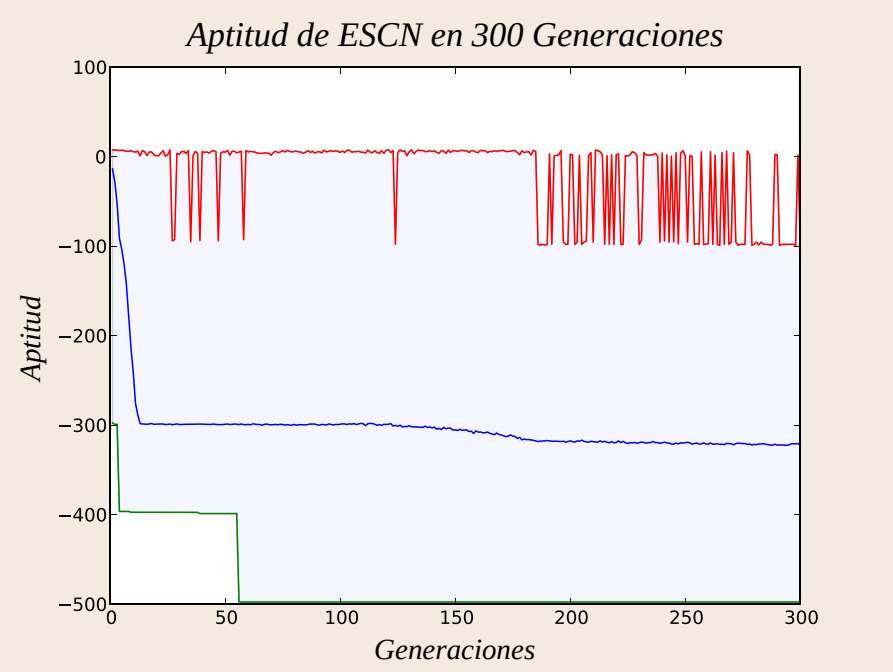


Figura 8.4: Rendimiento a lo largo de 300 generaciones de una sola ejecución de EGA. Se muestra la aptitud del mejor individuo (verde), el peor individuo (rojo) y el promedio de la población (azul). El algoritmo encuentra su aptitud más alta en la generación 55 y continúa sin progresar el resto de las generaciones.

8.1.1.1 *Evaluación de la ruta*

Se ha logrado encontrar un comportamiento capaz de generar una ruta que alcanza todos las metas de ambiente y evita las colisiones con obstáculos fijos. Podemos medir la distancia total recorrida por el robot acumulando la distancia recorrida en cada paso de simulación para determinar que tan eficiente es el comportamiento. Una ruta más larga implica mayor tiempo de navegación y utilización de energía, por lo que un robot eficiente en la construcción de su ruta de navegación tiene la capacidad de realizar más acciones en el ambiente.

Podemos construir una evaluación de cualquier ruta generada por un comportamiento al compararla con una ruta creada por un experto. La ruta del experto es construida con conocimiento completo del ambiente y de sus propiedades geométricas. En contraste, la ruta generada por el comportamiento es determinada a partir de la información obtenida dinámicamente de los sensores del robot, sin existir alguna representación interna del ambiente en el controlador al inicio de la simulación. Otra desventaja que el controlador del robot debe de superar es que la simulación induce errores de sensado y actuación, por lo que seguir una trayectoria bien definida es poco factible. Al considerar lo anterior, es sensato esperar que la ruta dada por el

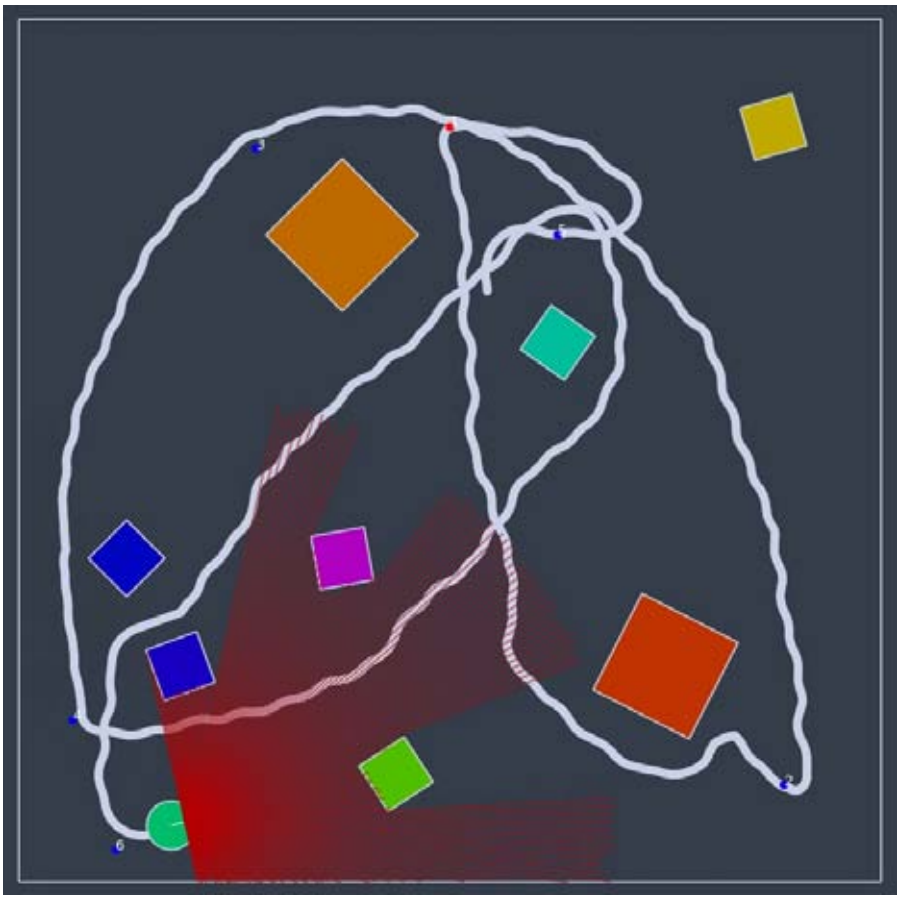


Figura 8.5: Ruta seguida por el individuo más apto para llegar a los seis objetivos en el ambiente. Podemos observar como el individuo logra negociar entre su objetivo actual (alcanzar la meta), y evitar los obstáculos en el ambiente que se encuentran en su camino.

experto sea más cercana al óptimo que las rutas generadas por lo comportamientos.

A partir del ambiente de entrenamiento, las dos rutas son generadas (Figura 8.6). Las longitud total de cada ruta es la siguiente:

RUTA DEL EXPERTO: 39.97 metros

RUTA DEL COMPORTAMIENTO: 43.47 metros

Esto quiere decir que la ruta generada por el comportamiento del mejor individuo tiene un error de 8.76 % con respecto a la ruta del experto.

8.1.2 *Evaluación estadística de un controlador*

Es importante validar que el ambiente utilizado para el entrenamiento del controlador sea representativo de las posibles configuraciones de obstáculos y metas a los que el controlador puede ser sometido. Uno podría generar a mano algunos ambientes configurados de

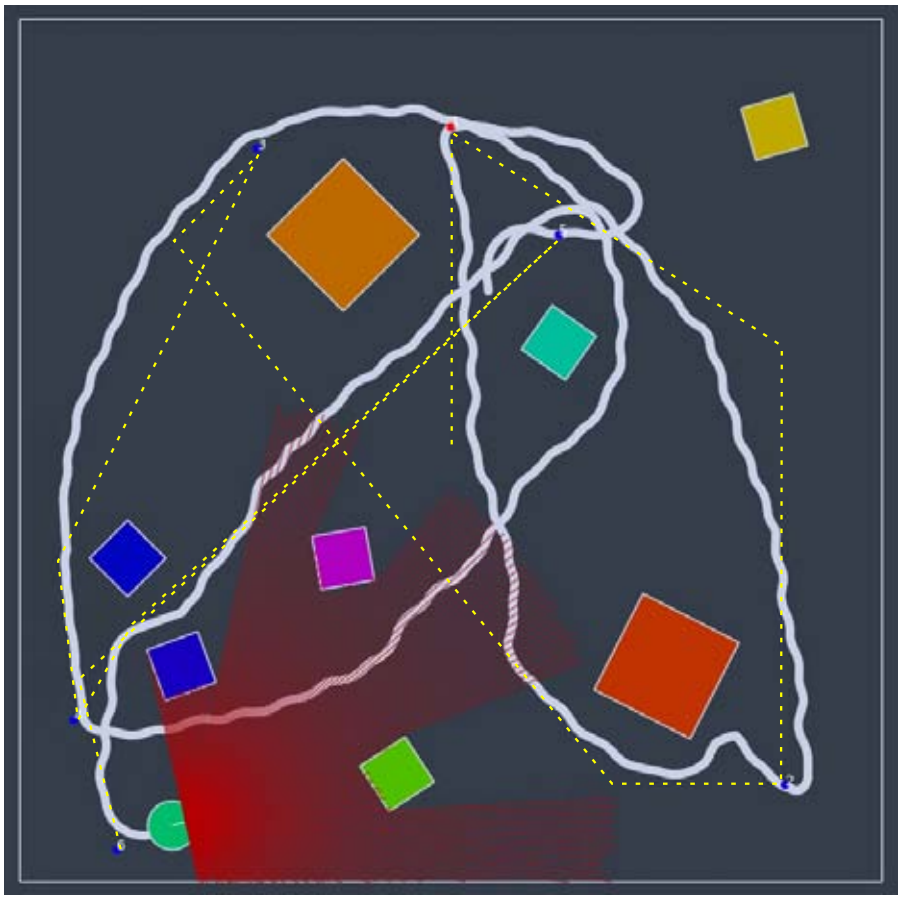


Figura 8.6: Comparación entre la ruta del experto (linea punteada amarilla) y la ruta generada por el comportamiento del mejor individuo encontrado (linea blanca).

manera distinta al de entrenamiento y simular el controlador dentro de ellos. Pero al obtener las métricas de evaluación resultantes, no podría decirse mucho acerca del rendimiento esperado del controlador en general. Además, estos ambientes estarían sesgados por criterios subjetivos del diseñador y no serían muy grandes en número.

Una mejor estrategia es generar ambientes donde las configuraciones de los obstáculos y las metas son alimentadas aleatoriamente a partir de valores reales de una distribución uniforme, es decir, todas las configuraciones del espacio de los posibles ambientes pueden ser generadas con igual probabilidad. Si generamos suficientes ambientes aleatorios, estaremos obteniendo una *muestra* representativa del espacio de ambientes a partir de la cual podemos obtener métricas que nos permitirán realizar conclusiones generales del controlador.

8.1.3 *Resultados de exploración*

Nuevamente, la ventaja de ESCN sobre las otras arquitecturas se puede verificar en las gráficas de aptitud de las distintas redes neuro-

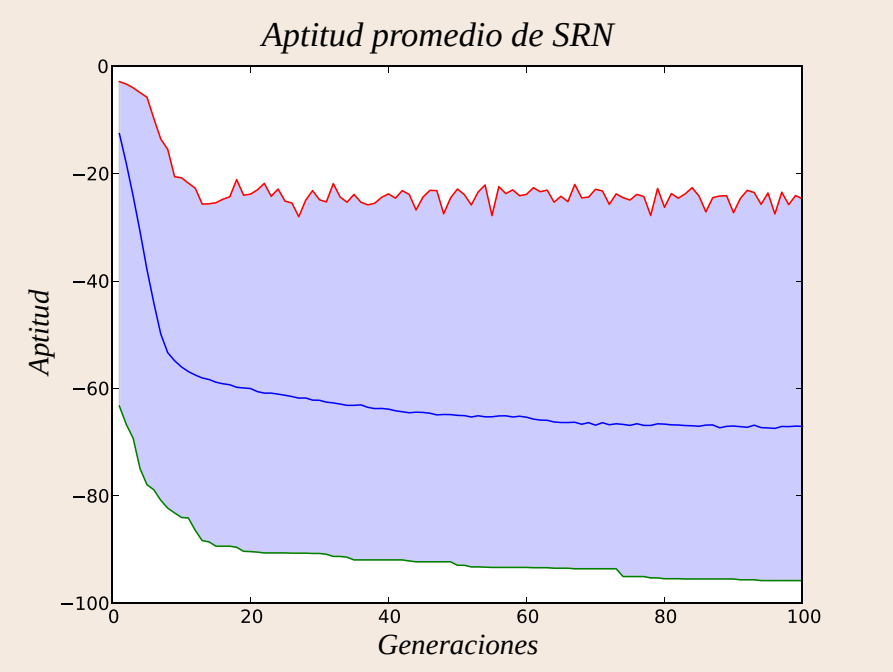


Figura 8.7: Rendimiento promediado de 20 ejecuciones del EGA del mejor individuo (verde), el peor individuo (rojo) y el promedio global de la población (azul), para SRN en la tarea de exploración.

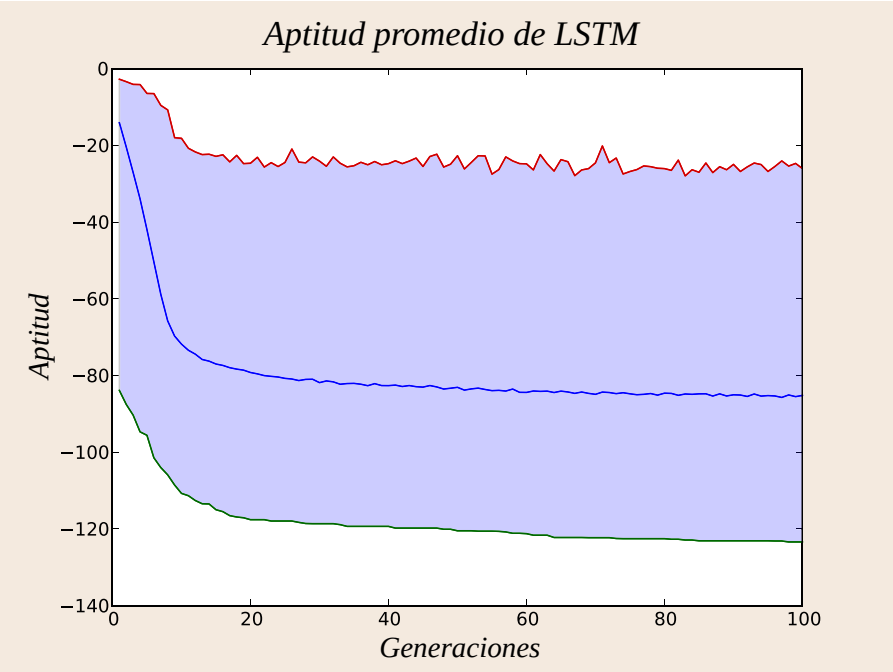


Figura 8.8: Rendimiento promediado de 20 ejecuciones del EGA del mejor individuo (verde), el peor individuo (rojo) y el promedio global de la población (azul), para LSTM en la tarea de exploración.

nales (Figura 8.7, Figura 8.9 y Figura 8.8). De hecho, el mismo ordena-

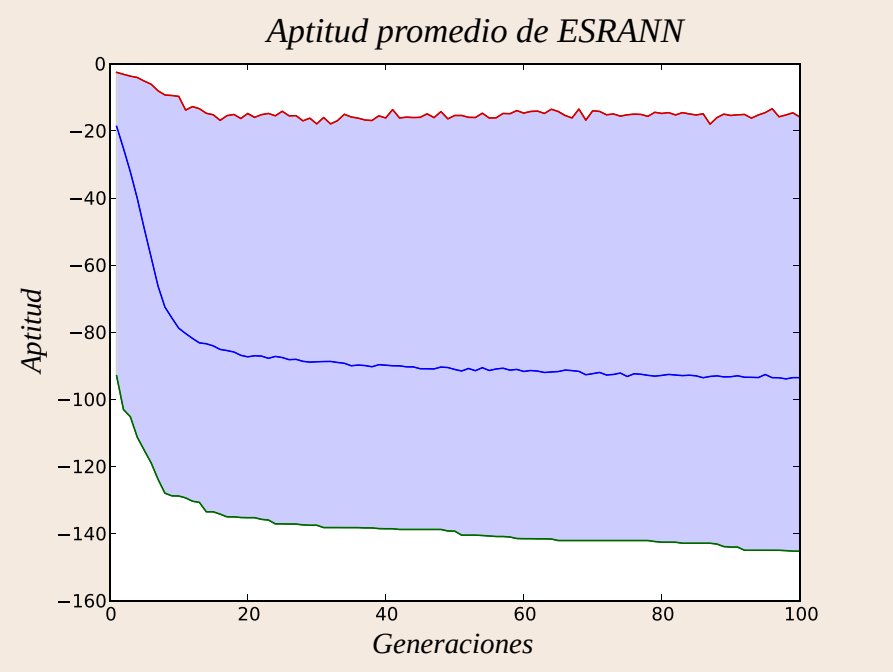
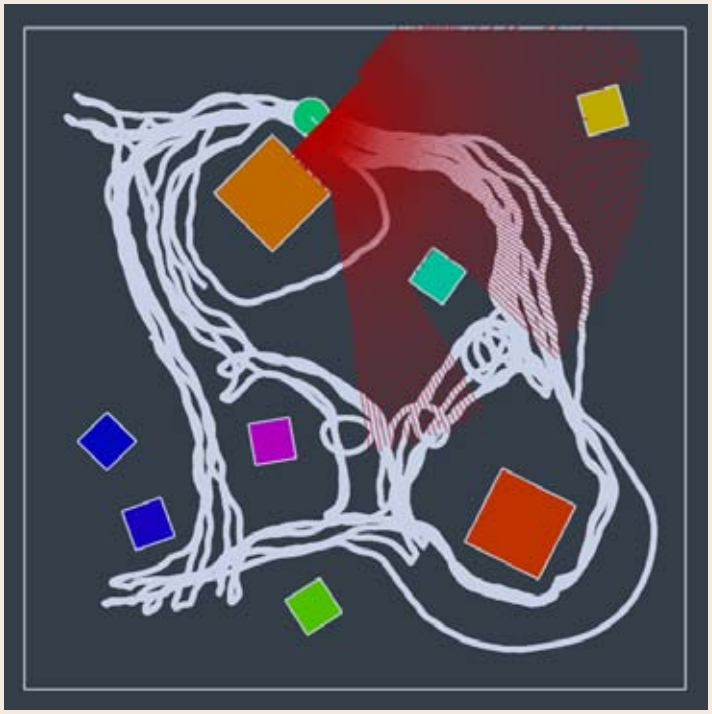


Figura 8.9: Rendimiento promediado de 20 ejecuciones del EGA del mejor individuo (verde), el peor individuo (rojo) y el promedio global de la población (azul), para ESCN en la tarea de exploración.

miento de aptitud obtenido en la tarea de navegación se preserva, al igual que la diferencia entre el individuo más apto y el menos apto.

Para la tarea de exploración, el individuo más apto encontrado en las 20 ejecuciones de EGA para ESCN realiza la tarea de exploración de manera efectiva en el ambiente (Figura 8.10), por lo que no es necesario realizar una búsqueda exhaustiva como la que se realizó en la tarea anterior.

A) Aptitud Media



B) Aptitud Alta

Figura 8.10: Comportamiento de dos individuos en el ambiente, ilustrado por la trayectoria seguida durante cinco minutos de simulación de ambos. El individuo *B* tiene mayor aptitud que el individuo *A* porque explora una mayor catindad del espacio del ambiente.

9

CONCLUSIONES Y TRABAJO A FUTURO

El presente trabajo expone la relevancia de las RANNs para el aprendizaje y control robótico, y una forma práctica en que pueden ser utilizadas para construir sistemas de procesamiento sensorial y acción reactiva. Particularmente, se enfoca en que los robots controlados por RANNs tienen la capacidad de crear estructuras de control y representaciones internas al interactuar con un ambiente. Estas propiedades fueron expuestas y analizadas experimentalmente, comparando tres diferentes arquitecturas de control neuronal recurrentes en dos tareas que requieren que el robot exhiba comportamientos dependientes del *contexto*. Los resultados cuantitativos mostraron que el mejor rendimiento en ambas tareas fue lograda por la arquitectura ESCN, una variación de la SCN de Pollack. Sin embargo, el hecho de que esta arquitectura sea más apta para el control robótico no puede ser establecido por completo, ya que esta suposición se debe completamente a los detalles de la configuración experimental y al procedimiento de entrenamiento evolutivo, y sólo puede ser determinada con un grado mayor de certeza con más experimentación extensiva.

En cuanto a la apreciación cualitativa del comportamiento de las arquitecturas de control a través del ambiente gráfico, parece ser que que las estructuras internas de la red no mapean las estructuras ambientales; más bien son responsables de generar un comportamiento funcionalmente adecuado que es desencadenado y modulado por el ambiente, y determinado por la estructura interna de los pesos sinápticos. Este comportamiento es el resultado de procesos adaptativos, y estos son los que han cambiado la arquitectura a lo largo de las generaciones de tal manera que su estructura física establezca las propiedades dinámicas para mantener un estado de equilibrio que le permite actuar efectivamente.

El método general de la robótica evolutiva presenta retos interesantes para los cuales técnicas particulares han sido desarrolladas anteriormente en la literatura, y otras propuestas en el presente trabajo. Tal vez el problema más frecuentemente enfrentado por los practicantes del área sea el de el balance entre el poder computacional necesario para resolver la tarea efectivamente (el cual está directamente relacionado con el tamaño de la red) y el considerable incremento del espacio de búsqueda que este aumento de la red presenta, afectan-

do la capacidad para encontrar una solución factible del algoritmo genético. Las formas de mitigar este problema expuestas en la tesis son:

1. Reducir la dimensionalidad de la información sensorial (comprimir).
2. Aumentar gradualmente la complejidad topológica de la red neuronal.
3. Reducir adaptativamente el tiempo de simulación de individuos poco aptos con el fin de acelerar el proceso de búsqueda, y aumentar las generaciones y población del algoritmo genético.

Aunque se lograron encontrar individuos con comportamientos útiles para hacer frente a las distintas tareas, aún es posible mejorar su rendimiento, principalmente en la evasión de obstáculos, en la velocidad de solución de la tarea y en las capacidades sensoriales del robot. En este capítulo se proponen ciertas extensiones al procedimiento que pueden ayudar a mejorar el rendimiento en los aspectos anteriores. También se contempla trabajo que no ha sido tratado en la literatura de la robótica evolutiva y podría representar una contribución valiosa si se llegara a un resultado relevante.

9.1 ALGORITMO NEAT CON CONEXIONES DE SEGUNDO ORDEN

Como se demostró en el capítulo de resultados (Capítulo 8), las arquitecturas que incorporan retroalimentación de segundo orden presentan una clara ventaja sobre las que utilizan retroalimentación simple en ambas tareas de navegación.

Sería interesante incorporar la habilidad de crear conexiones que permitan la retroalimentación de segundo orden a través del aumento de la estructura de la red. De esta forma, se aumenta la complejidad de la red con un mecanismo que puede resultar más benéfico para la capacidad de los individuos de resolver el problema que una conexión retroalimentada simple. También es posible mantener la capacidad del algoritmo de realizar una mutación de conexión para los dos tipos de retroalimentación, buscando observar la composición de conexiones del individuo más apto encontrado al final del algoritmo, respondiendo a las siguientes preguntas:

- ¿Qué proporción del total de las conexiones son conexiones de primer orden?
- ¿Qué proporción del total de las conexiones son conexiones de segundo orden?

El resultado más importante de este trabajo sería poder demostrar que agregar la capacidad de crear conexiones de segundo orden au-

menta (o no) considerablemente la aptitud de los individuos generados, a través de pruebas estadísticas extensivas.

9.2 ADICIÓN DE LA NOVEDAD COMO OBJETIVO DEL ALGORITMO

Lamentablemente, NSGA-II no logró producir resultados significativamente mejores al considerar los objetivos de manera separada, posiblemente porque existía ya cierta correlación entre ellos. Pero, ¿qué pasaría si agregamos a la innovación como un objetivo adicional al objetivo principal? Hipotéticamente, al ordenar los distintos frentes de Pareto, los individuos más aptos serían los de mayor innovación, y dentro de estos, los más aptos sería los que mejor satisfacen al objetivo principal. Esto resulta útil porque cuando la búsqueda del objetivo principal se queda atascada en un mínimo local, eventualmente el archivo de novedad saturará esa región de comportamientos y la búsqueda de novedad favorecerá a otros individuos que proporcionen innovaciones en su comportamiento. Si estos individuos con comportamiento novedoso resultan ser mejores para resolver el objetivo de búsqueda principal que los que estaban en el mínimo local anterior, habremos desatorado al algoritmo de búsqueda principal.

9.3 ESQUEMA DISTINTO DE COMPRESIÓN SENSORIAL

A pesar de que los algoritmos de búsqueda lograban encontrar un individuo que manifestaba un comportamiento útil para la tarea, esporádicamente tenían colisiones indeseadas con los obstáculos en el ambiente, a pesar de que los sensores proporcionaban información suficiente para evitarlos. Es probable que la compresión sensorial realizada con el mapa de Kohonen no representara adecuadamente la información actual de los sensores al momento de la colisión. Además, la obtención de las coordenadas de un vector sensorial en el mapa de Kohonen resulta ser bastante costosa computacionalmente, ya que requiere un cálculo de distancia euclidiana entre dos vectores de 20 valores de punto flotante por cada centroide del mapa. En el trabajo presente el mapa de Kohonen consta de 1,000 centroides.

En un trabajo relacionado con este problema [42], se propone utilizar redes neuronales convolucionales (Convolution Neural Network (CNN)) [22, 48] como un mecanismo de compresión sensorial para reducir el espacio de búsqueda de un algoritmo neuroevolutivo. Las CNNs son redes jerárquicas profundas que recientemente se han convertido el estado del arte en reconocimiento de patrones y procesamiento de imágenes debido a implementaciones en multiprocesadores de gráficos (GPUs).

Las CNNs tienen dos partes:

1. Un *detector de características* profundo que consiste en capas alternantes de *convolución* y *submuestreo*.

2. Un clasificador que recibe la salida de la capa final del detector de características.

Cada capa convolucional ℓ, tiene un banco de $m^\ell \times n^\ell$ filtros, F^ℓ, donde m^ℓ es el número de mapas de entrada (imágenes), I^ℓ, a la capa, y n^ℓ es el número de mapas de salida (entradas a la siguiente capa). Cada mapa de salida es calculado por:

$$I_i^{\ell+1} = \sigma\left(\sum_{j=1}^{m^\ell} I_j^\ell * F_{ij}^\ell\right), \quad i = 1 \dots n^\ell,\ \ell = 1,3,5\dots$$

donde $*$ es el operador de convolución, F_{ij}^ℓ es el i-ésimo filtro para el j-ésimo mapa y σ es la función sigmoide en el intervalo $(0,1)$. Se debe notar que ℓ siempre es impar por las capas de submuestreo entre cada una de las capas convolucionales.

Las capas de submuestreo reducen la resolución de cada mapa. Cada mapa es particionado en bloques que no se traslapan y un valor de cada bloque es utilizado en el mapa de salida. La operación de *max-pooling* es utilizada en este paso, ya que submuestrea el mapa simplemente tomando el máximo valor en el bloque como la salida, convirtiendo estas redes en Max-Pooling Convolution Neural Networks (MPCNN) [15, 75].

Las capas alternantes transforman la entrada representaciones abstractas que reducen su dimensionalidad progresivamente, y que después son clasificadas típicamente utilizando una FANN, que tiene una neurona de salida por cada clase. La red completa es usualmente entrenada utilizando el algoritmo de retropropagación. Por supuesto, esto requiere conocimiento *a priori* de etiquetado para el conjunto de entrenamiento, y en este contexto, no tenemos esa información. En vez de eso, la MPCNN se *evoluciona* sin utilizar un conjunto de entrenamiento etiquetado. Un conjunto de k vectores sensoriales se recolecta del ambiente, y luego las MPCNNs son evolucionadas para maximizar la función de aptitud:

$$f_{kernels} = min(D) + mean(D)$$

donde D es una lista de todas las distancias euclidianas,

$$d_{i,j} = \left\| f_i - f_j \right\|, \quad \forall i > j$$

entre los k vectores de características normalizados $\{f_1 \dots f_k\}$ que se generaron a partir de los k vectores sensoriales en el conjunto de entrenamiento por la MPCNN codificada en el genoma.

Esta función de aptitud obliga a las MPCNNs que están haciéndose evolucionar a generar vectores de características que estén dispersos

en el espacio de características, de tal manera que cuando la mejor MPCNN encontrada al final del algoritmo genético procese imágenes para los controladores evolucionados, proporcione suficiente poder discriminatorio para permitirles tomar las acciones correctas.

Parte V

APÉNDICES

A

RED NEURONAL RECURRENTE SIMPLE

Las *redes neuronales recurrentes de primer orden* (First-order Recurrent Artificial Neural Network (FRANN)) son básicamente FANNs con conjunto de *conexiones retroalimentadas* de algunos nodos en la capa oculta a los nodos en la capa de entrada (Figura A.1). Después de su entrenamiento, típicamente los pesos permanecen fijos. La red neuronal de primer orden que será considerada en el presente trabajo es una SRN [18], también llamada *red de Elman*, al igual que su inventor. Los nodos ocultos que se copian a la capa de entrada se conocen como *nodos de contexto*. La activación de los nodos de contexto en el tiempo t será parte de la entrada en el tiempo $t + 1$. De esta manera, las neuronas ocultas tienen un registro de sus activaciones previas, lo cual le permite a la red realizar tareas de aprendizaje que se extienden a lo largo del tiempo. Las neuronas ocultas también alimentan las neuronas de salida que reportan la respuesta de la red al estímulo aplicado externamente. Fuera de la conexión de los nodos de contexto a la entrada, la operación de cada nodo en la red es exactamente como los nodos en las FANNs. Debido a la naturaleza de la retroalimentación hacia las neuronas ocultas, estas neuronas pueden continuar reciclando información en la red a lo largo de varios pasos de tiempo, y por lo tanto descubrir representaciones abstractas en el tiempo. Por lo tanto, la SRN no es meramente un registro simple de los datos pasados.

Elman [18] describe el uso de la red neuronal simple para descubrir los límites de las palabras en una fuente continua de fonemas sin ningún límite en la representación. La entrada a la red neuronal representa el fonema actual. La salida representa un intento de adivinar cuál es el siguiente fonema de la secuencia. El rol de las unidades de contexto es proveer a la red de *memoria dinámica* para codificar la información contenida en la secuencia de fonemas, que es relevante para la predicción.

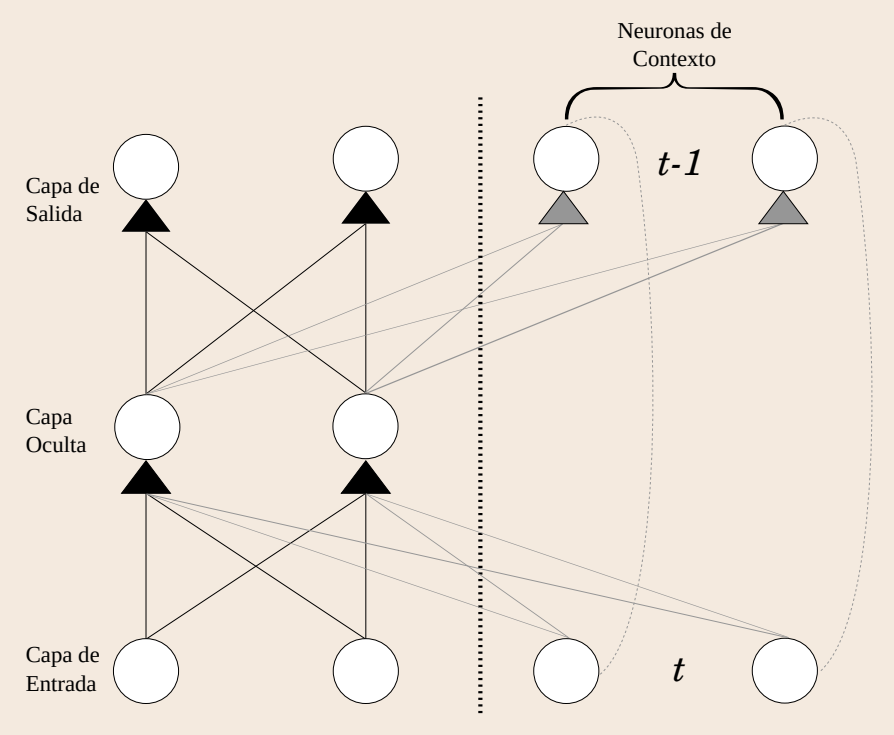


Figura A.1: Red neuronal recurrente simple con dos neuronas de contexto.

B

RED SECUENCIAL EN CASCADA CON UNIDAD DE DECISIÓN

En las *redes neuronales recurrentes de segundo orden* (Second-order Recurrent Artificial Neural Network (SRANN)), algunos de los pesos son una función de valores de activación previos. Estos pesos serán llamados *pesos dinámicos*. Esto significa que la función de la red será cambiada en cada paso de tiempo. La SRANN que será considerada en este proyecto será la SCN de Pollack [68]. Se puede pensar que la SCN está compuesta por dos subredes:

- Una *red funcional* que calcula la activación de las neuronas de salida.
- Una *red de contexto* que calcula algunos (o todos) los pesos de la red funcional. En el presente trabajo, los pesos de la capa de salida permanecen fijos, es decir, no son calculados por la red de contexto en cada paso de tiempo.

Los pesos de la red de contexto se llamarán *pesos de segundo orden* y permanecen fijos en la evaluación de la red. Para entender mejor la SCN tal vez sea mejor considerar una versión no recurrente de una red de segundo orden, por ejemplo, una que resuelva el problema XOR. El problema XOR consiste en aprender la función $\mathrm{XOR}(x_1, x_2)$ donde x_1 y x_2 son 0 o 1 y $\mathrm{XOR}(x_1, x_2) = 1$ si $x_1 \neq x_2$, de lo contrario $\mathrm{XOR}(x_1, x_2) = 0$. En la Figura B.1 se muestra una red neuronal simple no recurrente de segundo orden que resuelve el problema XOR (descrita originalmente en [68]). Esta red neuronal resuelve el problema con solo 4 pesos fijos en vez de 7, que son los necesarios para resolver el problema en una red neuronal de primer orden (incluyendo los pesos de sesgo). También se reporta en [68] que la SCN aprendió a resolver el problema XOR con aproximadamente cinco veces menos épocas típicamente.

Una red neuronal de segundo orden recurrente cambia sus pesos dinámicos de acuerdo con el estado interno (la activación de los nodos de contexto). Las neuronas que son la entrada a la red de contexto se llamarán *neuronas de contexto*. La activación de las neuronas de contexto actualiza los pesos de la red funcional en cada paso de tiempo, a través de los pesos en la red de contexto. La Figura B.2 describe una topología simplificada de las SCNs.

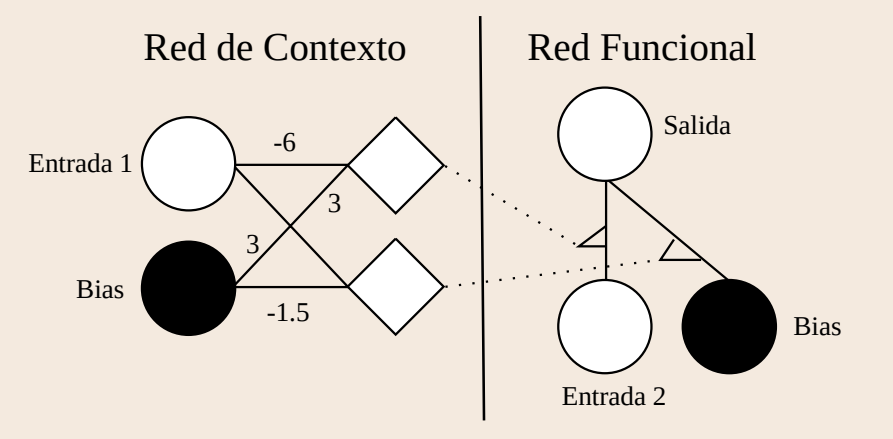


Figura B.1: Una red neuronal no recurrente de segundo orden que resuelve el problema XOR. Los diamantes representan unidades lineales donde el valor resultante es copiado, sin aplicarle ninguna función de escalamiento. Las conexiones con líneas punteadas indican que la activación de las unidades lineales es copiada a los pesos de la red funcional. Los nodos negros son unidades constantemente activas y los pesos de la conexiones a partir de ellos corresponden a los sesgos.

Los pesos dinámicos son determinados por:

$$W_{ix}(t) = c_1(t-1)W_{c_1W_{ix}} + c_2(t-1)W_{c_2W_{ix}} + \ldots + c_m(t-1)W_{c_mW_{ix}} + W_{b}W_{ix} \tag{B.1}$$

para un peso de una neurona de entrada i a una neurona de contexto o de salida x. La activación x de cualquier neurona de contexto o salida en la arquitectura de la Figura B.2 en el tiempo t está determinada por:

$$\begin{aligned} x(t) = \quad & f(c_1(t-1)(i_1(t)W_{c_1W_{i_1x}} + i_2(t)W_{c_1W_{i_2x}} + \ldots + i_n(t)W_{c_1W_{i_nx}} + W_{c_1W_{bx}}) + \\ & c_2(t-1)(i_1(t)W_{c_2W_{i_1x}} + i_2(t)W_{c_2W_{i_2x}} + \ldots + i_n(t)W_{c_2W_{i_nx}} + W_{c_2W_{bx}}) + \\ & \vdots \\ & c_m(t-1)(i_1(t)W_{c_mW_{i_1x}} + i_2(t)W_{c_mW_{i_2x}} + \ldots + i_n(t)W_{c_mW_{i_nx}} + W_{c_mW_{bx}}) + \\ & i_1(t)W_{bW_{i_1x}} + i_2(t)W_{bW_{i_2x}} + \ldots + i_n(t)W_{bW_{i_nx}} + W_{bW_{bx}}) \end{aligned} \tag{B.2}$$

donde $W_{\alpha W_{\beta x}}$ es el peso de segundo orden del nodo de contexto α al peso dinámico entre la neurona de entrada β y x. b es el *sesgo*, que siempre tiene una activación de 1,0.

Como se puede apreciar en la Ecuación B.2, la activación proveniente de la entrada y los nodos de contexto no sólo se suma, si no que se multiplica, por lo que podemos decir que la SCN tiene una activación *multiplicativa*.

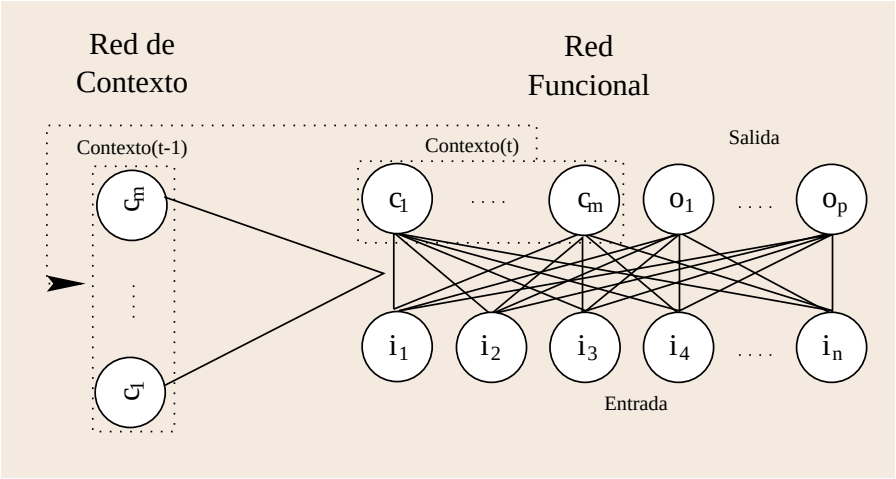


Figura B.2: Una SCN, como fue propuesta en [68]. La activación de la unidades de contexto en el tiempo t es la entrada de la red de contexto en el tiempo $t + 1$. Los sesgos han sido excluidos en esta figura, y también las unidades lineales que están presentes para cada peso en la red funcional. Esta figura carece de generalidad ya que pueden existir más capas en la red de contexto y la red funcional.

Una característica sumamente importante de las SRANNs es que son utilizadas para representar y aprender *autómatas determinísticos de estado finito* [85], es por eso que las SCNs han sido usadas para tareas de adquisición del lenguaje en [68, 69, 70, 35].

B.1 UNIDAD DE DECISIÓN

En vez de utilizar directamente una SCN, se optó por usar un modelo llamado ESCN por su autor [88] que incorpora una *unidad de decisión*, la cual en cada paso de tiempo determina si se aplicará la red de contexto para recalcular el valor de los pesos de la red funcional. Esta unidad de decisión es una neurona que tiene como entradas las salidas de las neuronas de contexto. La idea detrás de esta extensión es que el robot sea capaz de decidir selectivamente cuando cambiar su mapeo sensor-motor, en vez de restablecer la red funcional en cada paso de tiempo. La red de contexto se usa para adaptar la red funcional sólo cuando la activación de la unidad de decisión excede cierto umbral (0.5 para la función logística, cero para la tangente hiperbólica).

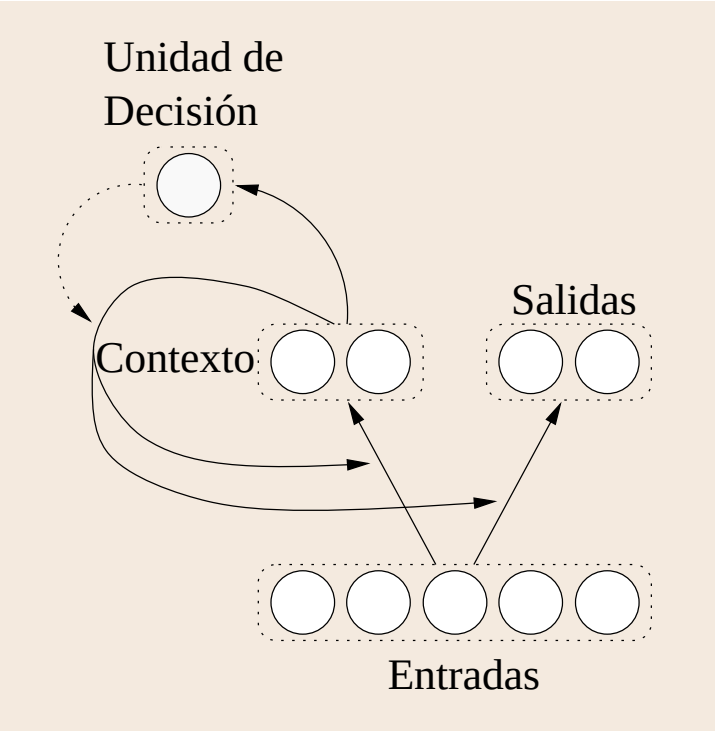


Figura B.3: Modelo simplificado de una ESCN. Una SCN con la unidad de decisión agregada con el fin de la autoregulación del cambio contextual.

C

LONG SHORT-TERM MEMORY

La unidad básica en la capa oculta de una red LSTM es el *bloque de memoria*; reemplaza las unidades ocultas en una RANN *tradicional* (Figura C.1). Un bloque de memoria contiene una o más *celdas de memoria* y tres compuertas adaptativas y multiplicativas, dos de las cuales regulan la entrada y la salida de todas las celdas en el bloque de memoria, y una llamada *compuerta de olvido* (forget gate), que restablece la memoria interna de la celda. Cada celda de memoria tiene en su núcleo una unidad lineal conectada recurrentemente a si misma llamada CEC, cuya activación llamamos el *estado* de la celda. Cuando las compuertas de entrada y salida están cerradas (activación alrededor de cero), las entradas irrelevantes y el ruido no entran en la celda, y el estado de la celda no perturba el resto de la red. La Figura C.2 muestra un bloque de memoria con una sola celda.

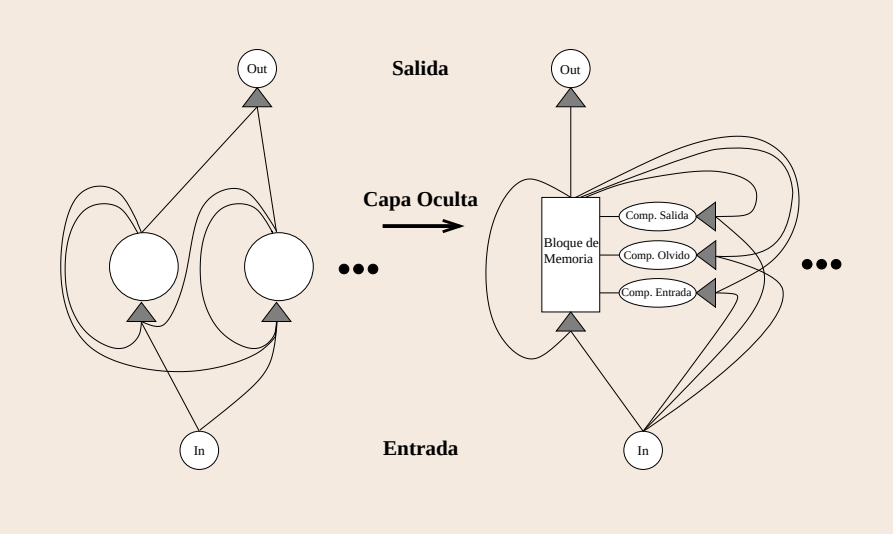


Figura C.1: *Izquierda*: RANN con una capa oculta completamente recurrente. *Derecha*: Red LSTM con bloques de memoria en la capa oculta (sólo se muestra uno).

Hochreiter y Schmidhuber (1997) [37] demostraron que LSTM puede resolver numerosas tareas que no podían ser resueltas por algoritmos previos para las RANNs. Sin embargo, el modelo original de LSTM (sin la compuerta de olvido) fallaba al intentar aprender a pro-

cesar correctamente ciertas series de tiempo muy largas o continuas que no habían sido segmentadas *a priori* en subsecuencias de entrenamiento apropiadas con comienzos y finales bien definidos en donde el estado interno de la red pudiera ser restablecido. El problema es que una fuente de entrada continua puede eventualmente causar que los valores internos de la celda crezcan sin control, aunque la naturaleza repetitiva del problema sugiera que deben ser restablecidos ocasionalmente.

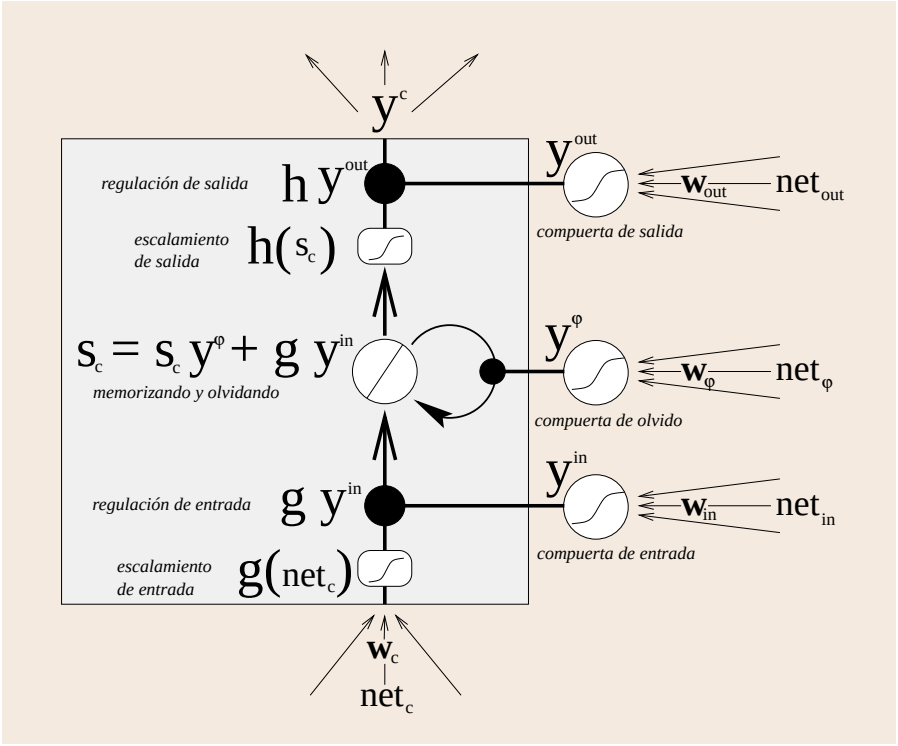


Figura C.2: Una celda LSTM tiene una unidad lineal con una conexión recurrente a si misma con un peso de 1,0 (CEC). Las compuertas de entrada y salida regulan el acceso de lectura y escritura a la celda cuyo estado es denotado por s_c. La compuerta del olvido multiplicativa puede restablecer el estado interno de la celda. La función g escala la entrada de la celda, mientras que la función h escala la salida de la celda.

LSTM ha sido utilizada para reconocer caracteres escritos a mano en línea, en donde un conjunto de coordenadas ordenadas en el tiempo representan el movimiento de la punta de la pluma. Un sistema basado en LSTM actualmente tiene el mejor rendimiento alcanzado para esta tarea, con una tasa de reconocimiento de palabras del 74,0 %, comparado con el mejor rendimiento previo de 65,4 % obtenido por un sistema basado en *modelos ocultos de markov* [51]. De hecho, ha sido demostrado que es posible aprender LSTMs para afrontar Partially Observable Markov Decision Process (POMDP)s utilizando métodos evolutivos para el aprendizaje [30].

C.1 EVALUACIÓN DE LA RED

El estado de la celda, s_c, se actualiza basado en su estado actual y cuatro fuentes de entrada: net_c es la entrada misma de la celda, mientras que net_{in}, net_{out} y net_φ son las entradas a las compuertas de entrada, salida y olvido. Consideramos pasos de tiempo discretos $t = 1, 2, \ldots$. Un solo paso involucra la actualización de todas las unidades (evaluación). La activación de las compuertas de entrada, salida y olvido (y^{in}, y^{out} y y^φ respectivamente) se calculan de la siguiente manera:

$$net_{out_j}(t) = \sum_m w_{out_j m} y^m(t-1)\,; \quad y^{out_j}(t) = f_{out_j}(net_{out_j}(t))$$

$$net_{in_j}(t) = \sum_m w_{in_j m} y^m(t-1)\,; \quad y^{in_j}(t) = f_{in_j}(net_{in_j}(t))$$

$$net_{\varphi_j}(t) = \sum_m w_{\varphi_j m} y^m(t-1)\,; \quad y^{\varphi_j}(t) = f_{\varphi_j}(net_{\varphi_j}(t))$$

En esta sección j indexará los bloques de memoria; v indexará las celdas de memoria en el bloque j (con S_j celdas), tal que c_j^v denota la v-ésima celda del j-ésimo bloque; v_{lm} es el peso en la conexión de la unidad m a la unidad l. El índice m indexa todas las unidades de origen, como se especifica en la topología de la red (si la activación de una unidad fuente $y^m(t-1)$ se refiere a una unidad de entrada, la entrada externa actual $y^m(t)$ es usada). Para las compuertas, f es la sigmoide logística (en el intervalo $[0, 1]$):

$$f(x) = \frac{1}{1+e^{-x}}$$

La entrada a la celda es

$$net_{c_j^v}(t) = \sum_m w_{c_j^v m} y^m(t-1)$$

que es *escalada* por g, una *función logística centrada* acotada en el intervalo cerrado $[-2, 2]$:

$$g(x) = \frac{4}{1+e^{-x}} - 2$$

El estado interno de la celda de memoria $s_c(t)$ es calculado sumando la entrada regulada (multiplicada) por la compuerta de entrada al estado en el paso de tiempo anterior $s_c(t-1)$, regulado por la compuerta de olvido:

$$s_{c_j^v}(0) = 0$$
$$s_{c_j^v}(t) = y^{\varphi_j} s_{c_j^v}(t-1) + y^{in_j}(t) g(net_{c_j^v}(t)) \quad para\ t > 0$$

La salida de la celda y^c es calculada escalando el estado interno s_c a través de la función h, y luego es regulada (multiplicada) por la activación de la compuerta de salida y^{out}:

$$y^{c_j^v}(t) = y^{out_j}(t) h(s_{c_j^v}(t))$$

h es una sigmoide centrada en el intervalo cerrado $[-1, 1]$:

$$h(x) = \frac{2}{1 + e^{-x}} - 1$$

Finalmente, asumiendo una topología de la red con una capa estándar de entrada, una capa oculta que consiste de bloques de memoria, y una capa de salida estándar, las ecuaciones para las unidades de salida k son:

$$net_k(t) = \sum_m w_{km} y^m(t-1), \quad y^k(t) = h_k(net_k(t))$$

donde m indexa todas las unidades que alimentan a las unidades de salida (típicamente todas las celdas de la capa oculta, pero no las compuertas del bloque de memoria). Como función sigmoide h_k utilizamos nuevamente la función h.

D

ALGORITMO GENÉTICO ECLÉCTICO

Este algoritmo utiliza selección determinista, cruzamiento anular, mutación uniforme y elitismo del tamaño de la población. La naturaleza probabilística de EGA está restringida a los parámetros de probabilidad de cruza y probabilidad de mutación. En EGA la evasión de convergencia prematura es lograda por una estrategia de dos componentes. Primero, los $2n$ individuos de las dos últimas generaciones son ordenados del mejor al menor y sólo a los mejores n se les permite sobrevivir. Luego los individuos son elegidos determinísticamente para cruzar el mejor con el peor (1 con n), el segundo mejor con el segundo peor (2 con $n-1$), etc. De esta manera, se generan n individuos nuevos. Por lo tanto, los mejores esquemas encontrados se retienen en todo momento, mientras que los esquemas que aparecen constantemente se mezclan con los que son más diferentes a ellos. A medida que el algoritmo progresa, los individuos que sobreviven se convierten en la élite de tamaño n de todo el proceso. La cruza anular (equivalente a la cruza en dos puntos) se prefiere sobre los esquemas de cruza uniforme o de un solo punto porque es menos disruptiva que la cruza uniforme, mientras que al mismo tiempo hace al proceso menos dependiente de una codificación específica que la cruza de un solo punto. Este algoritmo fue reportado por primera vez en [45] e incluía autoadaptación y mutación cataclísmica periódica. Sin embargo, estudios posteriores [43] mostraron que ninguno de los dos mecanismos era necesario en la vasta mayoría de los casos y han sido abandonados por razones prácticas. EGA es relativamente simple, rápido y fácil de programar.

D.1 PSEUDOCÓDIGO DE EGA

[Número de mutaciones esperado por generación] $B2M \leftarrow \lceil nL \cdot P_M \rceil$
Generar una población aleatoria
[Evaluar la población]
for $i = 1$ to G
  [Duplicar la población]
  for $j = 1$ to n
    $I(n+j) \leftarrow I(j)$
    $aptitud(n+j) \leftarrow aptitud(j)$

[Selección determinística para cruza anular]
for $j = 1$ to $\frac{n}{2}$
  Generar un número aleatorio $0 \leqslant \rho \leqslant 1$
  if $\rho \leqslant P_c$
    Generar un número aleatorio $0 \leqslant \rho \leqslant \frac{L}{2}$
    Intercambiar el segmento en ρ entre $I(j)$ y $I(n-j-1)$
[Mutación]
for $j = 1$ to B2M
  Generar un números aleatorios uniformes $0 \leqslant \rho_1, \rho_2 \leqslant 1$
  Mutar bit $\lceil \rho_2 L \rceil$ de $I(\lceil \rho_1 n \rceil)$
[Evaluar los nuevos individuos]
Calcular $aptitud(x_i)$ para $i = 1, \ldots, n$
[Selección $\mu + \lambda$]
Ordenar los $2n$ individuos por su aptitud, en forma ascendente

Donde:
$G \equiv$ número de generaciones
$n \equiv$ número de individuos
$I(n) \equiv$ el n-esimo individuo
$L \equiv$ longitud del cromosoma
$P_C \equiv$ probabilidad de cruza
$P_M \equiv$ probabilidad de mutación

E

RANDOM MUTATION HILL-CLIMBER

RMH es un algoritmo de optimización estocástico y local (en contraste con la optimización global). Es una extensión de algoritmos de ascenso determinísticos tal como *Simple Hill Climbing (first-best neighbor)*, *Steepest-Ascent Hill Climbing (best neighbor)*, y padre de técnicas como *Parallel Hill Climbing* y *Random-Restart Hill Climbing*. La estrategia de RMH es iterar el proceso de elegir aleatoriamente un vecino para una solución candidato y sólo aceptarla si resulta en una mejora. Este método fue propuesto para sobreponerse a las limitaciones de ascenso determinísticas que tenían una alta probabilidad de atorarse en óptimos locales debido a su aceptación avara de candidatos vecinos. Se ha reportado [55] que RMH ha tenido mejor rendimiento que el algoritmo genético canónico elitista en algunas funciones (llamadas *Royal Roads*) diseñadas para exhibir algunas de las características más favorables del algoritmo genético canónico elitista.

E.1 PSEUDOCÓDIGO DE RMH

```
actual ← SolucionAleatoria(L)
for j = 1 to G · n
  candidato ← VecinoAleatorio(actual)
  if aptitud(candidato) < aptitud(actual)
    actual ← candidato
retornar actual
```

Donde:
G ≡ número de generaciones
n ≡ número de individuos
L ≡ longitud del cromosoma

F

NONDOMINATIED SORTING GENETIC ALGORITHM II

La presencia de objetivos múltiples en un problema permite la creación de un conjunto de soluciones óptimas (conocido como soluciones óptimas de Pareto), en vez de una sola solución óptima. En la auscencia de más información, no es posible decidir si una de esta soluciones óptimas de Pareto es mejor que otra. Esto obliga al usuario a encontrar tantas soluciones óptimas de Pareto como sea posible. A lo largo de las últimas dos décadas, algunos algoritmos evolutivos multiobjetivo han sido sugeridos para afrontar este problema. Estos algoritmos tienen la habilidad de encontrar varias soluciones óptimas de Pareto en una sola ejecución. Debido a que los algoritmos evolutivos trabajan con una población de soluciones, un algoritmo evolutivo puede ser extendido para mantener un conjunto diverso de soluciones. Nondominatied Sorting Genetic Algorithm (NSGA), propuesto en [81], fue uno de los primeros de este tipo de algoritmos evolutivos. Las críticas principales al método de NSGA han sido las siguientes:

ALTA COMPLEJIDAD COMPUTACIONAL PARA EL ORDENAMIENTO: El algoritmo de ordenamiento no dominado tiene una complejidad de $O(MN^3)$ (donde M es el número de objetivos y N es el tamaño de la población). Esto vuelve a NSGA computacionalmente caro para tamaños de población grandes.

FALTA DE ELITISMO: Algunos resultados recientes [90, 72] muestran que el elitismo puede acelerar el rendimiento del algoritmo genético considerablemente, y también puede ayudar a prevenir la pérdida de buenas soluciones una vez que son encontradas.

NECESIDAD DE ESPECIFICAR EL PARÁMETRO σ_{share}: Los mecanismos tradicionales para asegurar la diversidad en una población con el fin de obtener una variedad amplia de soluciones equivalentes han dependido en su mayoría del concepto de *sharing*. Aunque existen trabajos para determinación dinámica del tamaño del parámetro de *sharing* [21], un mecanismo de preservación de la diversidad sin parámetros es deseable.

NSGA-II busca antender a todos estas críticas y proponer una versión mejorada de NSGA. Este nuevo algoritmo muestra mejor rendi-

miento que dos algoritmos genéticos multiobjetivo contemporaneos [17], Pareto-Archived Evolution Strategy (PAES) [40] y Strength Pareto Evolutionary Algorithm (SPEA) [89].

F.1 ORDENAMIENTO NO DOMINADO

La manera ingenua de abordar el problema del ordenamiento, para identificar soluciones del primer frente no dominado en una población de tamaño N, es comparar cada solución con cada otra solución en la población para averiguar si es dominada. Esto requiere $O(MN)$ comparaciones para cada solución, donde M es el número de objetivos. Cuando este proceso se continua para encontrar todos los miembros del primer frente no dominado en la población, la complejidad total es $O(MN^2)$. En esta etapa del algoritmo, todos los individuos en el primer frente no dominado son encontrados. Para encontrar los individuos en el siguiente frente no dominado, las soluciones del primer frente se descartan temporalmente y el proceso anterior se repite. En el peor de los casos, la tarea de encontrar el segundo frente también requiere tiempo $O(MN^2)$, particularmente cuando un número $O(N)$ de las soluciones pertenecen a los siguientes frentes no dominados. De esta manera, podemos observar que el peor caso es cuando hay N frentes y existe solo una solución en cada frente. Este proceso de ordenamiento requiere tiempo $O(MN^3)$ en total. Debe hacerse notar que $O(N)$ espacio de almacenamiento es requerido para este procedimiento. A continuación se describe un algoritmo de ordenamiento rápido, que requiere tiempo $O(MN^2)$.

Primero, por cada solución se calculan dos entidades:

1. La *cuenta de dominación* n_p, que es el número de soluciones que dominan a la solución p.
2. El *conjunto de soluciones* S_p que la solución p domina. Obtener este conjunto requiere $O(MN^2)$ comparaciones.

Todas las soluciones en el primer frente no dominado tienen su cuenta de dominación establecida como cero. Por cada solución p con $n_p = 0$, visitamos cada miembro q de su conjunto S_p y reducimos su cuenta de dominación en uno. Al hacer lo anterior, si la cuenta de dominación para cualquier miembro q se vuelve cero, lo agregamos a una lista Q. Estos miembros pertenecen al segundo frente no dominado. El proceso anterior se repite para cada miembro de Q, logrando identificar el tercer frente. El proceso continua de esta manera hasta que todos los frentes son identificados.

Para cada solución p que no esté en el primer frente, la cuenta de dominación n_p puede ser a lo más $N - 1$. Por lo tanto, cada solución p será visitada a lo más $N - 1$ veces antes de que su cuenta de dominación sea cero. En este punto, se le asigna a la solución un nivel de no dominación y nunca será visitada nuevamente. Debido a que hay

a lo más $N-1$ de estas soluciones, la complejidad total es $O(N^2)$, y la complejidad de todo el procedimiento es $O(MN^2)$. Es importante notar que aunque la complejidad del algoritmo rápido se ha reducido con respecto al algoritmo ingenuo, el espacio de almacenamiento ha aumentado a $O(N^2)$.

El pseudocódigo para algoritmo rápido de ordenamiento no dominado es el siguiente:

```
for each p ∈ P
  S_p = ∅
  n_p = 0
  for each q ∈ P
    if p ≺ q
      S_p = S_p ∪ {q}
    else if q ≺ p
      n_p = n_p + 1
  if n_p == 0
    p_rank = 1
    F_1 = F_1 ∪ {p}
i = 1
while F_i ≠ ∅
  Q = ∅
  for each p ∈ F_i
    for each q ∈ S_p
      n_q = n_q − 1
      if n_q == 0
        q_rank = i + 1
        Q = Q ∪ {q}
  i = i + 1
  F_i = Q
```

Donde:
$P \equiv$ población de soluciones
$S_p \equiv$ conjunto de soluciones dominadas por p
$n_p \equiv$ número de soluciones que dominan a p
$p_{rank} \equiv$ frente en donde se encuentra p
$\mathcal{F}_i \equiv$ conjunto de soluciones que pertenecen al frente i
$Q \equiv$ conjunto que almacena las soluciones del siguiente frente

F.2 PRESERVACIÓN DE LA DIVERSIDAD

NSGA-II utiliza la métrica de *crowding*, que no requiere que el usuario defina algún parámetro para mantener la diversidad entre los miembros de la población, y también tiene mejor complejidad computacional que el método utilizado en NSGA. Para describir este enfoque,

primero se define una métrica de estimación de densidad y después se presenta el operador de comparación de *crowding*.

ESTIMACIÓN DE DENSIDAD: Para obtener un estimado de la densidad de las soluciones que rodean una solución en particular, calculamos la distancia promedio entre los dos puntos en cada lado del punto actual a lo largo de cada objetivo. La cantidad i_{dist} sirve como un estimado del perímetro del cuboide formado al usar los vecinos más cercanos como los vértices (llamada distancia de *crowding*). En la (fig 1), la distancia de *crowding* de la i-ésima solución en su frente (designada con círculos sólidos) es el promedio de la longitud de los lados del cuboide (ilustrado en una caja punteada).

El cálculo de la distancia del *crowding* requiere ordenar la población de acuerdo con el valor de cada función objetivo en orden ascendente de magnitud. Por lo tanto, las soluciones que están en el límite (soluciones con los valores más grades o más pequeños) se les asigna un valor de distancia infinito por cada función objetivo. A todas las demás soluciones intermedias se les asigna un valor de distancia igual a la diferencia absoluta normalizada en los valores de la función de dos soluciones adyacentes. Este cálculo se continua con otras funciones objetivos. El valor global de la distancia de *crowding* se calcula como la suma de los valores individuales de distancia que corresponden a cada objetivo. Cada función objetivo es normalizada antes de calcular la distancia *crowding*. La complejidad de este procedimiento está gobernada por el algoritmo de ordenamiento. Dado que M ordenamientos independientes de a lo más N soluciones (cuando todos los miembros de la población están en el mismo frente $\mathcal{I}$) están involucrados, el algoritmo tiene una complejidad de $O(MN \log n)$. A continuación se presenta el pseudocódigo para la asignación de la distancia de *crowding*:

$l = |\mathcal{I}|$

for each i, set $\mathcal{I}[i]_{dist} = 0$

for each objective m

  $\mathcal{I} = \text{sort}(\mathcal{I}, m)$

  $\mathcal{I}[1]_{dist} = \mathcal{I}[l]_{dist} = \infty$

  for $i = 2$ to $(l - 1)$

    $\mathcal{I}[i]_{dist} = \mathcal{I}[i]_{dist} + (\mathcal{I}[i+1].m - \mathcal{I}[i-1].m)/(f_m^{max} - f_m^{min})$

Después de que todos los miembros de la población en el conjunto $\mathcal{I}$ tienen una métrica de distancia asignada, es posible comparar dos soluciones por su proximidad a otras soluciones. Una

solución con un valor menos en su métrica de distancia está más atestado por otras soluciones. Esto es exactamente lo que se compara en el operador de comparación de *crowding* propuesto a continuación.

OPERADOR DE COMPARACIÓN DE CROWDING: El operador de comparación de *crowding* ($\prec_n$) guía el proceso de selección en las diferentes etapas del algoritmo hacia un frente óptimo de Pareto uniformemente distribuido. Se presupone que cada individuo i en la población tiene dos atributos:

1. rango de no dominación (i_{rank})
2. distancia de *crowding* ($i_{distance}$)

A partir de estos atributos, definimos un ordenamiento parcial $\prec_n$ como:

$$\begin{aligned} i \prec_n j \quad & \text{if}(i_{rank} < j_{rank}) \\ & \text{or}\,((i_{rank} == j_{rank}) \\ & \text{and}\,(i_{dist} > j_{dist})) \end{aligned}$$

Es decir, entre dos soluciones con diferentes rangos de no dominación, preferimos la solución con el rango menor (mejor). De lo contrario, si ambas soluciones pertenecen al mismo frente, entonces preferimos la solución que está ubicada en una región menos poblada. Una vez que se tiene el ordenamiento anterior, podemos usar cualquier algoritmo genético para optimizar las soluciones en los distintos frentes de Pareto.

BIBLIOGRAFÍA

[1] Adami, Christoph, Charles Ofria y Travis C. Collier: *Evolution of biological complexity*. Proceedings of the National Academy of Sciences, 97(9):4463–4468, 2000. http://www.pnas.org/content/97/9/4463.abstract.

[2] Agre, Philip E. y David Chapman: *Pengi: an implementation of a theory of activity*. En *Proceedings of the sixth National conference on Artificial intelligence - Volume 1*, AAAI'87, páginas 268–272. AAAI Press, 1987, ISBN 0-934613-42-7. http://dl.acm.org/citation.cfm?id=1856670.1856718.

[3] Agre, Philip E. y David Chapman: *What Are Plans for?* En *Robotics and Autonomous Systems*, páginas 17–34. MIT Press, 1989.

[4] Albus, J.S.: *Outline for a theory of intelligence*. Systems, Man and Cybernetics, IEEE Transactions on, 21(3):473–509, 1991, ISSN 0018-9472.

[5] Arkin, R.C.: *Behavior-Based Robotics*. MIT press, 1998.

[6] Arkin, Ronald C.: *Towards the Unification of Navigational Planning and Reactive Control*. En *In AAAI Spring Symposium on Robot Navigation*, páginas 1–5, 1989.

[7] Brooks, Rodney: *Intelligence Without Representation*. Artificial Intelligence, 47:139–159, 1991.

[8] Brooks, Rodney A.: *A Robust Layered Control System For a Mobile Robot*. Informe técnico, Cambridge, MA, USA, 1985. http://www.ncstrl.org:8900/ncstrl/servlet/search?formname=detail&id=oai%3Ancstrlh%3Amitai%3AMIT-AILab%2F%2FAIM-864.

[9] Brooks, Rodney A.: *Elephants don't play chess*. Robotics and Autonomous Systems, 6:3–15, 1990.

[10] Brooks, Rodney A. y Jonathan H. Connell: *Asynchronous Distributed Control System for a Mobile Robot*. En *Storage and Retrieval for Image and Video Databases*, 1986.

[11] Carlos Coello Coello, Gary B. Lamont, David A. van Veldhuizen: *Evolutionary Algorithms for Solving Multi-Objective Problems*. Springer, 2a edición, 2007.

[12] Channon, Alastair: *Passing the ALife Test: Activity Statistics Classify Evolution in Geb as Unbounded*. En Kelemen, Jozef y Petr Sosík (editores): *Advances in Artificial Life*, volumen 2159 de *Lecture Notes in Computer Science*, páginas 417–426. Springer Berlin Heidelberg, 2001, ISBN 978-3-540-42567-0. http://dx.doi.org/10.1007/3-540-44811-X_45.

[13] Choset, Howie M: *Principles of robot motion: theory, algorithms, and implementations*. MIT press, 2005.

[14] Churchland, P.S. y T.J. Sejnowski: *The Computational Brain*. MIT press, Cambridge, MA, USA, 1992.

[15] Cireşan, Dan C., Ueli Meier, Jonathan Masci, Luca M. Gambardella y Jürgen Schmidhuber: *Flexible, High Performance Convolutional Neural Networks for Image Classification*. En *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume Two*, IJCAI'11, páginas 1237–1242. AAAI Press, 2011, ISBN 978-1-57735-514-4. http://dx.doi.org/10.5591/978-1-57735-516-8/IJCAI11-210.

[16] Connell, J.H.: *SSS: a hybrid architecture applied to robot navigation*. En *Robotics and Automation, 1992. Proceedings., 1992 IEEE International Conference on*, páginas 2719–2724 vol.3, 1992.

[17] Deb, K., A Pratap, S. Agarwal y T. Meyarivan: *A fast and elitist multiobjective genetic algorithm: NSGA-II*. Evolutionary Computation, IEEE Transactions on, 6(2):182–197, Apr 2002, ISSN 1089-778X.

[18] Elman, Jeffrey L.: *Finding structure in time*. COGNITIVE SCIENCE, 14(2):179–211, 1990.

[19] Firby, R. James: *An investigation into reactive planning in complex domains*. En *Proceedings of the sixth National conference on Artificial intelligence - Volume 1*, AAAI'87, páginas 202–206. AAAI Press, 1987, ISBN 0-934613-42-7. http://dl.acm.org/citation.cfm?id=1856670.1856706.

[20] Floreano, D. y F. Mondada: *Evolution of homing navigation in a real mobile robot*. Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on, 26(3):396–407, Jun 1996, ISSN 1083-4419.

[21] Fonseca, C.M. y P.J. Fleming: *Multiobjective optimization and multiple constraint handling with evolutionary algorithms. II. Application example*. Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on, 28(1):38–47, Jan 1998, ISSN 1083-4427.

[22] Fukushima, Kunihiko: *Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by*

shift in position. Biological Cybernetics, 36(4):193–202, 1980, ISSN 0340-1200. http://dx.doi.org/10.1007/BF00344251.

[23] Gat, Erann: *Integrating planning and reacting in a heterogeneous asynchronous architecture for controlling real-world mobile robots*. En *Proceedings of the tenth national conference on Artificial intelligence*, AAAI'92, páginas 809–815. AAAI Press, 1992, ISBN 0-262-51063-4. http://dl.acm.org/citation.cfm?id=1867135.1867260.

[24] Georgeff, Michael P. y Amy L. Lansky: *Reactive reasoning and planning*. En *Proceedings of the sixth National conference on Artificial intelligence - Volume 2*, AAAI'87, páginas 677–682. AAAI Press, 1987, ISBN 0-934613-42-7. http://dl.acm.org/citation.cfm?id=1863766.1863818.

[25] Giles, C.L., Dong Chen, Guo Zheng Sun, Hsing Hen Chen, Yee Chung Lee y M.W. Goudreau: *Constructive learning of recurrent neural networks: limitations of recurrent cascade correlation and a simple solution*. Neural Networks, IEEE Transactions on, 6(4):829–836, Jul 1995, ISSN 1045-9227.

[26] Giralt, Georges, Raja Chatila y Marc Vaisset: *An Integrated Navigation and Motion Control System for Autonomous Multisensory Mobile Robots*. En Cox, Ingemar J. y Gordon T. Wilfong (editores): *Autonomous Robot Vehicles*, páginas 420–443. Springer New York, 1990, ISBN 978-1-4613-8999-6. http://dx.doi.org/10.1007/978-1-4613-8997-2_31.

[27] Goldberg, David E. y Jon Richardson: *Genetic Algorithms with Sharing for Multimodal Function Optimization*. En *Proceedings of the Second International Conference on Genetic Algorithms on Genetic Algorithms and Their Application*, páginas 41–49, Hillsdale, NJ, USA, 1987. L. Erlbaum Associates Inc., ISBN 0-8058-0158-8. http://dl.acm.org/citation.cfm?id=42512.42519.

[28] Gomez, Faustino J. y Risto Miikkulainen: *Solving non-Markovian Control Tasks with Neuroevolution*. En *Proceedings of the 16th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'99, páginas 1356–1361, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[29] Gomez, Faustino J. y Risto Miikkulainen: *Transfer of Neuroevolved Controllers in Unstable Domains*. En Deb, Kalyanmoy (editor): *Genetic and Evolutionary Computation - GECCO 2004*, volumen 3103 de *Lecture Notes in Computer Science*, páginas 957–968. Springer Berlin Heidelberg, 2004, ISBN 978-3-540-22343-6. http://dx.doi.org/10.1007/978-3-540-24855-2_108.

[30] Gomez, Faustino J. y Jürgen Schmidhuber: *Co-evolving Recurrent Neurons Learn Deep Memory POMDPs*. En *Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation*, GECCO 2005, páginas 491–498, New York, NY, USA, 2005. ACM, ISBN 1-59593-010-8. http://doi.acm.org/10.1145/1068009.1068092.

[31] González-Banos, Héctor H, David Hsu y Jean Claude Latombe: *Motion planning: Recent developments*. Autonomous Mobile Robots: Sensing, Control, Decision-Making and Applications, 2006.

[32] Gould, Stephen Jay: *Full House: The Spread of Excellence from Plato to Darwin*. Belknap Press, Cambridge, MA, 1a edición, 2011.

[33] Grossberg, S.: *Neural Networks and Natural Intelligence*. MIT press, Cambridge, MA, USA, 1988.

[34] Gruau, Frederic, Darrell Whitley y Larry Pyeatt: *A Comparison between Cellular Encoding and Direct Encoding for Genetic Neural Networks*. En *Genetic Programming 1996: Proceedings of the First Annual Conference*, páginas 81–89. MIT Press, 1996.

[35] Harigopal, U. y H.C. Chen: *Grammatical inference using higher order recurrent neural networks*. En *System Theory, 1993. Proceedings SSST '93., Twenty-Fifth Southeastern Symposium on*, páginas 338–342, Mar 1993.

[36] Haykin, S.: *Neural Networks. A comprehensive foundation*. Prentice Hall, 2a edición, 1998.

[37] Hochreiter, Sepp y Jürgen Schmidhuber: *Long Short-Term Memory*. Neural Comput., 9(8):1735–1780, 1997, ISSN 0899-7667. http://dx.doi.org/10.1162/neco.1997.9.8.1735.

[38] Inman Harvey, Phil Husbands y Dave Cliff: *Seeing the Light: Artificial Evolution, Real Vision*. páginas 392–401. MIT Press/Bradford Books, 1994.

[39] Jong, EdwinD. de: *The Incremental Pareto-Coevolution Archive*. En Deb, Kalyanmoy (editor): *Genetic and Evolutionary Computation GECCO 2004*, volumen 3102 de *Lecture Notes in Computer Science*, páginas 525–536. Springer Berlin Heidelberg, 2004, ISBN 978-3-540-22344-3. http://dx.doi.org/10.1007/978-3-540-24854-5_55.

[40] Knowles, J. y D. Corne: *The Pareto archived evolution strategy: a new baseline algorithm for Pareto multiobjective optimisation*. En *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, volumen 1, páginas –105 Vol. 1, 1999.

[41] Kohonen, Teuvo: *Self-organized formation of topologically correct feature maps*. Biological Cybernetics, 43(1):59–69, 1982, ISSN 0340-1200. http://dx.doi.org/10.1007/BF00337288.

[42] Koutník, Jan, Juergen Schmidhuber y Faustino Gomez: *Evolving Deep Unsupervised Convolutional Networks for Vision-based Reinforcement Learning*. En *Proceedings of the 2014 Conference on Genetic and Evolutionary Computation*, GECCO 2014, páginas 541–548, New York, NY, USA, 2014. ACM, ISBN 978-1-4503-2662-9. http://doi.acm.org/10.1145/2576768.2598358.

[43] Kuri Morales, Ángel: *A Methodology for the Statistical Characterization of Genetic Algorithms*. En Coello Coello, CarlosA., Alvaro de Albornoz, LuisEnrique Sucar y OsvaldoCairó Battistutti (editores): *MICAI 2002: Advances in Artificial Intelligence*, volumen 2313 de *Lecture Notes in Computer Science*, páginas 79–88. Springer Berlin Heidelberg, 2002, ISBN 978-3-540-43475-7. http://dx.doi.org/10.1007/3-540-46016-0_9.

[44] Kuri Morales, Ángel y Edwin Aldana Bobadilla: *The Best Genetic Algorithm I - A Comparative Study of Structurally Different Genetic Algorithms*. En *MICAI (2)*, páginas 1–15, 2013.

[45] Kuri Morales, Ángel y Carlos Villegas Quezada: *A Universal Eclectic Genetic Algorithm for Constrained Optimization*. En *Proceedings of the 6th European Congress on Intelligent Techniques and Soft Computing*, volumen 1, 1998.

[46] Laird, J. y P. Rosenbloom: *An investigation into reactive planning in complex domains*. En *Proceedings of the Ninth National conference on Artificial intelligence*, AAAI'90, páginas 1022–1029. MIT Press, 1990.

[47] Latombe, Jean Claude: *Robot motion planning.: Edition en anglais*. Springer, 1990.

[48] LeCun, Y., L. Bottou, Y. Bengio y P. Haffner: *Gradient-based learning applied to document recognition*. Proceedings of the IEEE, 86(11):2278–2324, Nov 1998, ISSN 0018-9219.

[49] Lehman, Joel y Kenneth O. Stanley: *Abandoning Objectives: Evolution Through the Search for Novelty Alone*. Evol. Comput., 19(2):189–223, 2011, ISSN 1063-6560. http://dx.doi.org/10.1162/EVCO_a_00025.

[50] Lembcke, Scott: *Chipmunk2D 6.2.1 Documentation*, 2013. http://chipmunk-physics.net/release/ChipmunkLatest-Docs/, visitado el 2014-03-13.

[51] Liwicki, Marcus, Alex Graves, Horst Bunke y Jürgen Schmidhuber: *A novel approach to on-line handwriting recognition based on bidirectional long short-term memory networks*. En *In Proceedings of the 9th International Conference on Document Analysis and Recognition, ICDAR 2007*, 2007.

[52] Malcolm, Chris y Tim Smithers: *Symbol grounding via a hybrid architecture in an autonomous assembly system*. Robot. Auton. Syst., 6(1-2):123–144, Junio 1990, ISSN 0921-8890. http://dx.doi.org/10.1016/S0921-8890(05)80032-6.

[53] Maley, C. C.: *Four steps toward open-ended evolution*. En *GECCO-99: Proceedings of the Genetic and Evolutionary Computation Conference*, páginas 1336–1343. Morgan Kaufmann, 1999.

[54] Mataric, Maja J.: *Reinforcement Learning in the Multi-Robot Domain*. Autonomous Robots, 4:73–83, 1997.

[55] Mitchell, Melanie y John H. Holland: *When Will a Genetic Algorithm Outperform Hill Climbing?* En *Proceedings of the 5th International Conference on Genetic Algorithms*, páginas 647– , San Francisco, CA, USA, 1993. Morgan Kaufmann Publishers Inc., ISBN 1-55860-299-2. http://dl.acm.org/citation.cfm?id=645513.657746.

[56] Mitchell, T. M. (editor): *Machine Learning*. McGraw-Hill, New York, NY, 1a edición, 1997.

[57] Moravec, H.P. y A. Elfes: *High resolution maps from wide angle sonar*. En *Robotics and Automation. Proceedings. 1985 IEEE International Conference on*, volumen 2, páginas 116–121, 1985.

[58] N. Jakobi, P. Husbands y I. Harvey: *Noise and the reality gap: The use of simulation in evolutionary robotics*. Advances in Artificial Life: Proceedings of the 3rd European Conference on Artificial Life, páginas 704–720, 1995.

[59] Nelson, AL., E. Grant, G.J. Barlow y T.C. Henderson: *A colony of robots using vision sensing and evolved neural controllers*. En *Intelligent Robots and Systems, 2003. (IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, volumen 3, páginas 2273–2278 vol.3, Oct 2003.

[60] Nelson, Andrew L., Gregory J. Barlow y Lefteris Doitsidis: *Fitness functions in evolutionary robotics: A survey and analysis*. Robotics and Autonomous Systems, 57(4):345 – 370, 2009, ISSN 0921-8890. http://www.sciencedirect.com/science/article/pii/S0921889008001450.

[61] Nilsson, N. J.: *Shakey the Robot, Technical Report*. Informe técnico, 1984.

[62] Nolfi, Stefano y Dario Floreano: *Coevolving Predator and Prey Robots: Do .Arms Races.Arise in Artificial Evolution?* Artif. Life, 4(4):311–335, Octubre 1998, ISSN 1064-5462. http://dx.doi.org/10.1162/106454698568620.

[63] Nolfi, Stefano y Domenico Parisi: *Evolving non-Trivial Behaviors on Real Robots: an Autonomous Robot that Picks up Objects*. En *ROBOTICS AND AUTONOMOUS SYSTEMS*, páginas 187–198. Springer Verlag, 1995.

[64] Nordin, Peter, Wolfgang Banzhaf y Markus Brameier: *Evolution of a World Model for a Miniature Robot using Genetic Programming*. ROBOTICS AND AUTONOMOUS SYSTEMS, 25:105–116, 1998.

[65] Orebäck, Anders y Henrik I. Christensen: *Evaluation of Architectures for Mobile Robotics*. Autonomous Robots, 14(1):33–49, 2003, ISSN 0929-5593. http://dx.doi.org/10.1023/A%3A1020975419546.

[66] Parker, L.E.: *ALLIANCE: an architecture for fault tolerant multirobot cooperation*. Robotics and Automation, IEEE Transactions on, 14(2):220–240, 1998, ISSN 1042-296X.

[67] Pell, Barney, Douglas E. Bernard, Steve A. Chien, Erann Gat, Nicola Muscettola, P. Pandurang Nayak, Michael D. Wagner y Brian C. Williams: *An Autonomous Spacecraft Agent Prototype*. En *Autonomous Robots*, páginas 253–261. ACM Press, 1997.

[68] Pollack, J. B.: *Cascaded back propagation on dynamic connectionist networks*. Technical Report MCCS-86-67, 1986.

[69] Pollack, J. B.: *Language acquisition via strange automata*. En *The Twelfth Annual Conference of the Cognitive Science Society*, páginas 678 – 685. Lawrence Erlbaum Associates, Inc, 1990.

[70] Pollack, JordanB.: *The induction of dynamical recognizers*. Machine Learning, 7(2-3):227–252, 1991, ISSN 0885-6125. http://dx.doi.org/10.1007/BF00114845.

[71] Rosenschein, Stanley J. y Leslie Pack Kaelbling: *A situated view of representation and control*. Artificial Intelligence, 73(1–2):149 – 173, 1995, ISSN 0004-3702. http://www.sciencedirect.com/science/article/pii/0004370294000567, Computational Research on Interaction and Agency, Part 2.

[72] Rudolph, Günter: *Evolutionary Search under Partially Ordered Fitness Sets*. En *IN PROCEEDINGS OF THE INTERNATIONAL SYMPOSIUM ON INFORMATION SCIENCE INNOVATIONS IN ENGINEERING OF NATURAL AND ARTIFICIAL INTELLIGENT SYSTEMS (ISI 2001*, páginas 818–822. ICSC Academic Press, 2001.

[73] Saridis, G.N.: *Intelligent robotic control*. Automatic Control, IEEE Transactions on, 28(5):547–557, 1983, ISSN 0018-9286.

[74] Savage, Jesus, Rodrigo Savage, Marco Morales-Aguirre y Angel Kuri Morales: *Adaptive FPGA-based Robotics State Machine Architecture Derived with Genetic Algorithms*. En *Proceedings of the ACM/-SIGDA International Symposium on Field Programmable Gate Arrays*, FPGA '12, páginas 267–267, New York, NY, USA, 2012. ACM, ISBN 978-1-4503-1155-7.

[75] Scherer, Dominik, Andreas Müller y Sven Behnke: *Evaluation of Pooling Operations in Convolutional Architectures for Object Recognition*. En Diamantaras, Konstantinos, Wlodek Duch y LazarosS. Iliadis (editores): *Artificial Neural Networks - ICANN 2010*, volumen 6354 de *Lecture Notes in Computer Science*, páginas 92–101. Springer Berlin Heidelberg, 2010, ISBN 978-3-642-15824-7. http://dx.doi.org/10.1007/978-3-642-15825-4_10.

[76] Schoppers, M.J.: *Universal Plans for Reactive Robots in Unpredictable Environments*. En *Proceedings International Joint Conference on Artificial Intelligence*, páginas 1039–1046, 1987.

[77] Sharir, Micha: *Algorithmic motion planning in robotics*. Computer, 22(3):9–19, 1989.

[78] Siegelmann, Hava T.: *Neural Networks and Analog Computation: Beyond the Turing Limit*. Birkhäuser, Boston, 1a edición, 1999.

[79] Siegelmann, Hava T. y Eduardo D. Sontag: *On The Computational Power Of Neural Nets*. JOURNAL OF COMPUTER AND SYSTEM SCIENCES, 50(1):132–150, 1995.

[80] Spears, W. M. (editor): *Speciation using tag bits. In Handbook of Evolutionary Computation*. IOP Publishing Ltd., Bristol, UK, UK, 1a edición, 1997, ISBN 0750303921.

[81] Srinivas, N. y Kalyanmoy Deb: *Multiobjective Optimization Using Nondominated Sorting in Genetic Algorithms*. Evolutionary Computation, 2:221–248, 1994.

[82] Standish, Russell K.: *Open-Ended Artificial Evolution*. En *International Journal of Computational Intelligence and Applications*, volumen 167, 2001.

[83] Stanley, Kenneth O. y Risto Miikkulainen: *Evolving Neural Networks Through Augmenting Topologies*. Evolutionary Computation, 10(2):99–127, 2002. http://nn.cs.utexas.edu/?stanley:ec02.

[84] Tani, Jun: *Model-based learning for mobile robot navigation from the dynamical systems perspective*. Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on, 26(3):421–436, Jun 1996, ISSN 1083-4419.

[85] Tsoi, Ah Chung y A. D. Back: *Locally Recurrent Globally Feedforward Networks: A Critical Review of Architectures*. IEEE Transactions on Neural Networks, 5(2):229–239, 1994, ISSN 1045-9227. http://dx.doi.org/10.1109/72.279187.

[86] Watson, R.A, S.G. Ficiei y J.B. Pollack: *Embodied evolution: embodying an evolutionary algorithm in a population of robots*. En *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, volumen 1, páginas –342 Vol. 1, 1999.

[87] Weisstein, Eric W: *Hyperbolic Tangent*. 2011. http://mathworld.wolfram.com/HyperbolicTangent.html.

[88] Ziemke, T.: *Remembering how to behave: Recurrent neural networks for adaptive robot behavior*. CRC Press, 1a edición, 1999.

[89] Zitzler, Eckart: *Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications*, 1999.

[90] Zitzler, Eckart, Kalyanmoy Deb y Lothar Thiele: *Comparison of Multiobjective Evolutionary Algorithms: Empirical Results*. Evol. Comput., 8(2):173–195, Junio 2000, ISSN 1063-6560.