Centro de Investigación y de Estudios
Avanzados
del Instituto Politécnico Nacional

Unidad Zacatenco

Departamento de Computación

Jugador Inteligente de *Backgammon*

Tesis que presenta

Oscar Irineo Fuentes

para obtener el Grado de

Maestro en Ciencias
en Computación

Director de la Tesis: **Francisco Rodríguez Henríquez**

Codirector: **Nareli Cruz Cortés**

México, D.F. **8 de diciembre de 2006**

II

Resumen

Durante muchos años se han realizado diversas investigaciones en el área de la inteligencia artificial, entre las cuales destacan las que han propuesto nuevos mecanismos para tratar de simular la inteligencia humana por medio de computadoras. Entre uno de varios problemas relevantes destacan los juegos de mesa, y en específico para nuestro trabajo de investigación, el juego de *backgammon*.

En esta tesis se ha diseñado un algoritmo genético capaz de aprender y jugar el juego de *backgammon*, únicamente proporcionándole las estrategias básicas del juego. Esencialmente, nuestro algoritmo genético ajusta un conjunto de pesos que se utilizan para decidir qué estrategia es la mejor para determinada posición de las fichas en el tablero.

Existe un evaluador autómata estándar de nivel intermedio llamado Pubeval, con el cual, otros autómatas han competido para comparar su desempeño. El objetivo principal de esta tesis es que nuestro autómata compita en contra de Pubeval y obtenga porcentajes de victorias por encima del 50 %.

Aunque ya existen autómatas para jugar el juego de *backgammon*, no existen reportes en la literatura que utilicen únicamente algoritmos genéticos y reporten resultados en contra de Pubeval, ya que sólo han sido utilizados para probar algunas técnicas de aprendizaje.

Se realizaron varias pruebas, modificando parámetros del algoritmo genético, así como la forma de interpretación de las estrategias, para lograr cumplir con el objetivo. Los resultados comparativos al poner a competir nuestro autómata en contra de Pubeval se reportan al final del documento, mostrando también los resultados de otros autómatas previamente publicados.

Abstract

During many years, several research works have been conducted in the area of artificial intelligence, where new methods have been proposed for attaining some sort of human intelligence simulation through computers. In this respect, one important line of research is board games, and in particular for this work we are interested in studying the backgammon game.

For that matter, in this thesis we designed a genetic algorithm able to learn and play the backgammon game by a continuous refining of a set of basic game strategies. In essence, our genetic algorithm adjusts a set of weights in order to decide which one is the best strategy to follow given a specific board position.

In order to evaluate the performance of any given backgammon automata, it is customary to measure its level by playing several thousand games against Pubeval, an open source publicly available standard benchmark. In this thesis we aimed to produce a player able to obtain winning percentages above 50 % when confronted against Pubeval.

It was not before that we performed many experiments, and that we changed several genetic algorithm parameters as well as the way that the game strategies were interpreted by our automata, that we were able to accomplish our goal. Comparative results when our backgammon player plays against Pubeval are reported in the manuscript. Also, we compare our results against other backgammon players previously reported.

Agradecimientos

Al Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional

Al Consejo Nacional de Ciencia y Tecnología

Al proyecto 45306y del Consejo Nacional de Ciencia y Tecnología

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

En este capítulo se define el problema a resolver, se explica cuál es nuestra motivación para la realización de esta tesis así como el objetivo de la misma.

1.1. Planteamiento del problema

Como veremos en el siguiente capítulo, diversas heurísticas para desarrollar jugadores autómatas inteligentes de juegos de mesa tipo ajedrez, damas, *backgammon* y otros han sido reportadas en la literatura especializada. En particular, el juego de *backgammon* ofrece un reto especial debido a su naturaleza estocástica (se juega con dados) mezclado con estrategias y reglas bien definidas para vencer al oponente.

El juego de *backgammon* se juega entre 2 oponentes, cada uno con 2 dados. En total, existen 36 combinaciones posibles de los dados, pero debido a que en el juego, una combinación de (5,3) es equivalente a (3,5), entonces quitando las parejas repetidas obtenemos un total de 21 posibles combinaciones de los dados. Por cada turno de un jugador, en promedio se pueden realizar 20 movimientos válidos. Por lo general, en cada partida se realizan entre 50 y 60 jugadas. Si tomamos que, en promedio, se realizan 55 jugadas para terminar una partida, entonces podemos obtener en promedio el tamaño del espacio de búsqueda con la siguiente fórmula:

$$(20 * 21)^{55} \approx 1.899^{144}.$$

Como se puede observar, el universo de posibles juegos es extremadamente grande (1.899^{144}), así que el problema a resolver, es encontrar buenas jugadas

en cada turno dependiendo de los valores de los dados para poder ganar el juego. Como se mencionó anteriormente, por cada turno existen en promedio 20 movimiento legales, pero esto no es suficiente para considerar que una jugada es buena, ya que también es necesario tomar en cuenta las posibles jugadas del contrincante y ver que tanto nos pueden afectar en el desempeño del juego. Además también es necesario considerar nuestra siguiente posible jugada, para tener mayor certeza de que el movimiento realizado nos será útil en los siguientes movimientos. Idealmente, sería deseable seguir analizando las siguientes jugadas para así mejorar nuestra confianza sobre cuál es nuestro mejor movimiento, pero esto nos llevaría a explorar el espacio de búsqueda por completo, el cual es demasiado grande e imposible de computar de manera exhaustiva. De acuerdo a esto, si únicamente realizamos el análisis de las posibles jugadas del oponente y de nuestra siguiente jugada, lo cual es lo recomendable desde el punto de vista de la eficiencia, tenemos que el espacio de búsqueda para una buena jugada es

$$20(20 * 21)^2 = 3.528 \times 10^6 \text{ jugadas posibles}$$

en donde, a pesar de haber acortado el espacio de búsqueda a una exploración de la siguiente jugada, se sigue teniendo un espacio de búsqueda grande, por lo que es difícil encontrar una buena jugada de manera exhaustiva. Por lo tanto, se considera que es necesario la aplicación de una técnica que nos proporcione una buena jugada sin necesidad de explorar todo el espacio de búsqueda. En este caso, se intentará resolver el problema utilizando una heurística evolutiva.

1.2. Motivación

Como se ha observado, el espacio de búsqueda existente en el juego de *backgammon* es demasiado grande, haciendo imposible obtener el mejor movimiento por medio de una búsqueda exhaustiva.

En la literatura se han reportado diversas heurísticas para aprender el juego de *backgammon*, las cuales utilizan diversos métodos de inteligencia artificial, tales como redes neuronales, lógica difusa, sistemas de agentes, entre otras, de las cuales, muy pocas utilizan heurísticas evolutivas reportando resultados no demasiado buenos.

Recientemente fue propuesto GP-Gammon [1], el cual es un jugador autómata del juego de *backgammon* que utiliza programación genética. Este autómata ha proporcionado los mejores resultados hasta la fecha al competir en contra del *benchmark* estándar, ***Pubeval***. Debido a esto, en este trabajo de tesis se propone la utilización de una heurística evolutiva para la creación de un autómata para el juego de *backgammon*. Principalmente es con la finalidad de probar qué tan buena puede ser una heurística evolutiva en este ámbito.

1.3. Objetivo

El objetivo general de este trabajo de tesis es crear un autómata, utilizando una heurística evolutiva, el cual sea competitivo en contra de seres humanos, así como contra otros autómatas existentes; principalmente en contra del *benchmark* estándar, *Pubeval*.

El objetivo principal, es llegar a obtener porcentajes superiores al 50 % de victorias al competir contra *Pubeval*.

El resto de esta tesis está organizada de la siguiente manera. En el capítulo 2 se hace referencia a la teoría de juegos, así como a juegos de tablero reportados en la literatura, tales como ajedrez, damas, *othello* y *backgammon*. En el capítulo 3 se describe la historia del *backgammon*, se explican las reglas del juego y se mencionan algunas estrategias simples para obtener un buen desempeño en el juego. En el capítulo 4 se describen algunos conceptos básicos de los algoritmos genéticos, así como el algoritmo genético y sus operadores utilizados. En el capítulo 5 se describe la forma en que se solucionó el problema, las estrategias empleadas y se da la explicación de la forma en que se utilizó un algoritmo genético para hallar una solución con porcentajes de victoria altos. En el capítulo 6 se dan a conocer los experimentos realizados así como los resultados obtenidos. Por último, en el capítulo 7 se explican las principales contribuciones, conclusiones y trabajo futuro de este trabajo de investigación.

Capítulo 2

Antecedentes

En este capítulo se explican algunos conceptos básicos relacionados con la teoría de juegos. Inmediatamente después, se hace mención sobre algunos juegos de tablero, de los cuales se han realizado autómatas computacionales de gran relevancia, como es el caso del ajedrez, el *othello* y las damas, con un enfoque específico en los autómatas diseñados para el juego de *backgammon* siguiendo una estrategia inteligente.

2.1. Teoría de juegos[1]

La teoría de juegos es un área de la matemática aplicada que utiliza modelos para estudiar interacciones en estructuras formalizadas de incentivos (los llamados juegos). Con esto se puede dar solución a un gran número de problemas prácticos (en economía, tácticas militares, y otros campos) en los cuales existen dos o más bandos opuestos, en donde el resultado de cualquier acción de uno de los bandos, depende de las acciones del otro.

Inicialmente la teoría de juegos tuvo sus principales aplicaciones en economía, pero actualmente es aplicada a un gran número de áreas, tales como informática, política, biología y filosofía, entre otras. La teoría de juegos experimentó un crecimiento sustancial y se formalizó por primera vez a partir de los trabajos de John von Neumann y Oskar Morgenstern [5], antes y durante la Guerra Fría, debido sobre todo a su aplicación en estrategias militares.

[1]La mayor parte de la información presentada aquí se puede consultar de una manera más detallada en [2][3][4].

2.1.1. Historia de la teoría de juegos

La primera discusión conocida de la teoría de juegos aparece en una carta escrita por James Waldegrave en 1713. En esta carta, Waldegrave proporciona una solución minimax de estrategia mixta a una versión para dos personas del juego de cartas llamado *le Her*. Sin embargo no se publicó un análisis formal de teoría general de juegos hasta la publicación de *Recherches sur les principes mathématiques de la théorie des richesses*, de Antoine Augustin Cournot en 1838 [6]. En este trabajo, Cournot considera un duopolio y presenta una solución que es una versión restringida del equilibrio de Nash.

Aunque el análisis de Cournot es más general que el de Waldegrave, la teoría de juegos realmente no existió como campo de estudio independiente. Un toerema fundamental para la teoría de juegos es el minimax, el cual fue demostrado por Borel para $n = 3$ y $n = 5$ [7]. John von Neumann publicó una serie de artículos en 1928 en donde generaliza el teorema minimax, adjudicandose ser el creador de la teoría. Estos resultados fueron ampliados más tarde en su libro de 1944, *The Theory of Games and Economic Behavior*, escrito junto con Oskar Morgenstern. Este trabajo contiene un método para encontrar soluciones óptimas para juegos de suma cero de dos personas. Durante este periodo, el trabajo sobre teoría de juegos se centró, sobre todo, en teoría de juegos cooperativos. Este tipo de teoría de juegos analiza las estrategias óptimas para grupos de individuos, suponiendo que pueden establecer acuerdos entre sí acerca de las estrategias más apropiadas.

En 1950, aparecieron las primeras discusiones del dilema del prisionero[2], y se emprendió un experimento acerca de este juego en la corporación RAND. Alrededor de esta misma época, John Nash desarrolló una definición de una estrategia óptima para juegos de múltiples jugadores donde el óptimo no se había definido previamente, conocido como equilibrio de Nash. Este equilibrio es suficientemente general, permitiendo el análisis de juegos no cooperativos además de los juegos cooperativos.

La teoría de juegos experimentó una notable actividad en la década de 1950, momento en el cual los conceptos base, el juego de forma extensiva, el juego ficticio, los juegos repetitivos, y el valor de Shapley[3] fueron desarrollados. Además, en ese tiempo, aparecieron las primeras aplicaciones de la

[2]El problema del dilema del prisionero se describe en la sección 2.1.4

teoría de juegos en la filosofía y las ciencias políticas.

En 1965, Reinhard Selten introdujo su concepto de solución de los equilibrios perfectos del subjuego, que más adelante refinó el equilibrio de Nash. En 1967, John Harsanyi desarrolló los conceptos de la información completa y de los juegos bayesianos. Él, junto con John Nash y Reinhard Selten, ganaron el Premio Nobel de Economía en 1994.

En la década de 1970 la teoría de juegos se aplicó extensamente a la biología, en gran parte como resultado del trabajo de John Maynard Smith y su concepto de estrategia evolutiva estable.

En 2005, los teóricos de juegos Thomas Schelling y Robert Aumann ganaron el premio Nobel de Economía. Schelling trabajó en modelos dinámicos, que son los primeros ejemplos de la teoría de juegos evolutiva. Por su parte, Aumann contribuyó más a la escuela del equilibrio.

2.1.2. Conceptos básicos

En un juego pueden existir dos o más oponentes, por lo que se puede tener juegos para n jugadores. Un **juego** puede definirse como un curso de eventos el cual consiste de una sucesión de acciones por parte de los jugadores. Para que el juego sea susceptible de análisis matemático, también debe tenerse un sistema de reglas establecidas sin ambigüedad, así como el resultado del juego.

Un juego se realiza mediante jugadas sucesivas; cada una de las jugadas puede ser personal o aleatoria. Una jugada **personal** es aquella en la que el jugador está consciente de la acción (elección de una jugada entre un número posible en una situación dada) que realiza y conoce el resultado después de haber realizado la jugada. Un ejemplo de una jugada personal es cualquier movimiento en un juego de ajedrez. Cuando le corresponde su turno, el jugador hace una elección consciente de entre los posibles movimientos, dependiendo de la posición de las piezas sobre el tablero.

[3]Se llama "valor de Shapley" a la asignación que recibe cada jugador en una propuesta de reparto según un criterio de arbitraje diseñado por Lloyd S. Shapley. El criterio consiste en asignar un pago a cada jugador en proporción al número de coaliciones potencialmente vencedoras en las que el jugador participa de forma no redundante.

Una jugada **aleatoria** es la elección de una posibilidad de entre un número de jugadas por medio del resultado de un evento estocástico, en donde el jugador no está consciente de su elección, por ejemplo, el lanzamiento de una moneda al aire o el lanzamiento de los dados.

Algunos juegos contienen únicamente jugadas aleatorias, los cuales reciben el nombre de juegos de "azar". Otros sólo contienen jugadas personales (por ejemplo, damas, ajedrez). Pero también existen juegos mixtos como es el caso de la mayoría de los juegos de cartas y el *backgammon*.

La **estrategia** de un jugador se define como el conjunto completo de las reglas que determinan sus elecciones para todas las situaciones que se presentan en el curso de un juego. Para que el concepto de estrategia tenga algún significado, el juego debe contener jugadas personales, ya que no existen estrategias para juegos de azar puros.

Un juego puede ser **finito** o **infinito**, dependiendo del número de estrategias posibles. En un juego finito, cada jugador tiene un número finito de estrategias posibles.

2.1.3. Tipos de juegos

Los juegos se pueden clasificar de acuerdo a los métodos que son aplicados para resolverlos, por lo que se tienen las siguientes categorizaciones.

Juegos simétricos y asimétricos

Un juego **simétrico** es un juego donde las recompensas por jugar una estrategia particular dependen solamente de las otras estrategias empleadas, no de quién las juegue. Si se pueden intercambiar las identidades de los jugadores sin cambiar las recompensas, el juego es simétrico. La simetría puede aparecer en diferentes formas. Los juegos **ordinalmente simétricos** son juegos simétricos respecto a la estructura ordinal de las recompensas. Un juego es **cuantitativamente simétrico** si y sólo si es simétrico respecto a las recompensas exactas.

Los juegos **asimétricos** más estudiados son los juegos donde no hay conjuntos de estrategias idénticas para ambos jugadores. No obstante, puede haber juegos asimétricos con estrategias idénticas para cada jugador.

Juegos suma cero

Se dice que un juego es un juego de suma cero si la suma de las recompensas es cero, es decir, si uno de los bandos pierde exactamente tanto como lo que el otro gana. En los juegos de suma cero las metas que persiguen los jugadores son totalmente opuestas.

Juegos cooperativos

Un juego cooperativo se caracteriza por un contrato o acuerdo que puede hacerse cumplir. La teoría de los juegos cooperativos da justificaciones de contratos plausibles. La plausibilidad de un contrato está muy relacionada con la estabilidad.

Juegos simultáneos y secuenciales

Los juegos simultáneos son juegos en los que los jugadores mueven simultáneamente o en los que éstos desconocen los movimientos anteriores de otros jugadores. Los juegos secuenciales (o dinámicos) son juegos en los que los jugadores tienen algún conocimiento de las acciones previas. Este conocimiento no necesariamente tiene que ser perfecto; sólo debe consistir en alguna información. Por ejemplo, un jugador puede conocer que el oponente no realizó una acción determinada, pero sin quizás saber cuál de las otras acciones disponibles eligió.

Juegos de información perfecta

En este tipo de juegos, cada participante, al hacer una jugada, conoce los resultados de todas las jugadas hechas previamente, sean éstas personales o aleatorias. Este tipo de juegos es un subconjunto de los juegos secuenciales.

2.1.4. Representación

Dependiendo del tipo de juego, es conveniente utilizar una representación u otra, ya que ésta será de mayor utilidad para analizar el juego. Sin embargo, se puede dar el caso de juegos que se puedan representar de dos o más formas.

Para explicar los tipos de representación se presenta el problema del *dilema del prisionero*, el cual puede ser formulado como sigue:

La policía arresta a dos sospechosos. No hay pruebas suficientes para condenarles, y tras haberlos separado, visita a cada uno y les ofrece el mismo trato: "Si confiesas y tu cómplice continúa sin hablar, él será condenado a la pena total, 10 años, y tú serás liberado. Si él confiesa y tú callas, tú recibirás esa pena y será él quien salga libre. Si ambos permanecen callados, todo lo que podremos hacer será encerrarlos 6 meses por un cargo menor. Si ambos confiesan, ambos serán condenados a 6 años."

Representación de matriz

Los juegos simultáneos se suelen representar con una matriz, debido a que su análisis es mucho más simple de esta forma. En la matriz se utilizan las filas para las estrategias de el jugador 1 (A) y las columnas para las estrategias del jugador 2 (B). En cada casilla se especifican las recompensas de cada jugador; el primero representa las recompensas de A y el segundo las de B.

En el caso del ejemplo del dilema del prisionero podemos considerar los siguientes casos:

- Si el prisionero confiesa y su cómplice calla, la recompensa es de 0.
- Si el prisionero calla y su cómplice calla, la recompensa es de -0.5.
- Si el prisionero confiesa y su cómplice confiesa, la recompensa es de -6.
- Si el prisionero calla y su cómplice confiesa, la recompensa es de -10.

La matriz de recompensas para este problema se muestra en la figura 2.1

Representación de árbol

Los juegos secuenciales suelen representarse por medio de árboles, ya que un jugador tiene la posibilidad de elegir sus estrategias dependiendo de la elección de su oponente, en donde cada vértice o nodo representa un punto donde el jugador toma decisiones. El jugador se especifica por un número o letra situado junto al vértice. Las líneas que parten del vértice representan acciones posibles para el jugador. Las recompensas se especifican en las terminaciones de las ramas del árbol (hojas).

		B: Confiesa	B: Calla
A	Confiesa	–6,–6	0,–10
	Calla	–10,0	–0.5,–0.5

Figura 2.1: Matriz de recompensas para el juego del *dilema del prisionero*

Para representar el ejemplo del *dilema del prisionero*, en este caso se asume que el segundo prisionero conoce la respuesta elegida del primer prisionero y tiene la posibilidad de elegir a partir de la respuesta del otro. La figura 2.2 representa el árbol para el juego del dilema del prisionero.

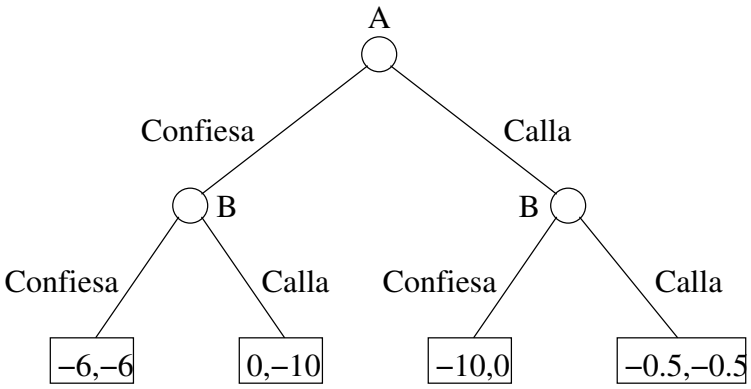

Figura 2.2: Árbol para el juego del *dilema del prisionero*

2.1.5. Métodos para la solución de juegos

Existen diversos métodos para dar una solución aceptable a los juegos, debido a la dificultad para encontrar una solución óptima. Existen métodos para resolver juegos sencillos y de pocas estrategias los cuales pueden ser calculados manualmente. También existen métodos que sirven para resolver

juegos con un gran número de estrategias, los cuales se resuelven principalmente por medio de computadoras.

Si un juego se repite muchas veces, una estrategia óptima para un jugador es una estrategia que le garantiza la ganancia media máxima posible (o lo que es lo mismo, la pérdida media menor posible). Hay ocasiones en las que la misma estrategia no nos lleva siempre a la mayor ganancia media, por lo que es necesario emplear estrategias mixtas, con un porcentaje de utilización para cada una de ellas durante el juego.

Método minimax

En este método lo que se hace es observar todas las recompensas que podemos obtener con cada estrategia y por cada estrategia, encontrar la mínima recompensa. Una vez encontradas todas las mínimas recompensas para cada estrategia, tomamos a la mayor de ellas, siendo ésta la que utilizaremos, ya que nos garantiza que al utilizar esta estrategia nuestra recompensa no será menor al valor indicado. Para saber lo que a nuestro oponente le conviene decidir se hace el análisis inverso, es decir, se obtienen las máximas recompensas de cada estrategia y al final se toma la mayor recompensa obtenida. La intersección de las estrategias es la solución al juego.

Para el caso de juego del *dilema del prisionero* se tiene que la solución es confesar siempre, ya que de otra forma se expone algún jugador a obtener una mayor pérdida si es que su cómplice no coopera. En la figura 2.3 se muestra la solución al juego.

Método geométrico

El método geométrico es otra forma de encontrar una solución a los juegos y observar gráficamente cuál es la mejor estrategia. Desafortunadamente este método no es muy conveniente cuando contamos con un gran número de estrategias ya que se vuelve imposible apreciar la solución. Este método se aplica principalmente a juegos de tipo suma cero.

En este método se asocian las estrategias a un eje o plano de nuestra gráfica, es decir, si tenemos 2 estrategias la estrategia A_1 se situa en el punto 0 del eje x y la estrategia A_2 en el punto 1 del eje x. Sobre estos puntos se traza una línea perpendicular A_1A_1 y A_2A_2. Estas líneas nos indicarán las

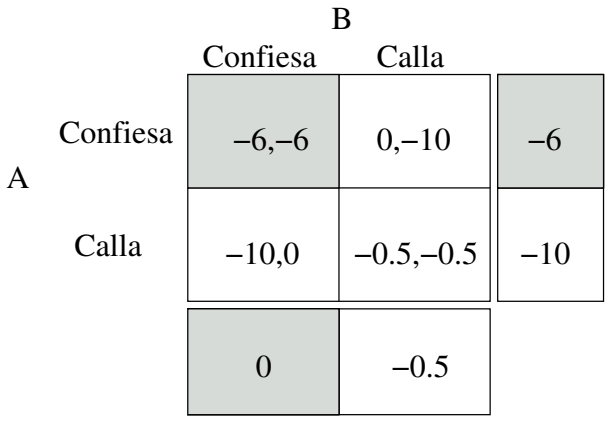

Figura 2.3: Solución al juego del dilema del prisionero por el método minimax

recompensas para cada estrategia de A. Finalmente, se trazan líneas B_1B_1 y B_2B_2 las cuales representan las recompensas del oponente. El límite inferior de la gráfica nos muestra la solución. Ésta es la forma para resolver un juego de 2×2. Para un juego con más estrategias se realiza un procedimiento similar, pero aumentando los planos y las líneas que cruzan, es por eso que se vuelve ilegible con un mayor número de estrategias.

En la figura 2.4 se muestra una matriz de recompensas junto con su solución por medio del método geométrico. Como se puede observar, la solución es utilizar una estrategia mixta. De acuerdo a este método se debe de utilizar dos terceras parte de los juegos una estrategia y la otra estrategia en la tercera parte restante.

Método de programación lineal

Este método se emplea principalmente para juegos con estrategias de $m \times n$. Básicamente consiste en tomar las estrategias que se presentan en una matriz de recompensas (A_{ij}) y pasar estos valores en forma de ecuaciones. En estos casos es seguro que nuestra solución óptima estará formada por estrategias mixtas, así que hay que multiplicar cada valor a_{ij} por una variable p_i, la cual representa el porcentaje de utilización de la estrategia i. Finalmente, cada ecuación se iguala al valor del juego v, éste es la recompensa media esperada al jugar un juego varias veces.

A \ B	B1	B2
A1	1	0
A2	0	$\frac{1}{2}$

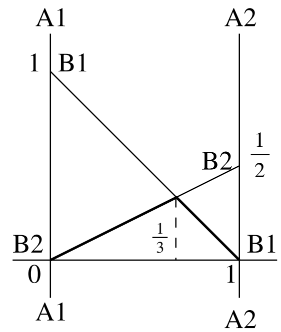

Figura 2.4: Método geométrico

Una vez realizado todo esto, nos queda un sistema de ecuaciones de la siguiente manera:

$$\begin{aligned} a_{11}p_1 + a_{21}p_2 + \ldots + a_{m1}p_m &\geq v \\ a_{12}p_1 + a_{22}p_2 + \ldots + a_{m2}p_m &\geq v \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots & \\ a_{1n}p_1 + a_{2n}p_2 + \ldots + a_{mn}p_m &\geq v \end{aligned}$$

el cual hay que resolver para obtener los valores de p_i, los cuales nos darán la solución al juego.

Estos métodos son los clásicos para resolver problemas de la teoría de juegos, aunque no son los únicos, ya que a veces no es posible dar una solución exacta a algunos de ellos, por lo que existen algunos métodos aproximados para solucionarlos, recientemente surgió la teoría evolutiva de juegos la cual aplica modelos inspirados en genética de poblaciones sobre los problemas. Difiere de la teoría de juegos clásica en que se concentra en las dinámicas de la estrategia en lugar de en sus equilibrios.

2.2. Trabajo previo

Existen diversos juegos de tablero tales como ajedrez, damas, *backgammon*, entre otros, de los cuales se han creado jugadores de computadora para competir contra otros jugadores de computadora o contra humanos, de entre los que se pueden citar los siguientes.

2.2.1. Ajedrez

Deep Blue [8] es un jugador de ajedrez desarrollado por investigadores de la IBM, el que se enfrentó en 1996 y 1997 al campeón mundial de ese entonces, Garry Kasparov. En el primer enfrentamiento que tuvieron salió victorioso Kasparov, pero en el segundo después de algunas mejoras, quien se llevó lo victoria fue Deep Blue. La forma en que Deep Blue funciona es con dos bases de datos, la primera conocida como libro de aperturas (*opening book*) y la segunda como libro extendido (*extended book*). Estas dos bases de datos guardan las mejores jugadas hechas por grandes maestros en los últimos años de práctica profesional en torneos de élite.

2.2.2. *Othello*

En 1997, C. T. Sun [9] desarrolló un juego de *othello*, que usa una combinación de algoritmos genéticos y conjuntos difusos. En su experimento, divide un juego en varios estados representados por un conjunto difuso. Después, cada estado es codificado dentro de un cromosoma, para evolucionarlo y obtener el mejor estado.

2.2.3. Damas

Chinoock [10] es un jugador de damas desarrollado en la Universidad de Alberta en Canada. Al igual que Deep Blue, utiliza dos bases de datos más una función de evaluación para decidir qué tan buena es una posición.

Evolutionary Checkers [11] es otro jugador de damas desarrollado por Kumar Chellapilla y David Fogel en la Universidad de California. Usa cómputo evolutivo en donde los individuos de la población son redes neuronales. La idea de realizar este jugador se debe a que se decía que Deep Blue así como Chinook no podian ser inteligentes ya que sólo hacian consultas a bases de

datos, por lo que decidieron hacer un jugador que aprendiera por sí mismo sin la necesidad de consultar bases de datos.

2.2.4. *Backgammon*

En 1995 G. Tesauro diseñó TD-Gammon [12], un jugador de *backgammon* que juega a un nivel competitivo en contra de humanos. Este programa tiene un aprendizaje exitoso mientras juega, ya que va aprendiendo las estrategias durante el juego. TD-Gammon emplea el método de diferencia temporal[3] para entrenar una red neuronal. La estrategia de diferencia temporal básicamente se encarga de ajustar los pesos de la red neuronal de acuerdo a los resultados obtenidos en una secuencia de juegos. TD-Gammon consiste de dos redes neuronales, una para cada fase del juego. Desafortunadamente, debido a que es una tecnología propietaria, TD-Gammon no puede ser usado para compararlo con otros enfoques. Sin embargo, existe un evaluador disponible llamado Pubeval [13], el cual también fue creado por G. Tesauro. Con Pubeval es posible crear jugadores autómatas de *backgammon* con un nivel intermedio. Pubeval es hasta el momento un evaluador estándar (*benchmark*) para el juego de *backgammon*.

En 1997, Pollack et al. [14] publicaron un artículo de un jugador autómata de *backgammon* llamado HC-Gammon, el cual usa la técnica de *Hill-Climbing* para entrenar una red neuronal. HC-Gammon inicia con una red con todos los pesos en cero y juega los siguiente juegos tomando los pesos del actual campeón para seguir jugando. Usando esta técnica, Pollack et al. concluyeron que el éxito de TD-Gammon se debía principalmente a la estructura de la co-evolución de la tarea de aprendizaje y las características del juego de *backgammon* y en un menor grado a la técnica sofisticada de aprendizaje empleada. Sin embargo, G. Tesauro refutó estas observaciones en un artículo posterior [15]. Las bajos porcentajes de victorias de este autómata se deben a que la técnica usada no es muy buena, ya que por lo general puede converger a óptimos locales.

ACT-R-Gammon (2000) [16]. La propuesta de Sanner et al. fue desarrollar un algoritmo que simule el proceso de aprendizaje humano en el dominio del *backgammon*. Ellos argumentaban que los humanos no pueden explorar

[3]El método de diferencia temporal se describe en la sección 5.4.1

miles de movimientos posibles como lo hace un programa de computadora. Por lo tanto, se deben usar métodos más eficientes para el aprendizaje. Basado en la teoría del conocimiento ACT-R (Adaptive Control of Thought-Rational, 1976), su programa ACT-R-Gammon analiza características generales del *backgammon* que pueden ser encontradas en los juegos ganados o perdidos. Emplean aprendizaje bayesiano para inferir la probabilidad de que cada una de las características resultantes de un movimiento puedan estar presentes en un juego ganado y utiliza correspondencia parcial (partial matching) sobre algunas posiciones del tablero. ACT-R-Gammon tiene un porcentaje de victorias de alrededor del 46 % en contra de Pubeval después de haber jugado 1,000 juegos. El bajo porcentaje de victorias obtenidas es debido a las pocas características de la teoría del conocimiento ACT-R empleadas, posiblemente si se emplean más características de esta teoría, se podría mejorar el desempeño.

GMARLB [17]. Otro enfoque interesante es el propuesto por Qi y Sun en el 2003. Su trabajo consiste en un enfoque híbrido, donde un algoritmo genético es aplicado a un equipo de agentes cuyas capacidades de aprendizaje están basadas en redes neuronales, llamado GMARLB (*Genetic algorithm based Multi Agent Reinforcement Learning Bidding*). El aprendizaje por reforzamiento realizado por el sistema de multiagentes es aplicado al juego de *backgammon*. La idea general de este enfoque es que los agentes son creados dentro de un equipo usando dos módulos de decisión, el módulo Q y el módulo CQ. Un único agente controlador en el equipo es el responsable de tomar las acciones adecuadas en cualquier momento durante el juego. Usando el módulo Q, el agente controlador selecciona la acción a ser realizada, mientras que el módulo CQ determina si el agente debe seguir con el control o ceder el control a otro agente. Una vez que un agente cede el control, un nuevo agente es seleccionado por medio de una oferta. Las capacidades de aprendizaje del módulo Q como del módulo CQ están basadas en redes neuronales. En [17] se reporta que este enfoque obtiene porcentajes de victoria del 56 % en contra de Pubeval depués de 400,000 juegos de entrenamiento. Sin embargo, una prueba realizada independientemente en [1] muestra que el porcentaje tan elevado se debe aparentemente a que la medida tomada es muy pequeña, ya que se realizó con un torneo de 50 juegos. De acuerdo a resultados experimentales independientes se muestra que el porcentaje de victorias de GMARLB se reduce al 51.2 % cuando juega en contra de Pubeval durante 1,000 juegos. Posiblemente el uso de agentes permita una mayor di-

versidad de opiniones para la decisión de una jugada y con forme la cantidad de juegos jugados aumenta, se van encontrando los agentes con las mejores decisiones.

Azaria y Sipper propusieron GP-Gammon [1][18][19], un jugador de *backgammon* que utiliza programación genética. GP-Gammon aplica la programación genética a la evolución de estrategias para jugar *backgammon*. Una vez tirados los dados, el programa genera todos los movimientos posibles. Cada individuo recibe una posición del tablero y regresa un número real que representa la puntuación dada al tablero. Finalmente el tablero con mayor puntuación es seleccionado para realizar el movimiento correspondiente. El porcentaje de victorias de GP-Gammon en contra de Pubeval es del 62.4 % de victorias en torneos de a 1000 juegos después de 2,000,000 de juegos de entrenamiento. El uso del lenguaje de programación Lisp y la forma de abstracción de las estrategias se adecuan muy bien a la heurística utilizada y es posible que esto les permita la obtención de buenos resultados.

Recientemente, un autómata de *backgammon* basado en controladores difusos, llamado Fuzzeval, fue propuesto en [20][21]. Fuzzeval, evalúa todos los tableros generados a partir de los movimientos posibles usando una base de reglas obtenida de un humano con un nivel intermedio de juego. Fuzzeval ajusta automáticamente la función de membresía asociada con las variables lingüisticas empleadas en la base de reglas, alineándolas con los valores promedios que fueron usados en el juego anterior. Fuzzeval reporta un porcentaje de victorias del 42 % en contra de Pubeval en torneos de 100 juegos. Debido a que las reglas son proporcionadas por un humano no tan experto en el juego, el desempeño de este autómata no es muy bueno.

Existen otros autómatas para el juego de *backgammon* tales como *Jellyfish* y *Snowie*, los cuales son de tecnología propietaria y no se tiene mucha información sobre éstos. Adicionalmente, como software libre esta *GNU-Backgammon*. Estos tres últimos utilizan redes neuronales y básicamente utilizan la idea de G. Tesauro con algunas modificaciones.

Capítulo 3

Juego de *backgammon*

3.1. Historia del *backgammon*

3.1.1. Orígenes del *backgammon*

Este juego de mesa es considerado, hasta donde se tiene conocimiento, el más antiguo de la historia. Se estima que se inventó hace más de 5 mil años en Mesopotamia por el Imperio Persa (actualmente Irán, Irak y Siria). Entre los años 1922 y 1934 el arqueólogo Leonard Woolley descubrió en tumbas de los Sumerios que se hacía mención a un tablero de juegos que se cree que es el más antiguo juego de *backgammon* preservado hasta nuestros días. En estas mismas investigaciones se hallaron otros cuatro juegos. Otra versión sobre el origen del backgammon indica que nació en Persia hace más de mil 600 años y que fue introducido en el continente europeo por los árabes. El juego transcurría en superficies planas, generalmente de madera, se usaban piedras en lugar de fichas y los dados eran hechos de huesos, madera, piedra o cerámica.

Los Faraones egipcios se divertían con un juego parecido al *backgammon*. Estos juegos han sido hallados en tumbas egipcias y también en jeroglíficos murales. Los egipcios usaban una cajita para rodar los dados y así eliminaban el engaño. Los griegos y los romanos también usaban una cajita para sus dados en sus respectivas versiones del juego.

3.1.2. Artefactos utilizados

A lo largo de la historia el juego ha sido asociado a los líderes y aristocracia de estas civilizaciones antiguas, tal como lo demuestran las reliquias y literatura recogidas en excavaciones arqueológicas hechas en Persia, Grecia, Roma y el Lejano Oriente. En Egipto fueron hallados tableros de juego de 3×10, 3×12 y 3×6, conocidos como el "Juego de los 30 cuadrados" o "Senat".

Estos artefactos datan de los años 3000 al 1788 AC y las reglas, así como los dados que se utilizaban son desconocidos. En la tumba real de Ur al Chaldees fueron encontrados tableros de madera, en el centro del reino Sumerio, los cuales datan del 2600 AC. Estos tableros fueron encontrados junto a dados tetraédricos y son conocidos como "Los Juegos Reales de Ur". Un conjunto de reglas para una versión del juego de esa época fue encontrado en una tableta cuneiforme que data del año 177 AC.

3.1.3. *Backgammon* Romano

Los romanos dejaron evidencia de un juego llamado *Ludus Duodecim Scriptorum*, "El Juego de las 12 Líneas", que utilizaba tableros de cuero y conjuntos de 30 fichas (15 de ébano y 15 de marfil) y data del año 600 AC. Se cree que es una derivación del juego egipcio "Senat". En el siglo 1 DC, este juego fue reemplazado por una variante de 2×12 líneas en lugar de las 3×12 de antes, acercándose más y más a la versión actual.

El juego llegó a Inglaterra con la conquista romana del siglo 1 DC, siendo conocido como "Tabola", nombre genérico que se le daba al tablero en que se jugaba el juego. Este pasatiempo era muy popular y era el juego favorito del Emperador Claudio. Alrededor del año 50 DC, Claudio escribió la historia del juego "Tabola" que, desafortunadamente, no llegó a nuestros días. Su carroza imperial estaba equipada con una cama y un juego de "Tabola" para poder jugar mientras viajaba.

El "Tabola" fue también el juego que causó las adicciones al juego que inundaron Roma antes de que fuese declarado ilegal por la República. La multa por jugar en cualquier momento excepto durante la "Saturnalia" era de cuatro veces la apuesta, aunque esta ley era débil y rara vez aplicada. En el siglo 6 DC el juego fue llamado "Alea" (El Arte de Apostar con Dados). El

"Alea" es el precursor directo del juego actual de *backgammon*, aunque hubo muchas variantes respecto a las posiciones de las fichas y los movimientos.

3.1.4. El *backgammon* en Asia

En Asia, un juego llamado "Nard" apareció antes del 800 DC en el sudoeste de Asia y Persia. El "Nard" se jugaba de forma similar al "Alea" y usaba sólo un dado para mover las fichas. También fue conocido como "Nardshir", "Nardeeshir", "Nard-i-shir" y "Takhteh Nard" que significa "Batalla en Madera".

Un texto antiguo descubierto reveló el simbolismo del juego:

- El tablero representa un año
- Cada lado tiene 12 puntas por cada mes del año
- Las 24 puntas representan las horas en un día
- Las 30 fichas representan los días del mes
- La suma de los opuestos del dado es 7 y representa los días de la semana
- Los colores de las fichas y las puntas del tablero, representan el día y la noche.

"T'shu-p'u" era el nombre del "Nard" en China a pesar de que fue inventado al oeste de la India y que llegó a China durante la dinastía Wei (220-250 DC) y fue popular durante el 479-1000 DC, mientras que en Japón se llamó "Sugoroku" y se declaró ilegal durante el reinado de la Emperatriz Jito (690 - 697 D.C.).

El "Nard" fue introducido en Europa a través de Italia, debido a las invasiones árabes en Sicilia en el año 902 DC. El término "Tabola" fue usado por varias culturas, haciendo que la difusión del juego en Europa fuese amplia, mientras que el "Nard" sufrió una difusión similar en Asia gracias a los árabes.

El juego "Nard" parece ser una leve variante del "Tabola", que incorporó aspectos del "Senat". La principal diferencia era que el "Tabola" utilizaba tres dados y el "Nard" dos; el uso de dos dados en el "Tabola" se volvió después muy popular.

3.1.5. Proliferación y estandarización del *backgammon*

La primera mención en inglés fue en "*The Codex Exoniensis*". En 1025 el "Nard" o "Tables" (como era llamado en inglés) era muy popular en las tabernas inglesas. El ajedrez superó al "Tables" en popularidad en el siglo 15 DC. Las primeras reglas que se conocen fueron escritas en el Libro de Juegos del Rey Alfonso X (siglo XIII)[22], en donde se especificaban 15 variaciones del *backgammon*. El término *backgammon* empezó a ser utilizado en 1645. Es tema de debate si el término Backgammon deriva del galés 'back' (pequeña) y 'gammon' (batalla) o del sajón 'back' (de regreso) y 'gamen' (juego), debido a la facultad de las fichas de volver al tablero de juego luego de ser comidas. En 1743, Edmond Hoyle escribió las reglas del juego de *backgammon*, que luego fueron modificadas en Estados Unidos en el siglo XX (1931) Aunque fue durante las décadas de los 60 y 70 cuando se popularizó de manera masiva.

Como muchos juegos jugados por dinero, se volvió impopular para las autoridades de Inglaterra y, hasta el reinado de Isabel I, las leyes que prohibían jugar "Tables" en establecimientos autorizados seguían en vigencia. A principios del siglo XVII, sin embargo, siguiendo algunas modificaciones a las reglas, el juego experimentó una renovación y pasó rápidamente por toda Europa otra vez bajo una variedad de nombres diferentes que en su mayoría han permanecido hasta el día de hoy.

3.1.6. Diferentes nombres

A lo largo de la historia se ha conocido al juego de *backgammon* de muchas maneras, debido a las variantes tanto en el juego como en la región donde se jugaba:

- Los egipcios lo llamaban *Senat*
- Los romanos *El juego de las 12 líneas*
- En Inglaterra era llamado *Tables*
- En Italia, *Tavola Reale*
- En España, *Tablas Reales*
- En Grecia aún es llamado *Tavli*

- En Francia, *Le Trictrac*
- En China, *T'shu-p'u*
- En Japón, *Sugoroku*

En estos dos últimos países se juega en tableros circulares. En la actualidad se le conoce con diferente nombre dependiendo del lugar:

- *Backgammon*, inglés
- *Gammon*, escocés
- *Tric-Trac*, francés
- *Puff*; alemán
- *Vrhcáby*, checo

Hay una familia entera de variantes: *Chouette* (versión para 3 ó 4 jugadores), *backgammon* cooperativo, *Sixey-Acey*, *backgammon* holandés, *backgammon* turco (*Moultezim*), *backgammon* griego (*Plakato*), *Gioul* (de Medio Oriente), *Acey Deucey* (versión de las fuerzas de EEUU del *backgammon* holandés), *Acey Deucey* europeo, *backgammon* ruso, *backgammon tabardo* y *backgammon* islandés (*Kotra*).

3.1.7. La historia del *backgammon* moderno

1920-1960

El dado de doblaje se cree fue introducido al juego en Nueva York en 1920 por algún apostador desconocido. De esta forma se elevó el nivel de destreza del juego y su popularidad, volviéndose un pasatiempo muy disfrutado. Era principalmente jugado por la clase alta en clubes privados. Las reglas fueron modificadas en 1931 en los Estados Unidos. Tuvo una leve caída en popularidad cuando ocurrió la gran depresión en los 1930s y un leve resurgimiento en la década de los 1940s. Su popularidad cayó de nuevo durante la Segunda Guerra Mundial.

1960-1990

La popularidad del juego aumentó nuevamente durante 1960 gracias a los esfuerzos del Príncipe Alexis Obelensky, que organizaba y promovía torneos, incluido el Primer Campeonato Oficial que tuvo lugar en las Bahamas y se mantiene hasta el día de hoy. Una gran proliferación de libros sobre *backgammon* tuvo lugar luego de la publicación del libro "Backgammon: El Juego de Acción", del mismo Obelensky. Los 70's son descritos como los años de gloria del *backgammon*, debido a que vio enormes aumentos en su popularidad, publicidad, torneos y literatura referida al *backgammon*, que incluía libros, revistas y columnas en periódicos. Se movió de la clase alta a la media y se convirtió en el pasatiempo popular de los jóvenes. En los torneos se volvieron comunes las cifras de seis dígitos en las apuestas y su popularidad se extendió por todos los Estados Unidos y Europa. En los 80's, sin embargo, se dio un declive en su popularidad, principalmente entre los jóvenes, debido posiblemente al surgimiento de los video juegos y toda la emoción que éstos proporcionaban. El interés por aprender a mejorar su juego entre los fieles del *backgammon* se volvió aún más fuerte debido a la invención del *backgammon* para computadora, que no sólo daba la oportunidad de jugar en contra de un rival con habilidades similares sino que también daba la oportunidad de ahorrar horas en tiro de dados.

1990 al día de hoy

En 1995, Gerald Tesauro, desarrollador de IBM, escribió un software que se enseña a sí mismo a jugar al *backgammon*, usando una red neuronal, con lo que logró obtener un jugador de primera clase llamado "TD-Gammon"[12].

El "FIBS" (Primer Servidor de *Backgammon* en Internet, por sus siglas en inglés) fue creado en 1993 por Andreas Schneider y subido a un servidor universitario en Suecia de forma gratuita. Más de 100 jugadores con conexión a Internet podían ser encontrados jugando a toda hora, ya que el sistema tenía la habilidad de guardar los juegos en transcurso, reproducir juegos ya jugados y comparar jugadores mediante cálculo de promedios. Frederic Dahl, de Noruega creó la primera red neuronal comercial de software de *backgammon*, con el llamado "Jellyfish".

Egger introdujo "Snowie", una versión más popular y comercial de software de *backgammon*, que tenía una interfaz más amigable para el usuario

y la habilidad de comparar partidas consideradas partidas modelo aun hoy en día. Finalmente cabe mencionar al "GNU-Backgammon", el cual supera a los anteriores debido a su código fuente abierto, que lo hace gratuito y con la posibilidad de que los programadores lo mejoren o modifiquen a gusto.

Esta información fue recopilada a partir de [23][24][20].

3.2. Reglas del *backgammon*

El juego de *backgammon* es un juego que consta de dos fases; la fase de contacto, que es cuando las fichas más atrasadas de los jugadores aún pueden ser capturadas; y la de carrera, en la cual las fichas de los jugadores ya no pueden ser capturadas y sólo se depende del valor de los dados para retirar las fichas.

Backgammon es un juego para dos jugadores, el cual se juega sobre un tablero que consiste de 24 triángulos, llamados casillas. Los triángulos alternan en color y están agrupados en cuatro cuadrantes de seis triángulos cada uno. Los cuadrantes son exteriores o interiores para cada jugador. Los cuadrantes exteriores y los interiores están separados por una barra.

Las casillas están enumeradas para cada jugador iniciando en el cuadrante interior del jugador. La casilla más lejana es la número 24, la cual corresponde a la número 1 del oponente. Cada jugador tiene 15 fichas de un color. La posición inicial de las fichas es: 2 en la casilla 24, 5 en la casilla 13, 3 en la casilla 8 y 5 en la casilla 6 de cada jugador, como se muestra en la figura 3.1.

Cada jugador tiene su propio par de dados y un dado compartido. Un dado para el doblado con los números 2, 4, 8, 16, 32 y 64, es usado para guardar el valor final del juego.

3.2.1. Objetivo del juego

El objetivo del juego es mover todas las fichas al cuadrante interior y después retirarlas de ahí. El primer jugador en retirar todas sus fichas, gana el juego.

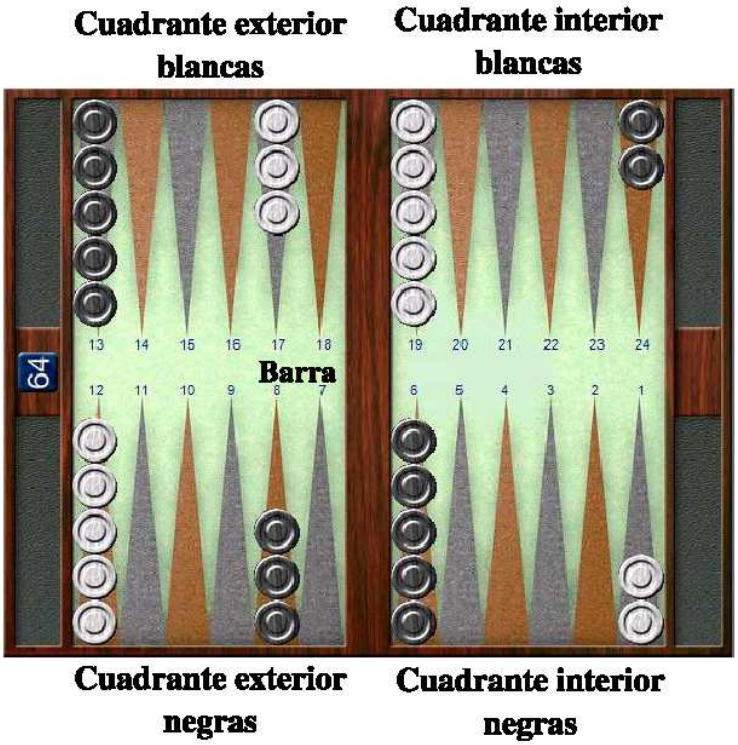

Figura 3.1: Posición inicial de las fichas

3.2.2. Movimiento de las fichas

Para iniciar el juego, cada jugador lanza un solo dado. Esto determina quién jugará primero y los números que se jugarán. Si el valor de los dados es el mismo, se vuelven a tirar los dados hasta que salgan valores diferentes. El jugador que obtiene el número mayor mueve sus fichas de acuerdo a los valores de los dados. Después de la primera jugada cada jugador tira sus dos dados alternando turnos.

Los valores de los dados le indican al jugador cuántas casillas moverá sus fichas. Las fichas siempre se mueven hacia adelante. En la figura 3.2 se muestra la dirección en que se deben mover las fichas negras, las fichas blancas avanzan en sentido opuesto. Además se deben seguir las siguientes reglas:

1. Una ficha puede ser movida únicamente a una casilla abierta, es decir,

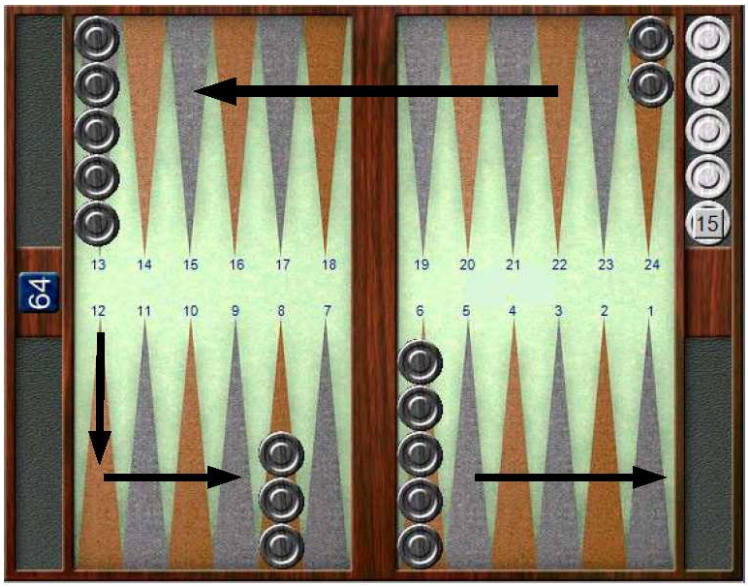

Figura 3.2: Dirección del movimiento de las fichas

una que no esté ocupada por dos o más fichas del oponente.

2. El número de los dos dados es interpretado como dos movimientos separados. Por ejemplo, si un jugador tira 5 y 3, puede mover una ficha 5 lugares hacia una casilla abierta y otra ficha 3 lugares hacia una casilla abierta, o puede mover un ficha 8 lugares hacia una casilla abierta, pero solamente si la casilla intermedia (la tercera o la quinta a partir de la posición inicial) está abierta (ver figura 3.3).

3. Un jugador que tira valores iguales, debe tirar los valores indicados dos veces. Una tirada de 6 y 6 significa que el jugador tiene cuatro números seis para usar, y puede realizar cualquier combinación de fichas que considere apropiadas para completar este requerimiento.

4. Un jugador debe usar ambos valores de los dados si es legalmente posible. Cuando únicamente puede ser jugado el valor de un dado, el jugador debe realizar este movimiento. O si alguno de los dos puede ser jugado, pero no ambos, el jugador debe mover al más grande. Cuando ningún valor puede ser usado, el jugador pierde su turno.

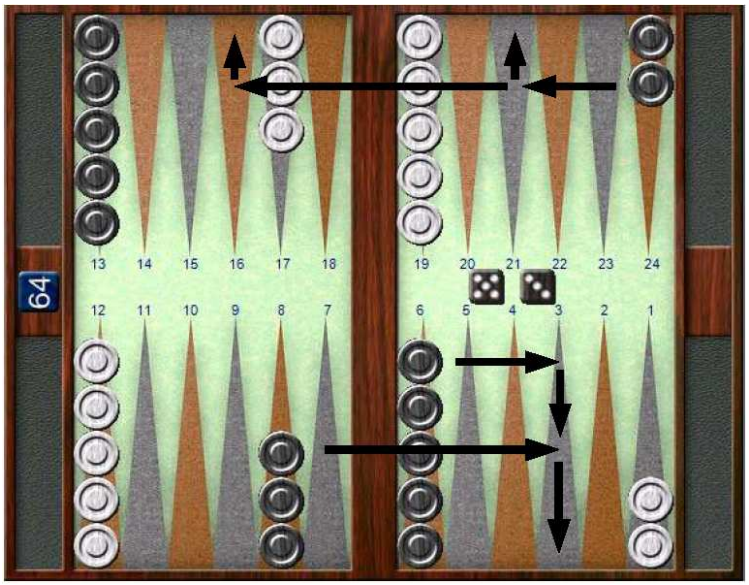

Figura 3.3: Dos formas distintas de jugar

3.2.3. Capturando e introduciendo las fichas

Si existe una única ficha en una casilla, se dice que esta ficha está aislada. Si una ficha oponente cae sobre ésta, es capturada y situada en la barra.

Si en algún momento, algún jugador tiene una o más fichas en la barra, su primera obligación es introducir esta ficha en el cuadrante interior del oponente. Una ficha es introducida moviéndola hacia una casilla abierta correspondiente a uno de los números de los dados.

Por ejemplo, si un jugador tira 4 y 6, puede introducir su ficha en la casilla 4 ó 6 mientras la casilla no esté ocupada por dos o más fichas del oponente (ver figura 3.4).

Si ninguno de los valores corresponden a una casilla abierta, el jugador pierde su turno. Si un jugador es capaz de introducir alguna, pero no todas sus fichas, debe meter las fichas que pueda y después perder el resto de su turno.

Después de que la última ficha de un jugador ha sido introducida, algún

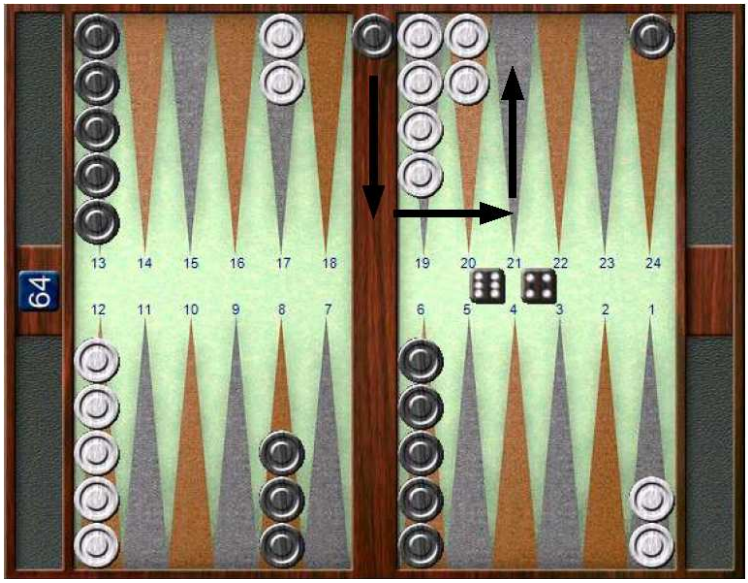

Figura 3.4: Forma de introducir una ficha

valor no usado de los dados debe ser jugado, moviendo la ficha que ha sido introducida o alguna otra.

3.2.4. Retirando fichas

Una vez que un jugador ha movido sus 15 fichas a su cuadrante interior, puede comenzar a retirarlas. Un jugador retira sus fichas tirando un número que corresponda a la casilla en la cual se encuentra alguna de sus fichas. Así tirando un 6, permite al jugador retirar una ficha que esté en la casilla 6.

Si no hay fichas en la casilla indicada por los dados, el jugador debe realizar un movimiento legal, moviendo una ficha que está en una casilla mayor a la indicada. Si no hay fichas en una casilla mayor a la indicada, se debe retirar la ficha que se encuentra en la casilla mayor (ver figura 3.5).

Un jugador debe tener todas sus fichas activas dentro de su cuadrante interior para poder retirarlas. Si una ficha es capturada durante este proceso, el jugador debe llevar esta ficha a su cuadrante interior antes de continuar retirando sus fichas. El primer jugador en retirar sus 15 fichas gana el juego.

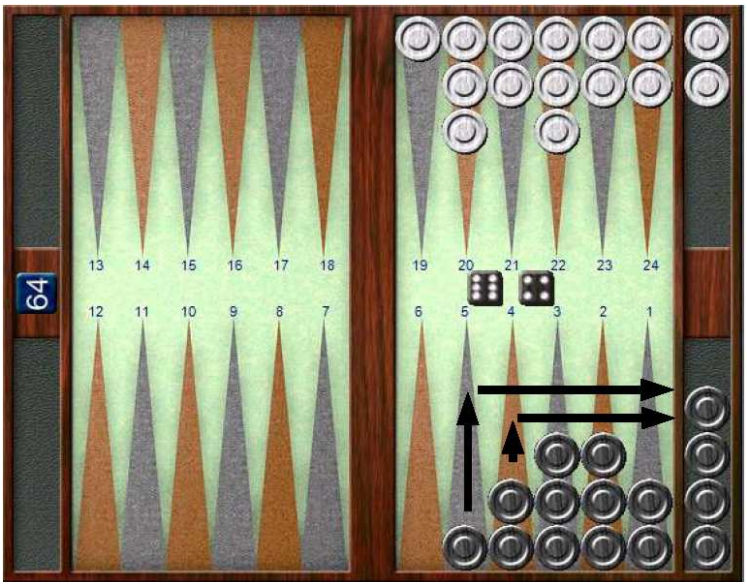

Figura 3.5: Retirando las fichas

3.2.5. Doblado

Backgammon es un juego para un número acordado de puntos. Cada juego inicia en un punto. Durante el curso del juego, un jugador que cree tener suficiente ventaja puede proponer doblar los puntos. Él puede hacerlo únicamente al inicio de su turno y antes de tirar los dados.

Al jugador que le han ofrecido doblar puede rechazarlo, en cuyo caso concede el juego y paga con un punto. De otra manera, puede aceptar el doblado y jugar por la nueva puntuación. El jugador que acepta un doblado se convierte en el propietario del dado y solamente él puede realizar el siguiente doblado.

Los siguientes doblados en el mismo juego son llamados redoblados. Si un jugador rechaza un redoblado, debe pagar el número de puntos que estaban establecidos antes del redoblado. De otra manera, se convierte en el propietario del dado y el juego continúa al doble de los puntos anteriores. No hay límite para el número de redoblados en el juego.

3.2.6. Tipos de victoria

Al final del juego, si el jugador perdedor ha retirado al menos una ficha, únicamente pierde el valor indicado por el dado para el doblado (un punto si no hubo doblados). Sin embargo, si el perdedor no retiró alguna de sus fichas, tiene un *gammon* y pierde el doble del valor indicado por el dado para el doblado. O, peor aún, si el perdedor no retiró alguna de sus fichas y aún tiene una ficha en la barra o en el cuadrante interior del ganador, tiene un *backgammon* y pierde el triple del valor indicado por el dado para el doblado.

3.3. Estrategia

En [25] y [26] recomiendan algunas estategias simples del juego de *backgammon*, para poder tener un buen desempeño y poder tener mayor posibilidad de ganar. A continuación se describen las estrategias recomendadas.

1. Ya que el *backgammon* es un juego de carrera y contacto, el primer jugador que logra hacer un bloqueo de seis casillas consecutivas y luego captura una ficha oponente, tiene mayor probabilidad de ganar.

2. Las fichas más atrasadas son las que deben recorrer la mayor distancia, por lo que el oponente deberá bloquear el avance de estas fichas y de ser posible, capturar las fichas aisladas que encuentre en su camino para hacer que recorran una mayor distancia. Si existe una ficha oponente en la barra el jugador deberá de bloquear su cuadrante interior para evitar que el oponente introduzca sus fichas al tablero, provocando la pérdida de turnos del oponente.

3. Para evitar que sean capturadas la fichas es necesario tener dos o más en una misma casilla y tratar de no tener muchas fichas aisladas.

4. Es recomendable tener una buena distribución de las fichas en el cuadrante exterior propio para que en el siguiente turno tengamos una mayor probabilidad de formar un bloque o formar bloques consecutivos.

5. Cuando ambos jugadores han logrado mover sus fichas hacia su lado del tablero, el juego se convierte en carrera y el jugador que tire los valores más altos, tendrá mayor probabilidad de ganar el juego.

Capítulo 4

Algoritmos genéticos

Los algoritmos genéticos fueron propuestos por John H. Holland [27], aunque él los llamó *planes reproductivos genéticos* pero con el tiempo se fueron popularizando con el nombre de algoritmos genéticos que es como se les conoce hoy en día. Principalmente estaban orientados a la resolución de problemas de aprendizaje de máquina, aunque con el paso del tiempo se fueron empleando para la resolución de otros tipos de problemas.

4.1. Conceptos básicos

Para poder entender el funcionamiento de los algoritmo genéticos, es necesario primero definir algunos conceptos básicos.

Cromosoma: Es una estructura de datos que contiene una cadena de variables de diseño o genes. Esta estructura puede estar representada como una cadena de bits o un arreglo de algún tipo de datos.

Gen: Es una subsección de un cromosoma el cual codifica el valor de una variable.

Genotipo: Es la codificación que se le da a las variables, por ejemplo una codificación binaria, como se muestra en la figura 4.2.

Fenotipo: Es la decodificación del cromosoma, es decir, es el valor obtenido al convertir la codificación (por ejemplo, binaria) a su valor real. Como en el caso de la figura 4.2 el número 18 es el fenotipo.

Individuo: Es un miembro de una población. Cada individuo contiene o está formado por un cromosoma.

Alelo: Es el valor posible que puede contener cierto gen.

Población: Es el conjunto de individuos de una generación.

4.2. Algoritmo genético básico

En el algoritmo básicamente se realizan los siguientes pasos:

1. Generar una población aleatoria de alguna cantidad determinada de individuos, la cual es conocida como población inicial.
2. Calcular la aptitud de cada uno de los individuos de la población.
3. Seleccionar con base en la aptitud, mediante algún método, a los individuos candidatos para reproducirse.
4. Aplicar los operadores genéticos de cruza y mutación a los individuos seleccionados anteriormente para generar la siguiente población.
5. Seleccionar al mejor individuo de la población y agregarlo a la nueva población. A este procedimiento se le llama elitismo.
6. Repetir los pasos 2 al 5 hasta que se cumpla cierta condición de paro.

4.3. Representación

Para la utilización de un algoritmo genético es importante definir un tipo de representación para los individuos, en este caso se le llama cromosoma a la estructura de datos utilizada, la cual está formada por genes, como se muestra en la figura 4.1, en la que se puede observar que un cromosoma puede estar formado de cualquier cantidad de genes, dependiendo de las condiciones del problema a resolver. Además cada gen tiene un valor V_i el cual se denomina alelo, en este caso, los alelos son los que nos indican el tipo de representación utilizada.

Existen varios tipos de representación tales como la representación binaria, en donde los alelos únicamente pueden tomar los valores de 1 o 0; también

V_1	V_2	V_3	...	V_n

Figura 4.1: Representación de un cromosoma

se puede utilizar la representación entera, en donde cada alelo podrá tomar valores de tipo entero; entre otras, las cuales se describen más adelante.

4.3.1. Representación binaria

La representación binaria es una de las más utilizadas ya que fue la que mejor funcionó cuando Holland realizó sus experimentos comparados con la representación real, ya que la representación binaria tiene más esquemas que la representación entera. En este tipo de representación los genes únicamente pueden tener los valores de 0 y 1, como se muestra en la figura 4.2

1	0	0	1	0	$= 18$

Figura 4.2: Representación binaria

Una de las desventajas que tiene este tipo de representación es cuando se tiene la necesidad de representar variables dentro de un rango amplio, o cuando se están manejando muchas variables, puesto que al usar representación binaria bajo estas condiciones, nos produce cromosomas de una longitud grande, haciendo que existan problemas de memoria y de velocidad al momento de la decodificación, además de que la convergencia es más lenta.

4.3.2. Representación binaria con códigos de Gray

Otro tipo de representación es la binaria con códigos de Gray, la cual surgió debido a que se observó que el espacio de búsqueda no es mapeado adecuadamete por el espacio de representación cuando se utiliza representación binaria simple, esto es, si en el espacio de búsqueda tenemos los números 3 y 4 los cuales son adyacentes, en el espacio de representación binaria tendremos los números 011 y 100, los cuales difieren por completo y no parecen adyacentes.

En los códigos de Gray, en todo número adyacente siempre existe diferencia de un bit, de esta forma la propiedad de adyacencia que pueda existir en el espacio de búsqueda se preserva en el espacio de representación.

Para convertir un número binario a un número en código de Gray hacemos un XOR a sus bits consecutivos de derecha a izquierda, permaneciendo igual el bit que está más a la izquierda. Por ejemplo, si tenemos el número 100 en binario, para obtener el número en código de Gray hacemos 0 XOR 0 = 0, 0 XOR 1 = 1, obteniendo como resultado 110. De esta forma, para los números 3 y 4 en el espacio de búsqueda tendremos los número 010 y 110 en el espacio de representación, los cuales únicamente varían en un bit. En la figura 4.3 se muestra la representación con códigos de Gray, junto con sus equivalentes en binario y en decimal.

1	0	0	1	0	= 11011 = 27

Figura 4.3: Representación con códigos de Gray

4.3.3. Representación para números reales

Como se mencionó anteriormente, la representación binaria tiene problemas cuando existen demasiadas variables o cuando el rango de las variables es muy amplio; lo mismo puede ocurrir con la representación con los códigos de Gray. En el caso de que se requiera representar números reales se tiene el problema de la precisión deseada, ya que si se requiere de una mayor precisión, el rango de la variable aumenta.

Existen varias formas para representar números reales; se ha sugerido la de utilizar el estándar del IEEE para precisión simple, en el cual un número real se representa usando 32 bits, de los cuales 8 se usan para el exponente usando una notación en exceso -127, y la mantisa se representa con 23 bits, como es el caso de la figura 4.4. El problema que se tiene con este esquema es que un pequeño cambio en el exponente puede tener un gran cambio en el espacio de búsqueda. Otra propuesta es usar el valor del número real tal cual en un gen como se muestra en la figura 4.5. También existe la propuesta de poner en cada gen un número entero perteneciente al número real y tener almacenada la posición del punto, tal como se muestra en la figura 4.6.

Signo	Exponente	Mantisa
0	10001011	101...01
1 bit	8 bits	23 bits

Figura 4.4: Representación real con el estándar IEEE

0.729	−1.212	3.145	...	1.345

Figura 4.5: Representación real con los números en cada gen

4.3.4. Otros tipos de representación

Los tipos de representación mencionados anteriormente no son los únicos que existen, pero si son de los más utilizados. Dentro de la teoría de los algoritmos genéticos se han propuesto diversas formas de representación entre las cuales se tiene representaciones de longitud variable, representación utilizando árboles, representación para permutaciones, representación matricial, representación multidimencional, entre otras. Principalmente, estos tipos de representaciones han surgido con la finalidad de evitar las desventajas que proporcionan los tipos de representación clásicos y otras más han surgido debido a que se adaptan mejor al problema a resolver.

4.4. Aptitud

La aptitud de un individuo o función de aptitud es un valor que se le asigna a cada individuo de la población para determinar qué tan bueno es con respecto a los demás. Esto es con la finalidad de determinar qué individuos de la población serán seleccionados para ser reproducidos y a partir de ellos obtener a los nuevos individuos.

El valor de la aptitud de un individuo depende principalmente de lo que se esté evaluando. Por ejemplo, si queremos encontrar el valor mínimo de la

4	9	3	7	2

$= 49.372$

Figura 4.6: Representación real utilizando números enteros

función $f = x^2$ y tenemos a los individuos $i_1 = 1$ y $i_2 = 2$, entonces $f(i_1) = 1$ y $f(i_2) = 4$, en donde se tiene que i_1 es menor que i_2. Por lo tanto, como queremos encontrar el valor mínimo de f, el individuo i_1 debe tener una mayor aptitud que el individuo i_2 para tener una mayor probabilidad de ser seleccionado, ya que es mejor.

4.5. Selección

La selección es un punto importante dentro de los algoritmos genéticos, ya que de ésta depende cuáles serán los individuos elegidos para reproducirse, debido a que la selección a menudo se lleva a cabo de forma probabilística, por lo que los individuos menos aptos tienen una probabilidad distinta de cero de ser seleccionados. Las técnicas de selección se dividen en tres grupos:

- Selección proporcional
- Selección mediante torneo
- Selección de estado uniforme

4.5.1. Selección proporcional

Este tipo de selección fue propuesto originalmente por Holland [27] y en ella los individuos son seleccionados dependiendo de su aptitud y del tamaño de la población. Este tipo de selección se puede dividir en cuatro subgrupos: [28]

1. La Ruleta
2. Sobrante estocástico
3. Universal estocástica
4. Muestreo determinístico

Además, a estos subgrupos se les pueden agregar mecanismos para modificar el número esperado de copias de cada individuo, como son:

- Escalamiento sigma
- Jerarquías
- Selección de Boltzmann

La ruleta

Este método fue propuesto por DeJong [29] y su algoritmo es el siguiente:

1. Calcular la suma de valores esperados T
2. Repetir N veces (N es el tamaño de la población):
   - *a*) Generar un número aleatorio r entre 0.0 y T
   - *b*) Ciclar a través de los individuos de la población sumando los valores esperados hasta que la suma sea mayor o igual a r
   - *c*) El individuo que haga que esta suma exceda el límite es el seleccionado.

El valor esperado es el número esperado de copias que se esperan obtener de un individuo. Este algoritmo tiene la desventaja de que el individuo menos apto puede llegar a ser seleccionado más de una vez. El algoritmo también se vuelve ineficiente cuando el tamaño de población crece, ya que este algoritmo tiene un orden de complejidad $O(n^2)$,

Sobrante estocástico

Este método fue propuesto por Booker [30] y Brindle [31] con la finalidad de que la probabilidad de elección de individuos esté más cerca al valor esperado de cada uno. Para ello obtiene los valores esperados (Ve) con la siguiente fórmula:

$$Ve = \frac{f_i}{\bar{f}}$$

donde f_i es la aptitud del individuo i y $\bar{f}$ es el promedio de la aptitud de todos los individuos en la población, para i de 1 A N y N es el tamaño de la población. El algoritmo es como sigue:

1. Seleccionar de manera determinística a los individuos cuyo valor esperado tenga una parte entera. El individuo se seleccionará el número de veces indicado por la parte entera.
2. Los valores restantes (parte fraccionaria) se usan probabilísticamente para rellenar la población.

El paso 2 puede realizarse de dos maneras distintas:

Sin reemplazo: Cada sobrante se usa para sesgar el tiro de una moneda que determina si un individuo se selecciona de nuevo o no.

Con reemplazo: Los sobrantes se usan para dimensionar los segmentos de una ruleta y se adopta esta técnica de manera tradicional.

El problema de este método es que puede producir convergencia prematura, ya que ejerce una mayor presión de selección.

Universal estocástica

Propuesta por Baker [32] con el objetivo de minimizar la mala distribución de los individuos en la población en función de sus valores esperados. El algoritmo es el siguiente:

1. Asignar a ptr un valor aleatorio entre 0 y 1
2. Para $sum = 0, i = 1; i <= n; i++$
3. Para $sum += Ve(i); sum > ptr; ptr += 1$
4. Seleccionar al individuo i

Los problemas que tiene este algoritmo es que puede ocasionar convergencia prematura y hace que los individuos más aptos se multipliquen muy rápidamente.

Muestreo determinístico

Esta es una variante a la propuesta de DeJong y es similar al sobrante estocástico, pero requiere de un algoritmo de ordenación. Su algoritmo es el siguiente:

1. Calcular $P_{selec} = \frac{f_i}{\sum f}$
2. Calcular $Ve_i = P_{selec} * n$ (n es el tamaño de la población)
3. Asignar determinísticamente la parte entera de Ve_i
4. Ordenar la población de acuerdo a las partes decimales (de mayor a menor)
5. Obtener los padres faltantes de la parte superior de la lista.

Este método tiene los mismos problemas que el sobrante estocástico.

Escalamiento sigma

Es una técnica ideada para mapear la aptitud original de un individuo con su valor esperado de manera que el algoritmo genético sea menos susceptible a la convergencia prematura. La idea principal de esta técnica es mantener más o menos constante la presión de selección a lo largo del proceso evolutivo. Usando esta técnica, el valor esperado de un individuo está en función de su aptitud, la media de la población y la desviación estándar de la población:

$$Ve_i = \begin{cases} 1 + \frac{f_i - \bar{f}}{2\sigma} & \text{si } \sigma \neq 0 \\ 1.0 & \text{si } \sigma = 0 \end{cases}$$

$$\sigma = \sqrt{\frac{n\sum f^2 - (\sum f)^2}{n^2}}$$

Si $Ve_i < 0$ se hace $Ve_i = 0.1$ El escalamiento sigma produce el siguiente comportamiento en la selección:

- Al inicio de una corrida, el valor alto de la desviación estándar impedirá que los mejores individuos obtengan los segmentos más grandes de la ruleta.
- Hacia el final, la desviación estándar será más baja y los individuos más aptos podrán multiplicarse más fácilmente.

Selección por jerarquías

Propuesta por Baker [33] para evitar la convergencia prematura en las técnicas de selección proporcional. El objetivo de esta técnica es disminuir la presión de selección. Los individuos se clasifican con base en su aptitud, y se les selecciona con base en su rango (o jerarquía) y no con base en su aptitud. Las jerarquías previenen la convergencia prematura, de hecho, lo que hacen, es alentar la velocidad de convergencia del algoritmo genético. El algoritmo es el siguiente:

1. Ordenar (o jerarquizar) la población con base en su aptitud, de 1 a N (donde 1 representa al menos apto).
2. Elegir $Max(1 \leq Max \leq 2)$
3. Calcular $Min = 2 - Max$

4. El valor esperado de cada individuo será:

$$Ve_i = Min + (Max - Min)\frac{jerarquia_i - 1}{N - 1}$$

   Baker recomendó $Max = 1.1$

5. Usar selección proporcional aplicando los valores esperados obtenidos de la expresión anterior.

Selección de Boltzman

Esta técnica fue propuesta por Goldberg [34] y está basada en el recocido simulado, la idea es usar una función de variación de temperatura que controle la presión de selección. Se usa un valor alto de temperatura al principio, lo cual hace que la presión de selección sea baja. Con el paso de las generaciones, la temperatura disminuye, lo que aumenta la presión de selección. De esta manera se incita a un comportamiento exploratorio en las primeras generaciones y se acota a uno más explotatorio hacia el final del proceso evolutivo.

Típicamente, se usa la siguiente expresión para calcular el valor esperado de un individuo.

$$Ve_i = \frac{e^{\frac{f_i}{T}}}{\langle e^{\frac{f_i}{T}} \rangle_t}$$

donde T es la temperatura y $\langle\rangle_t$ es el promedio de la población en la generación t.

Esta técnica tiene el inconveniente de que es necesario definir una función de variación de temperatura.

4.5.2. Selección mediante torneo

Esta técnica fue propuesta por Wetzel [35]. La idea es hacer la selección con base en comparaciones directas entre los individuos. Este tipo de selección se puede hacer de manera determinista o de manera probabilística. El algoritmo determinista es el siguiente:

1. Barajar los individuos de la población.

2. Escoger un número p de individuos (típicamente 2).

3. Compararlos con base en su aptitud.

4. El ganador del torneo es el individuo más apto.

5. Debe barajarse la población un total de p veces para seleccionar N padres (donde N es el tamaño de la población).

Para el algoritmo probabilista se elige un número real P entre 0 y 1, después se genera otro número aleatorio entre 0 y 1, si este número está dentro del rango de P, se selecciona al individuo más apto, en caso contrario se selecciona al menos apto. El valor de P es fijo durante todo el proceso evolutivo y $0.5 \leq P \leq 1$.

4.5.3. Selección de estado uniforme

Esta técnica fue propuesta por Whitley [36] y se utiliza en los algoritmos genéticos no generacionales, en los cuales sólo unos cuantos individuos son reemplazados en cada generación (los menos aptos). Esta técnica suele usarse cuando se evolucionan sistemas basados en reglas (por ejemplo, sistemas de clasificadores) en los que el aprendizaje es incremental. El algoritmo es el siguiente:

1. Llamaremos G a la población original de un algoritmo genético.

2. Seleccionar R individuos ($1 \leq R < M$) de entre los más aptos

3. Efectuar cruza y mutación a los R individuos seleccionados. Llamaremos H a los hijos.

4. Elegir al mejor individuo en H (o a los μ mejores).

5. Reemplazar los μ peores individuos de G por lo μ mejores individuos de H.

4.6. Cruza

El proceso de cruza consiste básicamente en seleccionar bloques de los cromosomas de individuos e intercambiarlos entre ellos. Para ello existen diversas técnicas de las cuales se describen a continuación las más básicas y

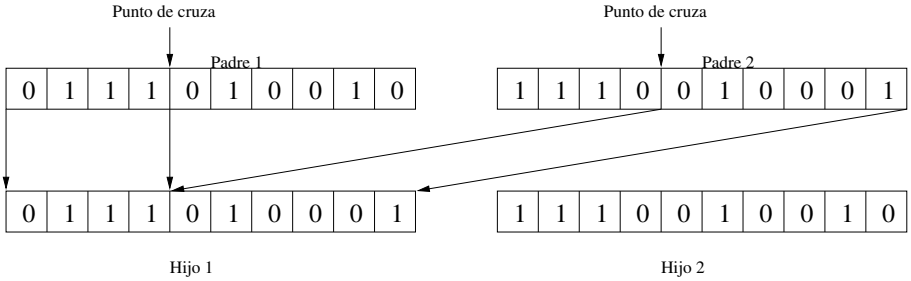

Figura 4.7: Cruza de un punto

utilizadas. Por lo general la cruza se realiza entre dos individuos seleccionados, llamados padres, y a partir de ellos se generan dos individuos para la nueva generación, llamados hijos.

4.6.1. Cruza de un punto

Esta técnica fue propuesta por Holland y como su nombre lo dice, consiste en generar un punto P de manera aleatoria, que nos indica la posición hasta donde se intercambiarán los genes de los cromosomas. Como se muestra en la figura 4.7 la primera parte del cromosoma del hijo 1 está formada por la primera parte del padre 1 y la segunda parte está formada por la segunda parte del padre 2 y la primera parte del cromosoma del hijo 2 está formada por la primera parte del padre 2 y la segunda parte está formada por la segunda parte del padre 1.

4.6.2. Cruza de dos puntos

En este caso, la cruza de 2 puntos consiste en elegir de manera aleatoria dos puntos, los cuales indicarán los bloques que se intercambiarán para producir a los hijos, como se ilustra en la figura 4.8

4.6.3. Cruza uniforme

Este tipo de cruza puede considerarse como una cruza generalizada, ya que puede tener n puntos de cruza. Para ello se utiliza un valor Pc el cual indica la probabilidad de tomar un gen de un padre o de otro. Si se elige

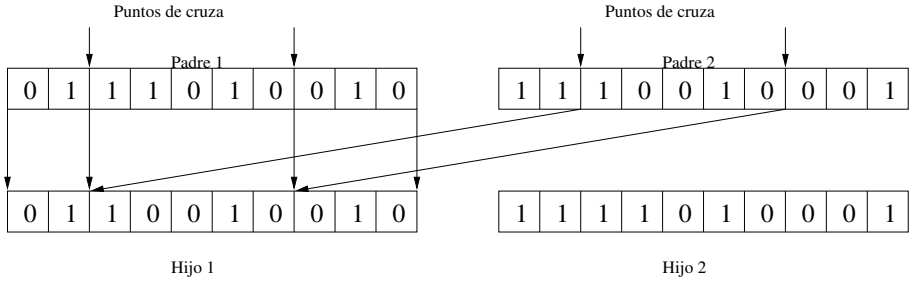

Figura 4.8: Cruza de dos puntos

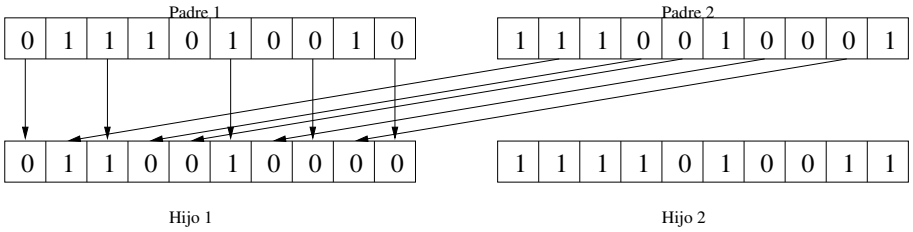

Figura 4.9: Cruza uniforme con $Pc = 0.5$

$Pc = 0.5$, los hijos deben de estar formados por exactamente la mitad de cada padre, como se muestra en la figura 4.9.

4.6.4. Otros tipos de cruza

Existen otros tipos de cruza que se han utilizado en otros trabajos, tal como es el caso de la cruza acentuada [37] en un intento por implementar un mecanismo de auto adaptación para la generación de los patrones favorables (o sea, los buenos bloques constructores) de la cruza. Otros tipos de cruza existentes dependen del tipo de representación que se este utilizando, como por ejemplo, la cruza para permutaciones, ya que en este tipo de problemas no debe de haber valores repetidos, por lo tanto, se necesita un tipo de cruza especial; o cuando utilizamos representación con número reales, necesitamos otro tipo de cruza la cual nos permita generar diversos números reales y así poder explorar mejor el espacio de búsqueda.

4.7. Mutación

La mutación es considerada un operador secundario en los algoritmos genéticos y por tanto no se aplica demasiadas veces sobre los cromosomas como es el caso de la cruza.

Básicamente lo que hace la mutación es tomar la información de un gen y cambiar el valor por alguno permitido, en el caso de la representación binaria o con códigos de Gray, debido a que sólo se permiten 0's y 1's, si se encuentra un 1 al mutar se cambiará el valor por un 0 y veceversa. En otros casos como en la representación real se puede elegir un número aleatorio que esté en el rango permitido e intercambiar el valor del gen. En otros casos, en donde se utilicen permutaciones, será necesario aplicar una técnica con la que se puedan conservar las reglas para permutaciones. Un ejemplo es seleccionar un número y luego seleccionar un lugar para anexar este número, recorriendo a todos los demás; con esto se sigue conservando que no existan números repetidos.

Debido a que la mutación es un operador secundario, la probabilidad de que ésta sea aplicada, por lo general es muy baja. Los porcentajes de probabilidad suelen estar entre el rango de 0.001 y 0.01. Hay quienes recomiendan que la probabilidad de mutación sea de $\frac{1}{L}$, donde L es la longitud del cromosoma. También se ha sugerido comenzar con porcentajes de probabilidad e ir disminuyéndolos conforme pasa el número de generaciones para que al final tengamos porcentajes de mutación pequeños. Esto con la finalidad de que al inicio se explore más el espacio de búsqueda y al final tenga una mejor convergencia.

4.8. Elitismo

El elitismo consiste en tomar al mejor individuo de la generación anterior y agregarlo a la nueva generación, reemplazando a algún nuevo individuo de manera aleatoria. Con esto, se asegura de que en las siguientes generaciones no se disminuirá el valor máximo alcanzado. Se ha demostrado en [38] que los algoritmos genéticos, requieren del elitismo para poder converger hacia el óptimo.

4.9. Micro algoritmo genético

El término micro algoritmo genético (micro-AG) hace referencia a un algoritmo genético con una población muy pequeña y algún proceso de reinicialización. Esta idea fue sugerida por D. E. Goldberg [39] a partir de algunos resultados teóricos, de acuerdo a los cuales una población de tres individuos era suficiente para encontrar convergencia sin importar el tamaño del cromosoma. El proceso sugerido por Goldberg fue el de aplicar los operadores genéticos a una población pequeña generada aleatoriamente, hasta alcanzar una convergencia. Una vez hecho esto, se genera un nuevo individuo a partir de la población anterior y los demás individuos son generados aleatoriamente.

Existen algunos reportes en la literatura sobre el uso de micro-AG con buenos resultados, tal como el caso de Krishnakumar [40], quien con una población de 5 individuos obtuvo mejores y más rápidos resultados que una algoritmo genético con población de 50 individuos sobre tres problemas distintos.

Para mayor información sobre algoritmos genéticos o algunos otros aspectos relacionados con la computación evolutiva se puede consultar [41].

Capítulo 5

Autómatas de *backgammon*

Este capítulo comienza con la explicación básica del sistema Heinze-Ortiz [20], ya que éste fue utilizado como infraestructura básica para agregar nuestra heurística. Después se da a conocer la abstracción del problema, mencionando algunas ideas iniciales, las cuales no funcionaron correctamente, y finalmente se presenta el algoritmo utilizado.

Para la solución del problema se empleó un algoritmo genético con variantes en los tipos de cruza, porcentajes de mutación, tamaño de población y tipos de representación.

5.1. Sistema Heinze-Ortiz

En esta sección se hace una breve descripción sobre el funcionamiento básico del sistema realizado por Heinze-Ortiz, ya que se utilizó este sistema como plataforma principal, agregándole únicamente el módulo inteligente desarrollado en esta tesis, el cual se basa en algoritmos genéticos.

Esta aplicación está desarollada en el lenguaje C# y está constituida por tres módulos con sus respectivas clases. Cada uno de los módulos tiene un proposito específico. Los tres módulos con los que cuenta son: **Interfaz del usuario de *Backgammon*, Juego de *Backgammon* y Agente de *Backgammon***. En la figura 5.1 se representa la forma en que interactúan las clases correspondientes a los módulos de Juego de *Backgammon* y Agente de *Backgammon*. A continuación se describe el funcionamiento de los módulos así como el de sus clases.

- **Interfaz del usuario de *Backgammon*** La principal función de este módulo es obtener solicitudes del usuario tales como realizar un movimiento, tirar los dados o elegir a los agentes que competirán; también será responsable de mostrar los diálogos necesarios para un juego válido.

- **Juego de *Backgammon*** Este módulo es responsable de la representación del juego, así como de toda la lógica del juego, asegurándose que no puedan realizarse movimientos inválidos, o que no se pueda finalizar un turno si aún existen movimientos válidos. Tambiés es responsable de asignar el turno a quien le corresponde jugar. En otras palabras, este módulo es responsable de toda la representación del juego para asegurar una partida legal de acuerdo a las reglas del *backgammon*. Una funcionalidad adicional de este módulo es que tiene la capacidad de obtener el historial de un juego con la finalidad de proporcionarlo al agente si así lo requiere, para su aprendizaje. Las clases correspondientes a este módulo se describen a continuación.

  Interfaz del juego Esta clase contiene métodos para iniciar un nuevo juego, realizar un movimiento, finalizar un turno, indicar al ganador de la partida, devolver el tipo de victoria obtenida (normal, *gammon* o *backgammon*), deshacer el último movimiento si aún no se ha finalizado el turno, validar un movimiento, indicar quien es el jugador actual, indicar la posición y cantidad de fichas de un jugador, obtener el valor de los dados, obtener los movimientos realizados por un jugador, asignar un valor a los dados y habilitar la edición del tablero.

  Historial del juego Esta clase es la encargada de almacenar en un arreglo dinámico las jugadas de una partida para que cuando sean solicitadas por un agente, éste las use para aprender.

  Representación del tablero Esta clase indica el estado actual del tablero, es decir, indica cuántas y a quién le pertenecen las fichas en una casilla determinada, las fichas que se encuentran en el tablero, así como las que se han retirado.

- **Agente de *Backgammon*** En este módulo están los agentes que son capaces de jugar *backgammon*. Aquí, *Agente* es la clase central. Cuando ésta es instanciada, carga e inicializa todos sus módulos de deci-

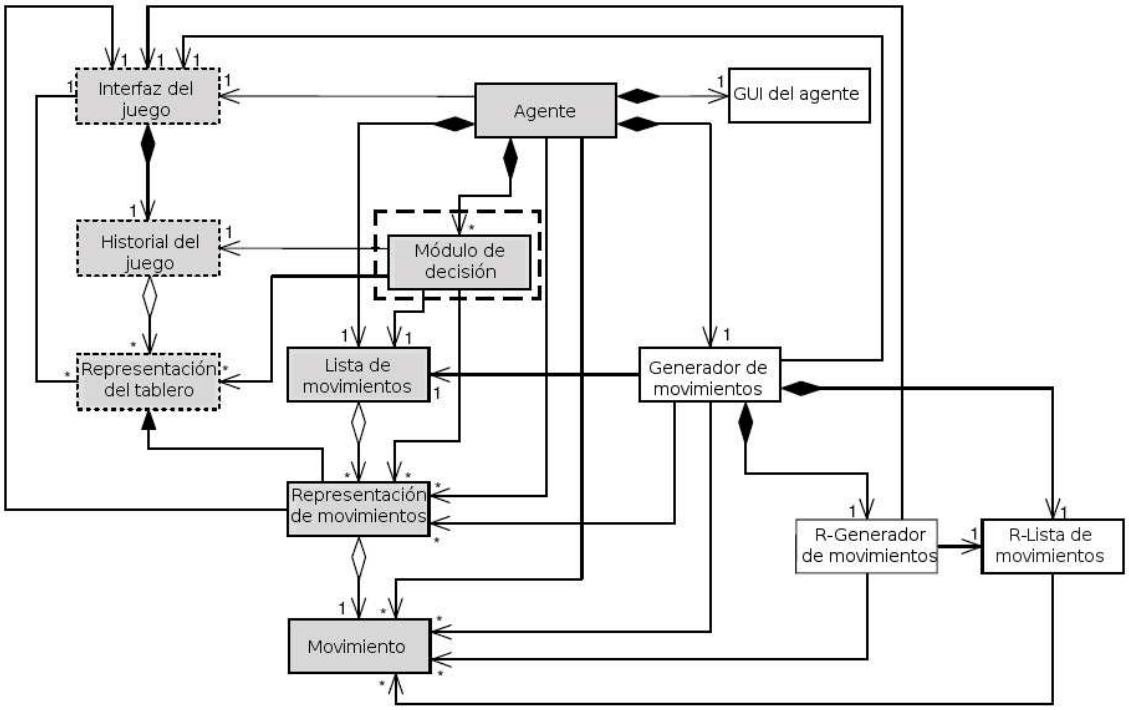

Figura 5.1: Interacción entre las clases del sistema

sión, así como a las clases *Lista de movimientos* y *Generador de movimientos*. La clase *Generador de movimientos* carga un objeto de tipo *R-Generador de movimientos*, responsable de generar todos los movimientos válidos, representados por la clase *Movimiento*. Cuando todos los movimientos válidos han sido generados, la clase *Generador de movimientos* es responsable de crear todos los posibles nuevos tableros representados por la clase *Representación de movimientos*, la cual puede ser creada en el juego. Después de crear la clase *Lista de movimientos* se envía a todos los módulos de decisión, los cuales son responsables de asignar una puntuación a todos los tableros. Finalmente, el agente realiza el movimiento con la mayor puntuación. Esta interacción es la que se muestra en la figura 5.1.

Agente Esta clase representa a un jugador de computadora el cual se apoya en sus módulos de decisión para jugar *backgammon*. Dentro de esta clase el agente generará todos los posibles movimientos que pueden ser realizados en el juego y permite que el módulo de decisión evalúe cada uno de los tableros posibles, realizando el movimiento con mayor puntuación. Si un juego ha finalizado, tiene un método el cual puede ser llamado para enviar el historial del juego a los módulos de decisión.

Lista de movimientos Se encarga de agregar los movimientos válidos a una lista, verificando que no se agreguen movimientos repetidos.

Representación de movimientos Representa un posible tablero nuevo que puede ser creado y recibe de los módulos de decisión la puntuación asignada a cada movimiento. Una vez recibida esta puntuación, la normaliza en un rango de -1 a 1.

Movimiento Esta clase representa un movimiento, la cual está formada por una variable que indica la posición de una ficha y hacia dónde se moverá, así como cuál es su primer movimiento y cuál el siguiente. Estos movimientos se almacenan en una lista ligada, ya que se pueden tener hasta cuatro movimientos.

Módulo de decisión Esta es una clase abstracta la cual debe ser heredada por todos los módulos de decisión. Contiene métodos para proporcionar el nombre del agente, normalizar la puntuación asignada a cada jugada, y proporcionar el historial del juego al agente.

Esta es la clase que ocuparemos para anexar la heurística y pueda interactuar con el sistema y los demás agentes.

5.2. Representación del problema

Uno de los aspectos más importantes para resolver un problema es la representación de éste, es decir, la forma de abstraerlo de una manera entendible para la computadora. Este punto es muy importante ya que el contar con una buena representación incrementa la probabilidad de obtener buenos resultados.

En el área de los juegos de tablero existen muchas formas para representar los problemas, esto depende principalmente de las características del juego. Como se explicó en la sección 1.1, en el juego de *backgammon* se tiene la posibilidad de jugar en promedio 20 movimientos legales por cada turno. Debido a esto, es recomendable obtener los movimientos válidos después de tirar los dados y proceder a evaluarlos por medio de la heurística propuesta, para finalmente realizar el mejor movimiento indicado por la heurística.

Ahora que sabemos que evaluaremos todos los movimientos posibles (en promedio 20), es necesaria una técnica para saber cuál es el mejor movimiento y por qué. Para ello se utilizaron métodos que intentan cuantificar las estrategias del juego, definidas en la sección 3.3.

La primera propuesta fue tomar en cuenta a las fichas que se movían, verificando qué tanto cumplían con las estrategias del juego. Al tomar en cuenta sólo las fichas que se mueven, se dejaba sin evaluar a las demás fichas, con lo cual no se tenía una buena forma de evaluar el movimiento por lo que no se obtuvieron buenos resultados, además que el código era demasiado extenso y complejo.

Posteriormente se cambió la forma de evaluar los movimientos, siendo ésta la que se está utilizando. Lo que se hace es simular el movimiento y evaluar la posición del tablero una vez realizado el movimiento. De esta forma se toman en cuenta tanto la posición de todas nuestras fichas, como las del oponente. Con esta nueva forma de evaluación se han obtenido los mejores resultados con un código mucho más simple y sencillo. Para realizar la evaluación de las estrategias se utilizan unas reglas, las cuales se almacenan en un vector.

Inicialmente se contaba con un vector de 24 reglas, pero después de una depuración se eligieron 16, ya que con éstas se obtuvo un mejor desempeño.

5.2.1. Vector de reglas

Para la evaluación de los movimientos se requiere de 16 reglas, las cuales están almacenadas en un vector que es utilizado por la heurística para ajustar el peso dado a cada regla.

El vector de reglas es básicamente un arreglo de 16 enteros en un rango de 1 a 100 excepto por el cuarto valor el cual está en el rango de 0 a 36 y el noveno el cual está en el rango de 3 a 15. Debido a que el *backgammon* consta de dos fases (contacto y carrera), el vector está dividido en dos partes, los primeros 12 valores son utilizados cuando el juego se encuentra en la fase de contacto y los 4 últimos valores se utilizan cuando el juego está en la fase de carrera. A continuación se define la conformación del vector de reglas.

- El primer valor del vector es un peso que se le da a la construcción de bloques consecutivos. Es decir, si se tienen 5 bloques consecutivos, el peso se multiplicará por 5, dando el peso final para esta regla.
- La segunda regla indica cuántas casillas tienen 2 o más fichas. Esto es con la finalidad de apropiarse de varias casillas para así tener una buena distribución de las fichas. El número de casillas apropiadas se multiplica por el segundo valor del vector para obtener su peso final.
- El tercer valor del vector se multiplica por la cantidad de fichas capturadas al oponente después de haber realizado el movimiento. Mientras más fichas del oponente hayan sido capturadas mayor será el peso asociado.
- Si existen fichas capturadas del oponente se calcula la probabilidad de que el oponente introduzca su ficha en el tablero. Si esta probabilidad es menor que el cuarto valor del vector se suma el quinto valor del vector al peso total del movimiento. Para manejar únicamente números enteros el rango de probabilidad está entre 0 y 36.
- El sexto valor del vector es un peso que se da por cada casilla con 2 o más fichas que estén en el cuadrante interno. Esto es para tener

bloqueado el cuadrante interno y para que, al capturar una ficha del oponente, se tenga una probabilidad baja de ingresar al tablero.

- Dependiendo de la cantidad de fichas atrasadas y en la casilla en la que se encuentren se asignará un peso para cada ficha el cual está indicado por el séptimo valor del vector. Este peso siempre será negativo, ya que no es recomendable tener muchas fichas atrasadas.

- El octavo valor del vector es un peso negativo que se aplica por cada ficha aislada en el cuadrante interno. Esta regla sólo se aplica en caso de que existan fichas atrasadas del oponente.

- El noveno valor del vector indica el número máximo de fichas que puede haber en una casilla. El décimo valor del vector es un peso negativo para evitar que se acumulen varias fichas en una sola casilla. Si el noveno valor es excedido, el décimo valor será restado al peso total. El rango del noveno valor del vector está entre 3 y 15, ya que en total hay 15 fichas y siempre es bueno tener 2 fichas en una casilla.

- Debido a que en algunas pruebas se observó que teniendo demasiadas fichas aisladas se perdían varios juegos de tipo *backgammon* o *gammon*, se puso una restricción al número máximo de fichas aisladas, en este caso se permiten 3 fichas aisladas. El undécimo valor del vector es un valor negativo que se asocia cuando existen más de 3 fichas aisladas.

- Debido a que existen movimientos en los que es forzoso dejar una o más fichas solas, se obtiene la probabilidad de que sea capturada la ficha aislada. El valor de probabilidad de captura se multiplica por el duodécimo valor del vector y es restado al peso total. Mientras más fichas aisladas existan, el peso restado será mayor. Esta regla no se aplica en caso de que tengamos 6 bloques consecutivos y que al menos una ficha del oponente esté atrapada por esta barrera.

- Estando en la fase de carrera se buscan las fichas atrasadas y se les asigna un peso de acuerdo a la posición en la que se encuentren. Si hay fichas en el cuadrante exterior propio se asigna un valor negativo de acuerdo a la cantidad de fichas y se multiplica por el decimotercer valor del vector. Si las fichas se encuentran en el cuadrante exterior del oponente, se multiplica por el decimocuarto valor del vector y si las fichas se encuentran en el cuadrante interno del oponente se multiplica

por el decimoquinto valor del vector. En este caso existen restricciones de los rangos las cuales son que el decimoquinto valor sea mayor o igual que el decimocuarto y que éste sea mayor o igual que el decimotercero. Estas reglas son para minimizar pérdidas tipo *backgammon* o *gammon*.

- El decimosexto valor del vector es un peso que se da para mover las fichas que están fuera del cuadrante interno hacia la casilla 6, ya que con esto se evita tener derrotas de tipo *gammon*.

Además del vector de reglas, existen unas reglas deterministas las cuales contribuyen directamente al valor final de cada movimiento, esto debido a que utilizando únicamente los pesos del vector de reglas se obtienen muchos movimientos con valores iguales, siendo que no todos los movimientos son buenos.

Estando en la fase de contacto, si es posible comenzar a retirar las fichas y existen fichas del oponente capturadas, se aumenta el valor del movimiento mientras menos fichas propias estén en el tablero, ya que con esto se obtienen más victorias de tipo *backgammon*. Se halló experimentalmente que si esta regla no se agrega, el autómata comienza a apilar sus fichas en las primeras casillas sin comenzar a retirarlas, dándole la oportunidad al oponente de que reingrese sus fichas al tablero.

En la fase de carrera se aumenta el valor del movimiento por cada ficha que se esté retirando, ya que siempre es mejor retirar las fichas en vez de sólo avanzarlas. Cuando no es posible retirar las fichas, pero existe una casilla con demasiadas fichas, se aumenta el valor si se mueven fichas de esa casilla. También se aumenta el valor cuando hay fichas que se mueven a una casilla vacía.

Al final, estos valores son sumados y el resultado es asignado a cada movimiento, éstos son ordenados de mayor a menor siendo el de mayor valor el elegido para ser jugado por el autómata. En caso de que dos a más movimientos tengan el mismo valor, se elige alguno de ellos de manera aleatoria.

5.3. Algoritmo genético

En esta sección se explica la manera en que se resolvió el problema utilizando un algoritmo genético, así como los tipos de operadores y los paráme-

tros empleados.

Inicialmente se realizó un programa determinista en el cual se establecieron las reglas que se consideraron adecuadas para el funcionamento del programa. En total se contaba con 24 reglas por lo que se utilizaba un vector de 24 enteros. Los valores asignados al vector se proporcionaron empíricamente, esto fue, dándoles mayor peso a las reglas que se consideraban mejor que las demás y un menor peso a las que se les consideraba de menor importancia, tratando de que fueran los mejores de acuerdo a las estrategias.

Una vez terminadas de establecerse las reglas para el vector, se procedió a realizar algunos experimentos utilizando el algoritmo genético para tratar de encontrar los mejores pesos. Al mismo tiempo, se realizaron pruebas para poder determinar la cantidad máxima de juegos que debía jugar cada individuo para asignarle su aptitud. Esto fue debido a que existe una variabilidad en el porcentaje de victorias dependiendo de la cantidad de juegos jugados. También se determinó que era necesario designar a tres individuos como élite de cada generación, ya que de esta manera se conserva al mejor individuo y no se pierde debido a la variabilidad.

En estos primeros experimentos se observó que no se tenían buenos porcentajes de victorias, por lo que se procedió a hacer una depuración de las reglas para determinar cuáles eran esenciales para el buen funcionamiento de la heurística; esto se hizo con la suposición de la existencia de reglas que se contraponían. Después de realizar dicha depuración se obtuvo un total de 16 reglas, por lo cual se tiene un vector de 16 enteros, descrito anteriormente, los cuales son ajustados por el algoritmo genético.

5.3.1. Algoritmo

Se está utilizando un algoritmo genético simple, con unas pequeñas variantes, el cual se muestra a continuación e inmediatamente se explican a detalle las variaciones y operadores utilizados.

Algoritmo 1. Pseudocódigo del algoritmo genético utilizado

Entrada: Tamaño de población TP, número de generaciones N, número de juegos J, probabilidad de cruza pc y mutación pm.
Salida: Encuentra el mejor individuo (vector de pesos)

```
1 : Crea una población inicial aleatoria de tamaño TP
    excepto por el primer individuo.
2 : Para i = 1 hasta N hacer
3 :   Calcular la aptitud para cada individuo
      Aptitud = porcentaje de victorias durante J juegos.
4 :   De acuerdo a la aptitud, seleccionar a los
      individuos a ser reproducidos (padres)
5 :   Con probabilidad pc aplicar la cruza a los padres para crear a
      los hijos.
6 :   Con probabilidad pm aplicar la mutación a los hijos.
      Los hijos ahora son la nueva población
7 :   Tomar a los 3 mejores individuos de la generación anterior y
      agregarlos a la nueva generación.
    Fin Para
8 : Regresa El mejor individuo
```

Tamaño de población : El tamaño de la población cambia con respecto al experimento que se realizaba; el mínimo número de individuos utilizados fue de 4 y el máximo de 80. Como se observa en el algoritmo, cuando se crea la población inicial, el primer individuo no se genera de forma aleatoria, ya que el éste es tomado de la propuesta determinista o se toma al mejor individuo del experiento anterior, en caso de que aplique, con la finalidad de que exista una guía inicial; a esto se le llama aprendizaje incremental.

Función de aptitud : Para asignar el valor de la función de aptitud lo que se hace, es poner a competir a los individuos del algoritmo genético en contra de *Pubeval* en un torneo de un número determinado de juegos (J). En la mayoría de los experimentos se realizaban torneos de 1,000 juegos, el porcentaje de victorias obtenido durante el torneo es la aptitud de cada individuo.

Cruza : Los tipos de cruza utilizados son la de un punto, de dos puntos y la cruza uniforme. La probabilidad de cruza pc siempre fue de 0.9.

Debido a que la cruza uniforme necesita un parámetro adicional para la elección del padre, se probaron con dos valores de probabilidad; estas probabilidades son de 0.3 y 0.5.

Mutación : Se utilizaron distintos valores de probabilidad de mutación pm, los cuales están en el rango de 0.01 a 0.14. Además se utilizó mutación variable la cual inicia con una probabilidad de 0.14 y se va decrementando gradualmente conforme aumenta el número de generaciones, hasta llegar a obtener una probabilidad de mutación de 0.01, la cual se mantiene para las últimas generaciones.

Selección : El tipo de selección utilizado fue la de el sobrante estocástico con reemplazamiento usando escalamiento sigma.

Elitismo : Como se observa en el algoritmo, estamos tomando a 3 individuos como élite, en vez de la manera tradicional que sólo es uno. Esto se realizó debido a que en torneos cortos existe una variabilidad grande en el porcentaje de victorias, por lo que la aptitud de los individuos varía de generación en generación y no siempre el individuo con mayor aptitud es el mejor de la generación. Dado esto, si únicamente tomamos un individuo como élite no obtendremos buenos resultados. Experimentalmente se observó que la mínima cantidad de individuos elitistas es tres, ya que así difícilmente se pierde al mejor individuo global, con esto se obtuvieron mejores resultados.

Tipo de representación : Se emplearon tres tipos de representaciones, que fueron representación entera, representación binaria y representación binaria con códigos de Gray.

Codificación y decodificación : Debido a que estamos utilizando representación binaria y de códigos de Gray, es necesario contar con un método para convertir estos números a enteros y viceversa, ya que para nuestro vector de pesos utilizamos números enteros. En ambos tipos de representación se está utilizando un cromosoma de 108 de longitud, ya que de acuerdo al rango de los pesos es la mínima longitud que se pudo obtener siendo el primer bit el menos significativo y el último bit el más significativo. Para codificar el vector de pesos, primero convertimos los valores enteros a números binarios y en caso de usar representación con códigos de Gray hacemos la conversión a partir del número en binario,

almacenando el valor resultante en el cromosoma. Para decodificar el cromosoma solamente se procede a realizar el proceso inverso, de código de Gray a binario y de binario a entero.

Criterio de paro : En la mayoría de los experimentos el criterio de paro utilizado fue cuando después de 5 generaciones el mejor individuo no era reemplazado.

Además del algoritmo genético simple, se utilizó un micro algoritmo genético, en el cual, la diferencia es que consiste de pocos individuos, por lo general menos de 6. El microalgoritmo génetico se utilizó para poder aumentar la cantidad de juegos por torneo, esto fue con la finalidad de hacer que la aptitud de los individuos no fuera tan variable, por lo que se utilizaron únicamente cuatro individuos en torneos de 100,000 juegos, y es aquí donde se utilizó la probabilidad de mutación variable, ya que con la probabilidad fija se tenian dos problemas: 1) Si la probabilidad de mutación es alta no se observa convergencia, 2) Si la probabilidad de mutación es baja la convergencia es demasiado rápida y no se explora suficientemente el espacio de búsqueda.

5.4. Solución por redes neuronales

En esta sección se definen algunos conceptos sobre diferencia temporal y redes neuronales, ya que adicionalmente se realizaron otros experimentos tomando como base el autómata TD-Gammon haciéndole una modificación con la finalidad de mejorar sus resultados.

5.4.1. Diferencia Temporal y TD-Gammon

El método de diferencia temporal (TD) es un tipo de aprendizaje por reforzamiento. La idea básica del método TD es que su aprendizaje está basado en la diferencia de predicciones sucesivas temporales. La fórmula que usa este método es como sigue:

$$V(s) = V(s) + \alpha[V(s') - V(s)]$$

- $V(s)$ es el estado anterior.
- $V(s')$ es el estado actual.

- α es un parámetro positivo el cual influye en el porcentaje de aprendizaje.

TD-Gammon utiliza 2 redes neuronales (una para cada fase del juego), para la fase de contacto utiliza una red neuronal con 248 entradas, una capa oculta con 128 neuronas. Para la fase de carrera usa una red neuronal con 224 entradas y una capa oculta de 116 neuronas, las entradas de las redes neuronales representan información sobre la posición del tablero. Las dos redes neuronales constan de 5 neuronas de salida, de las cuales la primer neurona es utilizada cuando se utiliza el método de diferencia temporal para hacer el ajuste. La segunda y tercera neurona indican una probabilidad de que determinada posición del tablero lleve a ganar una victoria de tipo normal o de tipo *gammon* y las últimas 2 neuronas son para indicar las derrotas de tipo normal o *gammon*. La figura 5.2 muestra un esquema similar al usado por las redes neuronales de TD-Gammon. Estas redes neuronales emplean el algoritmo de *back propogation*[42] para su entrenamiento. La red neuronal para la fase de contacto consta de 32384 pesos y la de fase de carrera consta de 26564 pesos. El entrenamiento de estas redes neuronales para la obtención de un autómata competitivo requiere de 2 000 000 de juegos, requiriendo de mucho tiempo de cómputo.

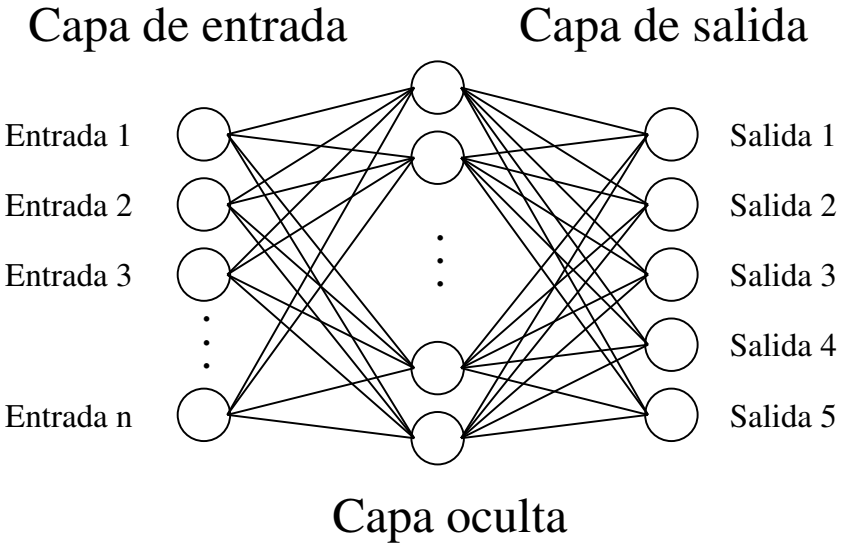

Figura 5.2: Esquema de la red neuronal usada por TD-Gammon

5.4.2. Ensamble de redes neuronales

Existen diversos métodos para clasificación o predicción, como por ejemplo el *bagging* [43] (votación para clasificación), cuyo principal objetivo es combinar los resultados de varios clasificadores para obtener uno nuevo, minimizando el error de predicción. Básicamente lo que se hace es tomar en cuenta las decisiones de todos los clasificadores y realizar la acción de la mayoría. Los clasificadores pueden ser de distintos modelos o del mismo modelo, pero con un aprendizaje distinto. Este método de clasificación funciona muy bien cuando los clasificadores no son muy estables, ya que se mejora mucho su desempeño.

De acuerdo a esto, tomamos las redes neuronales que son utilizadas por TD-Gammon y a partir de éstas generamos otras redes (réplicas, ver figura 5.3). En nuestro caso, las nuevas redes fueron generadas introduciendo un ruido gaussiano por cada peso de las redes originales, ya que el entrenar nuevas redes nos llevaría demasiado tiempo, debido a la cantidad de juegos necesarios para un buen desempeño. Al conjunto de las réplicas junto con las originales le llamamos ensamble de redes neuronales.

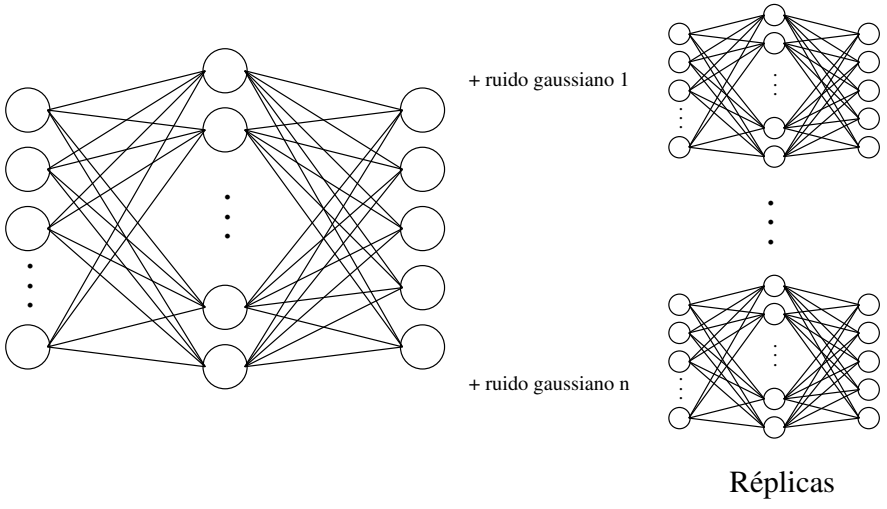

Figura 5.3: Ensamble de redes neuronales

En el siguiente capítulo se muestran los resultados experimentales obtenidos tras aplicar las técnicas descritas en este capítulo. Así mismo, se presentan

algunas gráficas que fueron necesarias realizar para explicar el comportamiento de nuestros resultado en comparación con los demás reportados.

Capítulo 6

Experimentos y resultados

En este capítulo se describen los experimentos y resultados obtenidos, tanto del algoritmo genético empleado, como del ensamble de redes neuronales, se presentan unas tablas para observar el progreso de los experimentos, así como una tabla con resultados comparativos y unas gráficas para saber la cantidad de juegos por torneo necesarios para poder garantizar la fiabilidad de nuestros resultados. A nuestro autámata creado le llamamos GA-Gammon.

6.1. Algoritmo genético

De todos los operadores y parámetros explicados anteriormente se realizaron experimentos con una mezcla de éstos.

En los primeros experimentos se utilizó representación entera aplicando los distintos tipos de cruza y mutación descritos anteriormente, pero de esta forma el porcentaje de victorias más alto fue de 44.2 %, como se muestra en la tabla 6.1.

	Normal	Gammon	Backgammon	Victorias
GA-Gammon	382	67	2	44.2 %
Pubeval	438	107	3	55.8 %
Diferencia	56	40	1	

Tabla 6.1: Resultados con representación entera

Posteriormente se utilizó representación binaria y se realizaron experimentos con los mismos tipos de cruza y mutación descritos anteriormente, en donde se lograron obtener porcentajes de victorias del 46.9 %, como se muestra en la tabla 6.2.

	Victorias	Normal	Gammon	Backgammon	Victorias
GA-Gammon	474	385	88	2	46.9 %
Pubeval	526	414	107	5	53.1 %
Diferencia	52	29	19	3	

Tabla 6.2: Resultados con representación binaria

También se utilizó un micro algoritmo genético con representación binaria, los distintos tipos de cruza mencionados anteriormente y con probabilidad de mutación de 0.14 para poder tener diversidad en los individuos. En este experimento no se obtuvieron buenos resultados debido a la variación en la aptitud de los individuos entre cada generación, perdiendo fácilmente al mejor individuo cuando comenzaban a parecerse entre sí. Además de que la función de aptitud se asignaba por torneos de 1 000 juegos, por lo que no fue una buena forma de asignar la aptitud para este caso. Esta variación principalmente se vio reflejada en los resultados finales, ya que en lugar de ir aumentando el porcentaje de victorias conforme aumentaba el número de generaciones, comenzaba a disminuir el porcentaje de victorias. El criterio para la reinicialización del micro algoritmo genético se realizaba cuando la diferencia del porcentaje de victorias de los individuos era menor o igual al 0.1 %. Al finalizar el experimento se obtenían porcentajes de victorias entre el 39 % y 41 %, siendo que los mejores individuos en algún momento de la evolución tenían entre el 42 % y 43 % de victorias.

Otro experimento realizado fue utilizando parte de la información de las redes neuronales, ya que éste es un buen jugador. Se intentó obtener información a partir de las redes neuronales para poder asignar la aptitud a cada individuo, con la finalidad de obtener resultados en un corto tiempo, ya que en este caso la aptitud de cada individuo aumentaba conforme su mejor movimiento fuera similar al mejor movimiento de las redes neuronales. La ventaja de este método es que se obtenían resultados rápidamente, pero la desventaja y principal razón por la que se dejó de usar esta técnica fue

que después de varios experimentos sólo se lograron obtener porcentajes del 42.3 % de victorias. Para poder explicar mejor esta técnica consideremos la tabla 6.3 en donde la columna de la izquierda representa los movimientos de la red neuronal para una jugada y la columna de la derecha representa los movimientos del i-ésimo individuo, en donde $M1$ representa al mejor movimiento de las redes neuronales y m_{i1} representan el del i-ésimo individuo, así mismo Mn, m_{in} son los peores movimientos, respectivamente. Los experimentos realizados fueron los siguientes:

RN	AG individuo$_i$
$M1$	m_{i1}
$M2$	m_{i2}
$M3$	m_{i3}
$\vdots$	$\vdots$
Mn	m_{in}

Tabla 6.3: Movimientos de la red neuronal y del i-ésimo individuo

- El primero consistió en asignar la aptitud a los individuos de acuerdo a la posición que tienen respecto al mejor movimiento de las redes neuronales. Es decir, si el mejor movimiento de las redes neuronales, es el mejor movimiento del individuo, entonces ese individuo recibe una aptitud mayor V ($V = 10,5$ ó 3, por lo que se realizaron 3 experimentos diferentes cambiando el valor de V). En caso de que un individuo tenga en segundo lugar el mejor movimiento indicado por las redes neuronales, éste recibira el valor de V decrementado en 1. Para el caso en que el individuo tenga en tercer lugar el mejor movimiento de las redes neuronales, éste recibirá una aptitud decrementada en 2 y así sucesivamente con las demás posiciones. Para éste y todos los casos sólo se asignaban valores positivos, así que si un individuo tenía al mejor movimiento en una posición mayor a V no se incrementaba ni se decrementaba la aptitud. Esto es:

  Si $m_{ij} = M1$ entonces $Aptitud_i = Aptitud_i + [V - (n-1)]$ para $j = min(V, n)$

- En el segundo experimento se tomó el mismo programa que en el primer experimento. La única diferencia es que sólo se incrementaba la aptitud

si el individuo tenía como mejor movimiento el mismo que las redes neuronales. Esto es:

$$\text{Si } m_{i1} = M1 \text{ entonces } Aptitud_i = Aptitud_i + V$$

- El tercer experimento que se realizó fue el de tratar de ir ajustando las reglas de acuerdo al peso de las redes neuronales, tomando como base los 3 mejores resultados que las redes neuronales proporcionaban. Si el mejor movimiento de un individuo era el mejor movimiento de las redes neuronales se le asignaba una aptitud de 10 puntos por cada movimiento igual. Si el mejor movimiento de un individuo era el segundo mejor de las redes se la asigna una puntuación de 5 y si el mejor movimiento de un individuo era el tercer mejor movimiento de las redes se le asignaba una aptitud de 3. Esto es:

$$\begin{aligned}&\text{Si } m_{i1} = M1 \text{ entonces } Aptitud_i = Aptitud_i + 10\\ &\text{Si } m_{i1} = M2 \text{ entonces } Aptitud_i = Aptitud_i + 5\\ &\text{Si } m_{i1} = M3 \text{ entonces } Aptitud_i = Aptitud_i + 3\end{aligned}$$

En los experimentos también se varió el número de juegos (10, 20 y 50) y el número de generaciones (100, 50, 20). En todos los experimentos se usaron 12 individuos de los cuales los tres mejores pasan directo a la siguiente generación.

También se realizó un experimento en el que se mezclaron el mejor individuo obtenido por las redes neuronales con el mejor individuo obtenido a partir del porcentaje de victorias, esto con la finalidad de tratar de encontrar un individuo intermedio, pero lo mejor que se obtuvo fueron individuos con porcentajes del 44 % de victorias.

Finalmente, utilizando representación con códigos de Gray, los operadores de selección, cruza y mutación descritos anteriormente se logró obtener un individuo con porcentajes de victorias del 48.3 %. Éste fue encontrado utilizando probabilidad de mutación de 0.01, y cruza uniforme con probabilidad de selección del padre de 0.3. Este individuo fue encontrado en la generación 28 con una población de 20 individuos.

Después de todos estos experimentos, se analizó la forma de jugar del mejor individuo obtenido, para así determinar sus posibles fallas y tratar

de mejorarlo. Al observar la forma de jugar del mejor individuo se pudo detectar que era posible agregar una regla determinista para que tratara de conservar el cuadrante interno propio bloqueado en el caso de que existieran fichas capturadas del oponente y el individuo ya estuviera en la posibilidad de retirar sus fichas, esto con la finalidad de obtener más victorias de tipo *backgammon*. Otro aspecto importante que se observó, es que tal individuo perdía muchos juegos tipo *gammon*, esto debido a que no avanzaba sus fichas atrasadas a tiempo. Haciendo pruebas se logró observar que si no se toma en cuenta la probabilidad de captura cuando las fichas están entre las casillas 21 a la 24 se obtienen mejores resultados, ya que de esta manera las fichas atrasadas tienen una mayor libertad para lograr salir a tiempo. Con esta modificación se obtuvo un porcentaje de victorias de 50.1 %, como se muestra en la tabla 6.4.

	Victorias	Normal	Gammon	Backgammon	Porcentaje
GA-Gammon	505	381	117	7	50.1
Pubeval	495	363	126	6	49.9

Tabla 6.4: Resultados con códigos de Gray después de ajustar las reglas

Después de estas modificaciones se realizaron otros experimentos, pero esta vez únicamente se utilizó representación con códigos de Gray. En estos últimos experimentos se varió la probabilidad de mutación, los tipos de cruza, y el tamaño de la población. También se volvió a utilizar el micro algoritmo genético, pero esta vez aumentando el número de juegos por torneo, que en este caso fue de 100,000 para que la aptitud no fuera tan variable. Con el micro algoritmo genético utilizando 4 individuos, mutación variable de 0.14 a 0.01 se logró encontrar el mejor individuo hasta el momento, el cual obtiene porcentajes de victorias del 50.59 %, con una diferencia porcentual de 1.18 % a nuestro favor, como se muestra en la tabla 6.5.

Adicionalmente, debido a que se cuenta con el autómata Fuzzeval, se realizaron pruebas para tener otro punto de comparación. Los resultados se muestran en la tabla 6.6, en donde nuestra diferencia porcentual es de casi el 20 %.

Debido a que existe mucha variación en el porcentaje de victorias de acuerdo a la cantidad de juegos, se realizaron unas pruebas para obtener la

	Victorias	Normal	Gammon	Backgammon	Porcentaje
GA-Gammon	508	382	119	7	50.59 %
Pubeval	492	361	127	4	49.41 %
Diferencia	+16	+21	-8	+3	+1.18 %

Tabla 6.5: Resultados finales en contra de Pubeval

	Victorias	Normal	Gammon	Backgammon	Porcentaje
GA-Gammon	576	485	78	13	59.54 %
Fuzzeval	424	387	36	1	40.46 %
Diferencia	+152	+98	+42	+12	+19.08 %

Tabla 6.6: Resultados finales en contra de Fuzzeval

cantidad de juegos necesarios para garantizar que el porcentaje de victorias es realmente el que se reporta. Inicialmente se reportaban los resultados después de jugar torneos de 1000 juegos, pero existía mucha variación en los resultados, por lo que se procedió a calcular experimentalmente el promedio y desviación estándar cuando se juegan torneos de 1000 juegos. Los resultados obtenidos se muestran en la figura 6.1 la cual tiene una desviación estándar de 1.708, por lo que se concluye que no es conveniente ni confiable reportar los resultados en torneos tan cortos, ya que de esta manera podemos tener porcentajes por encima del 54 %.

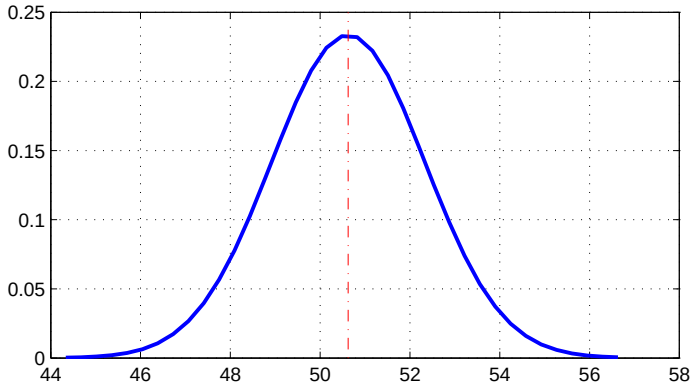

Figura 6.1: Gaussiana a 1,000 juegos. Con $\mu = 50.625$ y $\sigma = 1.708$

Las figuras 6.2 y 6.3 muestran la función de distribución de probabilidad de la normal para 80,000 y 100,000 juegos, respectivamente. En ellas se indica el promedio μ y su desviación estándar σ con respecto al porcentaje de victorias, y podemos podemos observar que sus respectivas desviaciones estándar son muy pequeñas y sus promedios son muy similares.

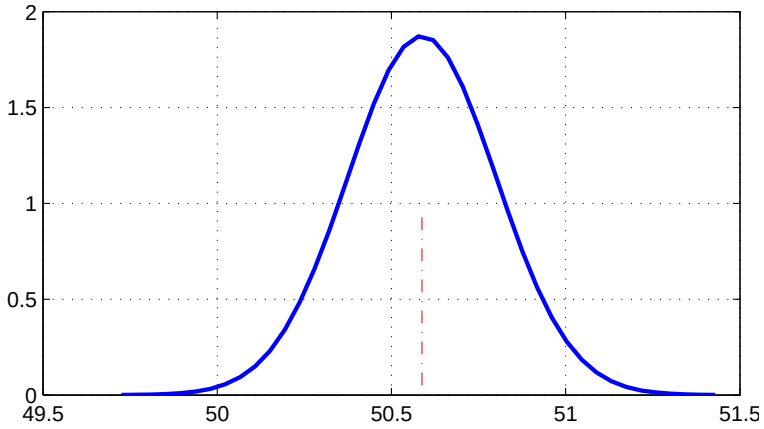

Figura 6.2: Gaussiana a 80,000 juegos. Con $\mu = 50.5875$ y $\sigma = 0.213$

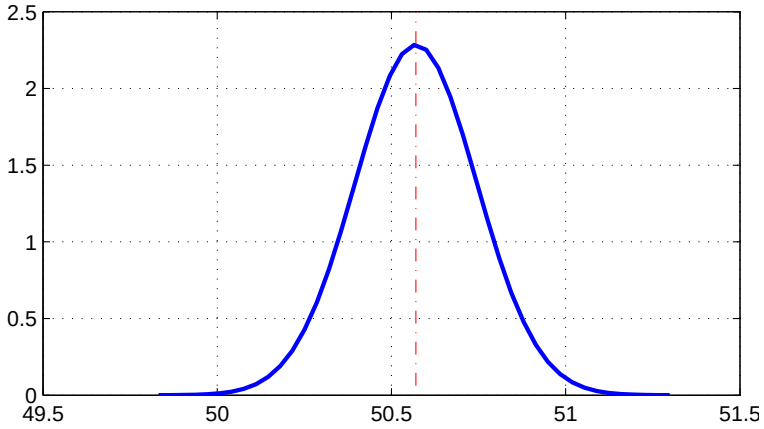

Figura 6.3: Gaussiana a 100,000 juegos. Con $\mu = 50.5703$ y $\sigma = 0.1746$

Con este experimento, ahora sí podemos garantizar que el porcentaje de victorias que reportamos está en el 50.6 % redondeado, después de jugar

torneos de 100,000 juegos, siendo que los demás autómatas reportan sus resultados en torneos cortos. La tabla 6.7 muestra los resultados comparativos con respecto a otros autómatas reportados en la literatura especializada.

Autómata	vs Pubeval	Técnica utilizada	JPT[1]
GP-Gammon [1]	62.4 %	Programación Genética	1 000
GMARLB [17]	51.2 %	AG-SMA-RNA[2]	1 000
GA-Gammon	**50.6 %**	Algoritmo Genético	100 000
ACT-R-Gammon [16]	45.94 %	ACT-R[3]	1 000
Fuzzeval [20][21]	42.0 %	Lógica difusa	100
HC-Gammon [14]	40.0 %	Hill Climbing	100

Tabla 6.7: Resultados comparativos

6.2. Redes neuronales

Para los experimentos con las redes neuronales únicamente se hicieron variaciones sobre dos parámetros: la desviación estándar del ruido gaussiano agregado a cada peso y el número de réplicas generadas a partir de la red neuronal original.

Se hicieron experimentos con 3, 10 y 40 réplicas de las redes neuronales. Estas fueron generadas al agregar un ruido gaussiano con $\sigma = 0.1, 0.5$ y 0.025 sobre cada uno de los pesos de la red original. Con la combinación de estos parámetros se realizaron experimentos en los que sólo las réplicas decidían la jugada a realizarse, sin tomar en cuenta la red neuronal original, obteniendo porcentajes de victorias por debajo del 50 %.

En otros experimentos se incluyó a la red neuronal original para tomar la decisión sobre la jugada a realizarse. En la mayoría de estos experimentos se comenzaba con porcentajes de victorias favorables, pero conforme se incrementaba el número de juegos, este porcentaje disminuía hasta que se estabilizaba alrededor del 50 %. Cabe mencionar que al agregar el ruido gaussiano

[1] Juegos Por Torneo
[2] Algoritmo Genético-Sistema de Multi Agentes-Red Neuronal Artificial
[3] Teoría del Conocimiento ACT-R

con $\sigma = 0.1$ también se obtenían porcentajes por debajo del 50 %, por lo que no es recomendable agregar demasiado ruido.

Excepto por el experimento en el que se crearon 10 réplicas de la red neuronal con un ruido gaussiano de $\sigma = 0.025$, en donde la red original es considerada para decidir la jugada a realizarse, se logró obtener un porcentaje de victorias del 50.3 % del ensamble en contra de TD-Gammon, después de un torneo de 100 000 juegos, como se muestra en las tabla 6.8.

	Victorias	Normal	Gammon	Backgammon	Porcentaje
Ensamble	50296	35680	14061	555	50.3 %
TD-Gammon	49704	35111	14102	491	49.7 %

Tabla 6.8: Resultados del torneo del Ensamble contra TD-Gammon

Finalmente, se procedió a realizar una competencia en contra de *Pubeval*, en donde TD-Gammon tiene un porcentaje de victorias del 65.3 % después de 100 000 juegos y el ensamble tiene un porcentaje de victorias del 65.8 %, como se muestra en las tablas 6.9 y 6.10. Por tanto, hemos obtenido un buen resultado, sabiendo que es muy difícil obtener porcentajes elevados de victorias.

	Victorias	Normal	Gammon	Backgammon	Porcentaje
TD-Gammon	62474	38664	21334	2476	65.3 %
Pubeval	37526	28314	8768	444	34.7 %

Tabla 6.9: Resultados del torneo de TD-Gammon contra Pubeval

	Victorias	Normal	Gammon	Backgammon	Porcentaje
Ensamble	62938	39120	21260	2558	65.8 %
Pubeval	37062	28040	8588	434	34.2 %

Tabla 6.10: Resultados del torneo del Ensamble contra Pubeval

Capítulo 7

Conclusiones y trabajo futuro

Conclusiones

Al finalizar la implantación del algoritmo genético y realizar experimentos en forma exhaustiva, logramos cumplir con el objetivo, el cual fue el de generar un autómata capaz de obtener porcentajes de victorias por encima del 50 % al competir en contra de *Puvebal*.

Al realizar los experimentos finales, se observó que con una pequeña cantidad de juegos existe mucha variación en el porcentaje de victorias. Es por eso que se realizaron otro tipo de pruebas para encontrar la cantidad de juegos necesarios con los cuales tenemos un nivel de confianza del 98 % de forma que podamos garantizar nuestro porcentaje de victorias.

De acuerdo a este problema de variabilidad encontrado y al citado en [1] es posible que en la última tabla de resultados, los otros autómatas tengan variaciones muy elevadas, ya que en sus reportes únicamente utilizan 1000 juegos.

Una de las ventajas que se ofrece con este trabajo es que se utilizó un algoritmo genético simple el cual es muy fácil de programar.

Debido a que se obtuvieron los resultados esperados se concluye que la forma de abstraer el problema y la interpretación de las estrategias del juego, por medio del vector de reglas, junto con los operadores de selección, cruza y mutación aplicados al algoritmo genético fueron de gran utilidad para cumplir con el objetivo.

Con respecto al ensamble de redes neuronales, se puede concluir que posiblemente las redes neuronales que utiliza TD-Gammon son muy estables ya que la mejora no fue mucha, pero sí significativa al utilizar el ensamble.

Los resultados mostrados en este trabajo de tesis fueron publicados en [44]

Trabajo futuro

De acuerdo a experimentos realizados podría ser conveniente emplear otro método para la abstracción del problema, tal vez el utilizado por una de las técnicas reportadas en la literatura, las cuales obtienen porcentajes de victorias elevados.

También podrían emplearse otras técnicas para el proceso de evolución. En lugar de que los todos individuos evolucionen jugando en contra de *Pubeval* se podría poner a los individuos a jugar en contra de distintos autómatas para tener una mayor diversidad en las jugadas, así como también se podría intentar probar con la coevolución, esperando mejorar nuestros resultados.

Bibliografía

[1] Y. Azaria and M. Sipper. GP-Gammon: Using Genetic Programming to Evolve Backgammon Players. In *Proceedings of the 8th European Conference on Genetic Programming*, volume 3447 of *Lecture Notes in Computer Science*, pages 132–142. Springer, 2005.

[2] E. S. Venttsel. *Introducción a la Teoría de los Juegos*. Temas Matemáticos, Limusa Wiley, 1973.

[3] La enciclopedia libre Wikipedia. Teoría de juegos. http://es.wikipedia.org/wiki/Teoría_de_juegos.

[4] Stanford Encyclopedia of Philosophy. Game theory. http://plato.stanford.edu/entries/game-theory/.

[5] J. von Neuman and O. Morgenstern. *Theory of Game and Economic Behavior*. Princeton, Nueva Jersey: Princeton University Press, 1944.

[6] A. A. Cournot. *Recherches sur les principes mathématiques de la théorie des richesses*. Dunod, 1838 (2001)).

[7] W. Stadler. Initiators of Multicriteria Optimization. Division of Engineering, San Francisco State University.

[8] M. Campbell. Knowledge discovery in Deep Blue. *Communications of the ACM*, 42(11):65–67, 1999.

[9] J. S. Jang, C. T. Sun, and E. Mizutani. *Neuro-fuzzy and Soft Computing*. Prentice Hall, 1997.

[10] J. Schaeffer, R. Lake, P. Lu, and M. Bryant. Chinook: The World Man-Machine Checkers Champion. *AI Magazine*, 17(1):21–29, 1996.

[11] K. Chellapilla and D. B. Fogel. Evolving Neural Networks to Play Checkers without Relying on Expert Knowledge. *IEEE Transactions on Neural Networks*, 10(6):1382–1391, 1999.

[12] G. Tesauro. Temporal Difference Learning and TD-Gammon. *Communications of the ACM*, 38(3):58–68, 1995.

[13] G. Tesauro. Software-source code benchmark player, pubeval.c. http://www.bkgm.com/rgb/rgb.cgi?view+610, 1993.

[14] J. B. Pollack, A. D. Blair, and M. Land. Coevolution of a Backgammon Player. In *Artificial Life V: Proceedings of the Fifth International Workshop on the Synthesis and Simulation of Living Systems*, pages 92–98. MIT Press, 1997.

[15] G. Tesauro. Comments on co-evolution in the succesful learning of backgammon strategy. *Machine Learning*, 32:241–243, 1998.

[16] S. Sanner, J. R. Anderson, C. Lebiere, and M. Lovett. Achieving Efficient and Cognitively Plausible Learning in Backgammon. *In 17th International Conference on Machine Learning*, pages 823–830, 2000.

[17] D. Qi and R. Sun. Integrating Reinforcement Learning, Bidding and Genetic Algorithms. In *Proceedings of the International Conference on Intelligent Agent Technology (IAT)*, pages 53–59. IEEE Computer Society Press, 2003.

[18] M. Sipper, Y. Azaria, A. Hauptman, and Y. Shichel. Attaining Human-Competitive Game Playing with Genetic Programming. *IEEE Transactions on Systems, Man and Cybernetics, Part C, Draft.*

[19] Y. Azaria and M. Sipper. GP-Gammon: Genetically Programming Backgammon Players, DRAFT. 2005.

[20] M. Heinze. Intelligent Game Agents. Developing an Adaptive Fuzzy Controlled Backgammon Agent. Master's thesis, Aalborg University Esbjerg, December 2004.

[21] M. Heinze, D. Ortiz Arroyo, H. L. Larsen, and F. Rodriguez-Henríquez. Fuzzeval: A Fuzzy Controller-Based Approach in Adaptative Learning for Backgammon Game. In *MICAI 2005*, volume 3789 of *LNAI*, pages 224–233. Springer-Verlag Berlin Heidelberg, 2005.

[22] S. M. Golladay. Critical Text of Alfonso X, Libro de los juegos, transcribed by Sonja Musser Golladay. http://www.u.arizona.edu/ smusser/hsms.html, 2006.

[23] Gammon Village Inc. Backgammon game. http://www.gammonvillage.com/backgammon/, 1996.

[24] F. Alcalá. El juego de backgamon. http://www.pacosden.8k.com/regla_backgammon.html, 2001.

[25] B. Robertie. *Backgammon For Winners, 3rd Edition*. Cardoza, 2002.

[26] E. Heyken and M. B. Fischer. *The Backgammon Handbook*. Crowood Press (UK), 1990.

[27] J. H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, Michigan, 1975.

[28] D. E. Goldberg and Kalyanmoy Deb. *A Comparison of Selection Schemes Used in Genetic Algorithms*. In Gregory J. E. Rawlins, editor, Foundations fo Genetics Algorithms, 1991.

[29] A. K. De Jong. *An Analysis of the Behavior of a Class of Genetic Adaptative Systems*. PhD thesis, University of Michigan, 1975.

[30] L. B. Booker. *Intelligent Behavior as an Adaptation to the Task Environment*. PhD thesis, Logic of Computers Group, University of Michigan, 1982.

[31] A. Brindle. *Genetic Algorithms for Function Optimization*. PhD thesis, Department of Computer Science, University of Alberta, Edmonton, Alberta, 1981.

[32] J. E. Baker. Reducing Bias and Inefficiency in the Selection Algorithm. In *Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms*, pages 14–22. In J. J. Grefenstette, editor, 1987.

[33] J. E. Baker. Adaptative Selection in Methods for Genetic Algorithms. In *Proceedings of the First International Conference on Genetic Algorithms*, pages 101–111. In J. J. Grefenstette, editor, 1985.

[34] D. E. Goldberg. A note on Boltzmann Tournament Selection for Genetic Algorithms and Population-Oriented Simulated Annealing. *Complex Systems*, 4:445–460, 1990.

[35] A. Wetzel. *Evaluation og the Effectiveness of Genetics Algorithms in Combinatorial Optimization.* PhD thesis, University of Pittsburgh, Pittsburgh, Philadelphia, USA, 1983.

[36] D. Whitley. The GENITOR Algorithm and Selection Pressure: Why Rank-Based Allocation of Reproductive Trials is Best. In *Proceedings of the Third International Conference on Genetic Algorithms*, pages 116–121. In J. D. Schaffer, editor, 1989.

[37] J. D. Schaffer and A. Morishima. An Adaptative Crossover Distribution Mecanism for Genetic Algorithms. In *J. J. Grefenstette, editor, Genetic Algorithms and Their Applications*, Lawrence Erlbaum Associates, Hillsdale, New Jersey, pages 36–40. Proceedings of the Second International Conference on Genetic Algorithms, 1987.

[38] G. Rudolph. Convergence Analysis of Canonical Genetic Algorithm. *IEEE Transactions on Neural Networks*, 5:96–101, 1994.

[39] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley Publishing Company, Reading, Massachusetts, 1989.

[40] K. Krishnakumar. Micro-Genetic Algorithms for Stationary and non Stationary Function Optimization. In *SPIE Proceedings: Intelligent Control and Adaptative Systems*, pages 289–296, 1989.

[41] A. E. Eibend and J. E. Smith. *Introduction to Evolutionary Computing.* Springer, Berlin, 2003.

[42] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning Internal Representation by Error Propagation. *D. Rumelhart and J. McClelland, Eds., Parallel Distributed Processing, MIT Press, Cambridge, Mass*, 1, 1986.

[43] L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[44] O. Irineo-Fuentes, N. Cruz-Cortes, F. Rodriguez-Henriquez, D. Ortiz-Arroyo, and H. Legind-Larsen. GA-Gammon: A Backgammon Player Program Based on Evolutionary Algorithms. *Fifth Mexican International Conference on Artificial Intelligence (MICAI 06), IEEE CS Press, Tlaxcala, Tlax.*, 2006.