Sólo puede quedar uno: Evolución de Bots para RTS basada en supervivencia

A. Fernández-Ares[1], A.M. Mora[2], P. García-Sánchez[1], P.A. Castillo, and J.J. Merelo[1]

[1] Depto. de Arquitectura y Tecnología de Computadores
[2] Depto. de Lenguajes y Sistemas Informáticos
ETSIIT-CITIC, Universidad de Granada, España.
{antares,amorag,pablogarcia,pacv,jmerelo}@.ugr.es

Resumen Este artículo propone un algoritmo evolutivo para optimizar el comportamiento de bots (NPCs) que no requiere de una función de fitness explícita, usando en su lugar combates por pares (a modo de "justa") en los que sólo uno de los contendientes sobrevivirá. Este proceso hará las veces de mecanismo de selección del algoritmo, en el que sólo los ganadores pasarán a la siguiente generación del mismo. Se ha utilizado un algoritmo de Programación Genética, diseñado para generar motores de comportamiento para bots del conocido RTS Planet Wars. Este método tiene dos objetivos principales: en primer lugar, paliar el efecto que la naturaleza "ruidosa" de la función de fitness añade a la evaluación y, en segundo lugar, generar bots más generales (menos especializados) que los que se obtienen mediante algoritmos evolutivos en los que se usa siempre un contendiente común para evaluar los individuos. Más aún, la omisión de un proceso de evaluación explícito reduce el número de combates necesarios para evolucionar, lo que reduce a su vez el tiempo de cómputo del algoritmo. Los resultados demuestran que el método converge y que es menos sensible al ruido que otros métodos más tradicionales. Además de esto, con este algoritmo se obtienen bots muy competitivos en comparación con otros bots de la literatura.

1. Introducción y descripción del problema

Los Algoritmos Evolutivos (AEs) han sido aplicados a juegos durante mucho tiempo, a pesar de que la evaluación de posibles estrategias generadas es *ruidosa* [16] en el sentido de que existe una incertidumbre inherente sobre el verdadero *fitness* o calidad del *Bot* (NPC) que se esté evaluando. Esto es debido a que dicha evaluación depende de varios factores estocásticos: reglas del juego y estado actual, el comportamiento de los rivales o el estado inicial del juego, dado que todos ellos tendrán influencia en el resultado que pueda obtener el agente.

Planet Wars, el RTS (*Real-Time Strategy game*) que se propuso como banco de pruebas para la celebración del Google AI Challenge 2010[3], obviamente, también presenta este problema. Este juego ha sido utilizado por numerosos autores para el estudio de técnicas de Inteligencia Computacional (IC) en RTSs, tales como la generación automática u optimización de bots o mapas del juego [7,26,6,13].

[3] `http://planetwars.aichallenge.org/`

A modo de resumen, el objetivo del jugador es conquistar los planetas neutrales y los que posee el enemigo en un simulador "espacial". Cada jugador poseerá planetas (recursos) que producen naves (unidades) en función de una tasa de crecimiento. El jugador deberá controlar su flota de naves (sólo podrá actúar sobre ellas), enviándolas a otros planetas para conquistarlos o reforzarlos (en caso de pertenecer al jugador previamente). En el primer caso, dichas naves se estrellarán literalmente contra los planetas, reduciendo el número de naves o puntos necesarios para conquistarlos (en una relación de 1 por nave estrellada). Si este número llega a 0 el jugador pasará a dominar ese planeta, que generará naves para él a partir de entonces. El ganador será aquel que posea todos los planetas al final de la partida o aquel al que le resten unidades (si un jugador se queda sin naves pierde la partida).

Existen ciertas restricciones, como un límite de tiempo de un segundo para tomar decisiones, por lo que se podrían considerar *turnos*[4]; o también la imposibilidad de guardar y considerar acciones de turnos anteriores (memoria de acciones realizadas).

La Figura 1 muestra una captura de pantalla del juego.

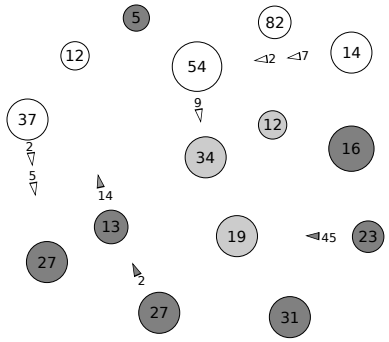

Figura 1. Captura de la simulación de un estado temprano del juego Planet Wars. Los planetas blancos pertenecen al jugador, los planetas grises oscuros pertenecen al oponente y los planetas grises claros no pertenecen aún a ningún jugador. Los triángulos representas flotas de naves. Los números (tanto en planetas como en flotas) representan el número de naves que lo componen. El tamaño de los planetas se refiere a la tasa de crecimiento (número de naves que genera por turno) que tiene asociada, de modo que será mayor cuanto más grande sea el planeta.

Un bot de PlanetWars se encargará de controlar toda la flota, realizando las acciones pertinentes sobre las naves para doblegar la flota enemiga y conquistar todos los planetas en liza.

Este juego presenta el problema mencionado antes en el cálculo del fitness [16]. De modo que, como medida habitual, se realizan diversos combates para evaluar a cada individuo (en mapas diferentes y contra diferentes enemigos), de modo que su fitness se obtiene como una media o sumatoria de los resultados obtenidos. De esta forma,

[4] Aunque usemos este término, se considera que el juego transcurre en tiempo real.

idealmente, se obtendría una medida más precisa (menos ruidosa) de la calidad de cada individuo. Pero, en realidad, no es aún una solución totalmente fiable dado que depende de diversos valores del propio bot, como el número de victorias obtenidas o de naves generadas en las batallas; así como de otros relacionados con el rendimiento del rival, que podría ser también un bot.

De modo que, incluso si obtuviésemos una evaluación estadísticamente significativa, el calculo del fitnes podría incluir una tendencia relacionada con el oponente seleccionado. Este problema está relacionado con el sobre-entrenamiento de la población para enfrentarse a un rival específico, es decir, los individuos aprenden a jugar excesivamente bien contra él y obtienen peores resultados contra otros enemigos [16,15].

Este artículo propone un AE co-evolutivo (AEC) [20] para mejorar la IA de bots de Planet Wars por medio de una evaluación implícita del fitness. Ésta se basará en la *supervivencia de los individuos*. Para ello, el proceso de selección se transformará en un *torneo* (o *justa*, para diferenciar el término del clásico torneo de los AEs), en el que sólo los ganadores sobrevivirán y pasarán a ser individuos (padres) de la siguiente generación. De este modo se evita el cálculo del fitness y, por lo tanto, la influencia del ruido que ésta función introduce se reduce. Además, con este enfoque se evitan algunos de los parámetros de configuración del algoritmo, como el número de combates o los rangos para las puntuaciones, así como la consideración de un rival en particular para combatir.

Este modelo es más cercano al proceso real de selección natural [4], en el que sólo los mejores sobreviven, de lo que lo están los AEs canónicos. De forma que el método propuesto se podría denominar un *algoritmo co-evolutivo competitivo* [22,11], en el que el fitness (implícito aquí) depende de una competición contra otros individuos.

La propuesta se ha implementado como un algoritmo de Programación Genética (PG) [12], debido a la flexibilidad que este modelo ofrece respecto a los Algoritmos Genéticos (AGs) [9], es decir, con PG se pueden crear nuevas reglas de comportamiento, mientras que el AG estaría centrado en la optimización de un conjunto ya existente. Además, esta técnica la hemos aplicado con buenos resultados en trabajos anteriores en esta misma línea [8,5].

Dado que el algoritmo se ejecuta en conjunción con Planet Wars, la justa se ha modelado como un combate en el juego. El ganador de la batalla pasa a la siguiente generación como padre de los siguientes bots, el perdedor se elimina de la población. Además, la consideración de todos los individuos como oponentes, en lugar de uno sólo, hace el entrenamiento (la evolución) más generalista, lo que producirá bots capaces de enfrentarse con éxito a un rango más amplio de rivales (aprenderán diversas estrategias).

Se han realizado varios experimentos, a fin de medir la convergencia en la evolución, así como la influencia del ruido en la misma, y se han comparado los resultados con los obtenidos por otros algoritmos y bots de la literatura. Nuestro objetivo es comprobar si el método propuesto es válido para evaluar individuos, si resulta menos sensible a la influencia del ruido y si los bots generados son suficientemente competitivos.

2. Conceptos preliminares y estado del arte

La evolución de motores de IA para controlar NPCs/Bots se ha convertido en una técnica muy exitosa en el entorno de los videojuegos. Existen dos enfoques principales: partir de un conjunto de reglas, cuyos parámetros o variables son optimizados *off-line* (antes de la partida real) por medio de AGs [7,17,6,3,10]; o bien usar PG para crear automáticamente el conjunto de reglas que compondrán dicho motor de IA partiendo de una serie de condiciones y acciones (antecedentes y consecuentes de las reglas) [8,5].

El problema en ambos casos suele ser la consideración de un bot específico para la evaluación de los potenciales individuos o la definición de una función de fitness que realmente pueda valorar el rendimiento de cada bot en la batalla, atendiendo, normalmente, a diversos factores.

Una posible forma de evitar estos problemas es el uso de AECs, en los que el comportamiento de un individuo depende del de otros individuos de la población. En el ámbito de los videojuegos, los AECs se empezaron a aplicar en juegos de tablero como Backgammon [21], o Go [23] hace muchos años. Además, en años posteriores, este enfoque se ha aplicado a videojuegos, dentro de la rama de IC. Por ejemplo Togelius et al. [24] estudiaron los efectos de la co-evolución en una población de controladores para un juego de carreras de coches; Cook et al. [2] presentaron una propuesta co-evolutiva para el diseño automático de niveles de un juego de plataformas; y recientemente, Cardona et al. [1] experimentaron con el rendimiento de un algoritmo competitivo para la evolución simultánea de controladores para Ms. PacMan y el grupo de Fantasmas simultáneamente.

Nuestro trabajo también se ha enfocado como un método co-evolutivo, que es competitivo en la selección de individuos para ser padres de la siguiente generación y, a su vez, cooperativo puesto que todos los individuos forman parte del mismo proceso evolutivo.

Centrándonos en los trabajos dentro del ámbito de los RTSs, Livingstone [14] comparó diferentes modelos de AI y propuso un entorno co-evolutivo de generación de los mismos, considerando diferentes niveles de AI (nivel de comandante, nivel de unidades, etc), de modo que era un enfoque cooperativo. El que se propone aquí se diferencia también en que el nivel de AI que se evoluciona es el mismo para todos los individuos.

Finalmente, Nogueira et al. [18] consideraron en una publicación reciente el uso de la llamada *Hall of Fame*, es decir, un conjunto de rivales que componen la élite de los oponentes hasta el momento, para evolucionar bots para el RTS *RobotWars*. Los mismos autores aplicaron una versión de dicho enfoque a Planet Wars [19]. En ella, describieron un algoritmo de aprendizaje similar al que proponemos, pero centrado en un subconjunto de individuos (la élite), que creemos que podría tener un efecto negativo en cuanto a la capacidad de generalización de los bots. Además, los autores aplican una función de fitness que depende de diversos parámetros, varios combates y medidas de puntuación adicionales.

La propuesta de este trabajo implementa un enfoque co-evolutivo basado en supervivencia, que evita el uso de un función de fitness explícita. El objetivo es intentar minimizar el efecto del ruido que introduciría dicha función [16]. Además, se evita el uso de parámetros adicionales y el uso de un rival (o rivales) específico en los combates que pudiera llevar a un sobre-entrenamiento contra él (o ellos).

3. Survival Bots

Esta sección describe el algoritmo propuesto para generar bots llamados *Survival-Bots*. En él se combina un algoritmo de PG [12] con diferentes políticas de selección y reemplazo, basadas en la supervivencia de los mejores y con un enfoque co-evolutivo.

3.1. Generación de Bots mediante PG

El algoritmo basado en Programación Genética (llamado *GPBot*) evoluciona un conjunto de parámetros que modela un árbol de decisión. Durante la evolución cada individuo de la población (un árbol) se evalúa. Para esto, el árbol que modela el motor de comportamiento de un agente, es colocado en un mapa en una partida de Planet Wars. Dependiendo de los resultados obtenidos, el agente (individuo) obtiene un valor fitness que se usa durante el proceso evolutivo.

Durante cada turno de la partida el árbol decidirá la mejor estrategia a seguir, seleccionando por cada planeta un objetivo y un porcentaje de naves a enviar. Estos árboles de decisión son árboles binarios de expresiones compuestas por dos diferentes tipos de nodos:

- *Decisión*: una expresión lógica formada por una variable, el operador $<$, y un número entre 0 y 1. Es el equivalente al concepto "primitiva" en el campo de la PG.
- *Acción*: una hoja del árbol (o sea, un "terminal"). Cada decisión es el nombre del método a llamar del planeta que ejecuta el árbol. Este método indica a qué planeta enviar un porcentaje de las naves disponibles (de 0 a 1).

Las decisiones, definidas por un experto humano, se basan en los valores de las distintas *variables* (también definidas por el experto) que son computadas considerando algunas otras variables del juego.

- *myShipsEnemyRatio*: Relación entre las naves del jugador y las naves del enemigo.
- *myShipsLandedFlyingRatio*: Relación entre las naves del jugador que vuelan y están aterrizadas.
- *myPlanetsEnemyRatio*: Relación entre el número de planetas del jugador y del enemigo.
- *myPlanetsTotalRatio*:Relación entre el número de planetas del jugador y del total (incluyendo los del enemigo y los neutrales).
- *actualMyShipsRatio*: Relación entre el número de naves en el planeta específico que evalúa el árbol y el total de naves del jugador.
- *actualLandedFlyingRatio*: Relación entre el número de naves aterrizadas y volando desde el planeta específico que evalúa el árbol, y el total de naves del jugador.

Finalmente, las posibles *acciones*, según conocimiento experto, son:

- *Attack Nearest (Neutral—Enemy—NotMy) Planet*: El objetivo es el planeta más cercano. *NotMy* se refiere a un planeta que no pertenezca al jugador, es decir, uno de los otros dos: enemigo o neutral.

- *Attack Weakest (Neutral—Enemy—NotMy) Planet*: El objetivo es el planeta con menos naves.
- *Attack Wealthiest (Neutral—Enemy—NotMy) Planet*: El objetivo es el planeta con más tasa de crecimiento.
- *Attack Beneficial (Neutral—Enemy—NotMy) Planet*: El objetivo es el planeta más beneficioso, es decir, el que tiene mayor tasa de crecimiento dividido por el número de naves que alberga.
- *Attack Quickest (Neutral—Enemy—NotMy) Planet*: El objetivo es el planeta más fácil de conquistar: el menor producto entre la distancia del planeta que ejecuta el árbol y el número de naves en el planeta objetivo.
- *Attack (Neutral—Enemy—NotMy) Base*: El objetivo es el planeta con más naves (es decir, la base).
- *Attack Random Planet*. Atacar un planeta aleatorio.
- *Reinforce Nearest Planet*: Reforzar el planeta más cercano al que ejecuta el árbol.
- *Reinforce Base*: Reforzar al planeta con más naves del jugador.
- *Reinforce Wealthiest Planet*: Reforzar al planeta del jugador con mayor tasa de crecimiento.
- *Do nothing*. No hacer nada.

Un ejemplo de un árbol de decisión posible se muestra a continuación. Este ejemplo tiene un total de 5 nodos, con dos decisiones y tres acciones, con una profundidad de tres niveles.

```
if(myShipsLandedFlyingRatio < 0.796)
   if(actualMyShipsRatio < 0.201)
      attackWeakestNeutralPlanet(0.481);
   else
      attackNearestEnemyPlanet(0.913);
else
   attackNearestEnemyPlanet(0.819);
```

El comportamiento del bot se explica en el Algoritmo 1.

3.2. Selección por supervivencia

El algoritmo presentado en la sección anterior se ha combinado con una *evaluación implícita del fitness* que se ocupará de la selección y el reemplazo. Dicha evaluación es, en esencia, un combate entre individuos en un determinado mapa. Hemos denominado a este enfrentamiento *justa* (para diferenciarlo del clásico torneo de los AEs). De forma que la selección de padres para la siguiente generación se consigue mediante la resolución de estos combates, siendo el ganador (el superviviente) el elegido para continuar en la población y siendo el perdedor eliminado de la misma.

De esta forma, el proceso de selección intenta fomentar la supervivencia de los mejores individuos, paliando en cierta medida el ruido que añaden las funciones de evaluación explícitas (o numéricas) [16]. De modo que los individuos que no son capaces de ganar un combate son fuertemente penalizados y eliminados de la población. Aún así, seguirá habiendo ruido, el cuál es intrínseco al problema en sí. Es decir, un individuo

Algorithm 1 Pseudocódigo del agente propuesto. El mismo árbol se ejecuta durante toda la ejecución del agente.

```
//Al principio de la ejecución el agente recibe el árbol
árbol ← leerÁrbol()
while el juego no termine do
  // iniciar el turno
  calcularPlanetasGlobales() // p.e. Base o Base Enemiga
  calcularRatiosGlobales() // p.e. myPlanetsEnemyRatio
  for cada p en planetas del jugador do
    calcularPlanetasLocales(p) // p.e. NearestNeutralPlanet a p
    calcularRatiosDePlanetas(p) //p.e. actualMyShipsRatio
    ejecutarÁrbol(p, árbol) // Enviar un porcentaje de naves al destino
  end for
end while
```

con peores condiciones que otro podría ganar un combate por causas ajenas a su buen rendimiento, como errores del rival o condiciones más favorables en el desarrollo del juego. Aunque Planet Wars parte siempre de escenarios totalmente equivalentes para los contendientes. En cualquier caso, la presencia de ruido permitirá, a su vez, añadir diversidad a la población, lo cual siempre es beneficioso en un algoritmo de optimización combinatoria como este.

En este enfoque hablamos de "iteración" como sinónimo de generación, dado que no se trata de un proceso evolutivo clásico, como se explicará más adelante.

3.3. Reemplazo de los perdedores

Hemos implementado la política de reemplazo del enfoque estacionario clásico de los AEs [25]. De forma que una gran parte de la población pasa a la siguiente generación y otra parte es sustituida. Este enfoque pretende fomentar la explotación en la búsqueda, a fin de mejorar el factor de convergencia del método. El objetivo es reducir la exploración que es más sensible en un espacio de búsqueda ruidoso como este.

De modo que el algoritmo propuesto realiza dos combates (justas) por generación. Los contendientes se seleccionan de forma aleatoria entre los individuos de la población, asegurando simplemente que no se eligen los mismos bots para los dos combates.

Los dos ganadores pasan a ser los padres de la *operación de cruce*, que genera dos nuevos hijos, que son además mutados e incluidos en la población en el lugar de los individuos que hayan perdido el combate. Hemos considerado cruce de sub-árboles y mutación a nivel de nodo, dado que con ellos obtuvimos buenos resultados en trabajos previos [5].

Este enfoque presenta un componente de aleatoriedad mayor que el AE estacionario clásico, debido a la falta de una función de fitness que asigne un valor representativo a cada individuo. La selección aleatoria entre todos los individuos aumenta la probabilidad de que los malos individuos salgan de la población, lo que reduciría también el ruido presente en ésta. Este será un factor muy relevante, como se demostrará en los experimentos realizados.

El Algoritmo 2 muestra la combinación del algoritmo de PG propuesto con la evaluación implícita del fitness mencionada y los mecanismos de selección y reemplazo.

Algorithm 2 Pseudocode of the proposed SurvivalBot.

```
poblacion ← inicializarPoblacion()
while criterio de terminación no cumplido do
  descendientes,perdedores,seleccionados ← {}
  /* Se eligen dos contendientes aleatorios para la justa */
  contendientes ← seleccionarContendientes(poblacion)
  /* Los contendientes luchan y se obtiene un ganador y un perdedor */
  ganador1,perdedor1 ← justa(contendientes)
  /* Los bots selecionados no participan en más combates */
  seleccionados ← seleccionados + ganador1 + perdedor1
  /* Contendientes de la segunda justa */
  contendientes ← selectContenders(poblacion)
  /* Los contendientes luchan y se obtiene un ganador y un perdedor */
  ganador2,perdedor2 ← justa(contendientes)
  seleccionados ← seleccionados + ganador2 + perdedor2
  /* Se guarda referencia a los perdedores */
  perdedores ← perdedores + perdedor1 + perdedor2
  /* PROCESO EVOLUTIVO */
  hijo1,hijo2 ← cruce(ganador1,ganador2);
  hijo1,hijo2 ← mutacion(hijo1,hijo2)
  descendientes ← descendientes + hijo1 + hijo2
  /* Reemplazo de perdedores */
  poblacion ← poblacion - perdedores
  poblacion ← poblacion + descendientes
end while
```

4. Experimentos y resultados

Se han realizado varios experimentos para estudiar diferentes aspectos de la propuesta, pero debemos señalar que el principal objetivo no es la generación de los mejores bots (los más competitivos) posibles a toda costa, sino, en primera instancia, probar la validez de este método co-evolutivo basado en selección por supervivencia o justa. De modo que comprobaremos que el algoritmo converge y que el ruido tiene una influencia menor que en otros casos. Posteriormente analizaremos la calidad de los bots obtenidos para demostrar que son bots suficientemente buenos. Éstos se podrán mejorar en un futuro usando el mismo método pero con configuraciones más adecuadas para ello (como un ajuste de parámetros más fino, ejecutarlos durante más generaciones, etc).

El conjunto de parámetros considerado en nuestro algoritmo co-evolutivo de PG (Co-PG), SurvivalBot, se muestra en la Tabla 1. Éstos son los mismos utilizados previamente por los autores en el trabajo [8], GPBot, con los que se obtenían buenos bots. Además, usamos la misma configuración porque GPBot se usará como base comparativa en los experimentos. De modo que, para hacer más justa la comparativa, se han fijado

4000 iteraciones en SurvivalBot (2 combates por iteración), dado que en GPBot se realizaron 8000 combates (32 individuos * 5 combates por evaluación * 50 generaciones). Se han considerado 5 mapas representativos (con configuraciones muy diferentes entre sí) que se eligen de forma aleatoria antes de cada combate.

Se han realizado 30 ejecuciones en cada caso, para obtener resultados estadísticamente significativos.

Nombre Parámetro	*Valor*
Tamaño población	32
Inicialización	Aleatoria (árboles de 3 niveles)
Tipo de Cruce	Cruce de sub-árbol
Tasa de Cruce	0,5
Máximo número de turnos por combate	1000
Mutación	Mutación de 1 nodo
Tasa de Mutación	0,25
Selección	Justa (torneo entre 2)
Reemplazo	Estacionario
Criterio de parada	4000 iteraciones
Profundidad máxima de los árboles	7
Ejecuciones	30
Mapas usados en la justa	1 aleatorio elegido entre los mapas #76 #69 #7 #11 #26

Tabla 1. Conjunto de parámetros usado en los experimentos.

4.1. Análisis de las ejecuciones

En este primer conjunto de experimentos analizaremos la convergencia del método propuesto, dado que se espera que su comportamiento, aún sin función explícita de fitness, sea similar al de otros AEs. El problema radica en que es difícil mostrar esta convergencia sin contar con un valor numérico para el fitness. Por ello, una vez terminada la ejecución, se ha analizado la población de cada generación haciendo uso de una función *Score*, que habíamos definido y usado en artículos anteriores para valorar el rendimiento de un bot en el juego. Ésta se basa en el desarrollo de varios combates, y la consideración del número de victorias obtenidas, turnos requeridos para ganar y turnos resistidos en caso de haber perdido.

De modo que cada individuo i perteneciente a la población de cada generación/iteración se ha enfrentado con GPBot, obteniendo un Score, definido como:

$$Score_i = \alpha + \beta + \gamma \tag{1}$$

Donde

$$\alpha = v, \alpha \in [0, N] \tag{2}$$

$$\beta = N \times \frac{t_{win} + \frac{1}{N \times t_{MAX}+1}}{\frac{t_{win}}{v+1} + 1}, \quad \beta \in [0, N], \quad t_{win} \in [0, N \times t_{MAX}] \tag{3}$$

$$\gamma = \frac{t_{defeated}}{N \times t_{MAX} + 1}, \quad \gamma \in [0, 1], \quad t_{defeated} \in [0, N \times t_{MAX}] \tag{4}$$

Considerando el número de combates realizados (N), el número de victorias contra GPBot (v), el número total de turnos usados para ganar a GPBot (t_{win}), el número total de turnos aguantados, si ha sido vencido ($t_{defeated}$) y el número máximo de turnos que dura un combate ($t_{MAX} = 1000$). Esta función tiende a favorecer las victorias frente a los turnos. Cada individuo se ha enfrentado 3 veces en 10 mapas diferentes: los 5 usados durante la evolución, llamados *mapas entrenados* y otros 5 nuevos (*mapas no entrenados*), por tanto $N = 30$.

La Figura 2 muestra los *boxplots* del *Score* obtenido por toda la población de las 30 ejecuciones en cada iteración. Agrupados según los mapas conocidos (con los que se ha evolucionado) a la izquierda, o los mapas nuevos a la derecha. Como se puede ver la tendencia es creciente, como era deseable. Respecto a los resultados que se muestran ligeramente mejores en los mapas no conocidos (no entrenados previamente), este hecho se debe a la propiedad del algoritmo propuesto de fomentar la generalización, evitando el poco deseable efecto de sobre especialización. De forma que los bots obtenidos no "aprenden" a jugar en mapas específicos ni contra rivales específicos, por lo que su rendimiento contra GPBot en este caso es relativamente bueno en todos los individuos.

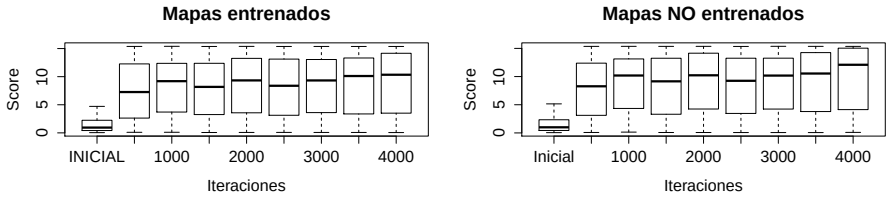

Figura 2. *Score* obtenido en el enfrentamiento contra el mejor GPBot de todos los SurvivaBots obtenidos en cada generación/iteración de las 30 ejecuciones.

Este estudio se complementa en primer lugar con las dos gráficas que se muestran en la Figura 3. En ellas se presenta el porcentaje de individuos (normalizado entre 0

y 1) que gana un determinado número de combates (de 0 a 30) contra GPBot de los que hay en la población inicial (izquierda) y en la final (derecha). Como se puede ver, los individuos de las 30 poblaciones iniciales en las ejecuciones tienden a ganar muy pocos o ningún combate, mientras que en las poblaciones finales, el porcentaje de bots capaces de vencer la mayoría de las veces e incluso todos los combates se incrementa en gran medida. Lo que demuestra que las poblaciones, en general, han evolucionado positivamente y han conseguido bots cada vez más competitivos durante el proceso evolutivo.

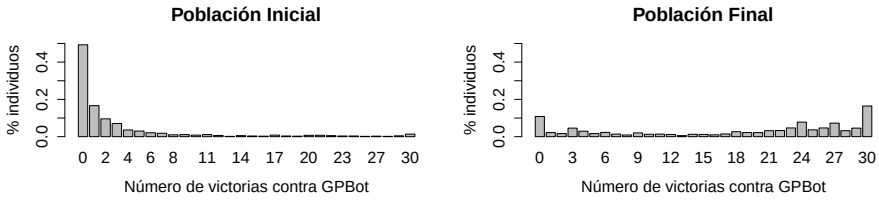

Figura 3. Histograma del número de victorias contra GPBot para todos los individuos de las poblaciones iniciales y finales de las 30 ejecuciones, en los 30 combates (10 mapas * 3 enfrentamientos)

Por último, si estudiamos la *edad* (número de generaciones que sobrevive un individuo) de los bots evolucionados, podremos entender mejor la dinámica del proceso. Para ello, la Figura 4 muestra las edades de los individuos en una ejecución. Como se puede ver, la edad media se mantiene en torno a un valor estable durante toda la ejecución, lo que nos indica que la población está continuamente mejorando, de modo que los hijos son capaces de vencer a sus padres en pocas generaciones. Existen valores extremos, debidos, sin duda, al componente aleatorio que hay en la selección de los padres, por el que algunos individuos combatirán muy pocas veces (o ninguna) en una ejecución.

A tenor de lo visto en estos experimentos, consideramos que la evolución del algoritmo es la adecuada. Pasaremos a analizar la influencia del ruido en este proceso.

4.2. Análisis de ruido

En este estudio se compararán los mejores 30 bots obtenidos por GPBot con los mejores 30 SurvivalBots (de las 30 ejecuciones). Para elegir el mejor SurvivalBot de cada ejecución, se ha realizado un torneo *todos contra todos* entre los bots de la última generación. El bot que ha ganado más combates ha sido considerado como el mejor. Hemos aplicado este método para evitar el uso de una función Score sobre bots evolucionados sin fitness, a fin de evitar las desventajas que esta función conlleva, como hemos explicado a lo largo del artículo.

Éstos se han enfrentado contra el mejor rival de nuestros trabajos, el bot experto o especializado ExpGeneBot [6], capaz de adaptar sus estrategias a las características del

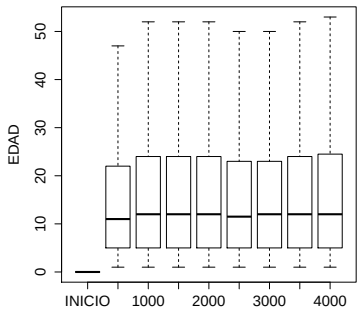

Figura 4. *Boxplots* de la edad (número de generaciones/iteraciones que sobrevive un bot) de la población en una ejecución.

mapa en el que se desarrolla el combate. Dichos enfrentamientos se han realizado en los mismos 10 mapas considerados en el experimento anterior, en cada uno de los cuales se han realizado 30 combates y calculado el Score de los bots implicados, siguiendo la Ecuación 1.

A fin de comprobar la influencia que ha tenido el ruido en la generación de los SurvivalBots, hemos calculado un *factor de ruido* para cada bot, el cual se ha obtenido como la diferencia entre el máximo y el mínimo de los Scores obtenidos en los 30 combates que ha llevado a cabo cada bot en cada mapa.

La Figura 5 muestra los *boxplots* de los 30 GPBots y SurvivalBots. Según la definición del factor de ruido que hemos propuesto, una distancia grande entre valores significará una mayor incertidumbre en los resultados. Las gráficas muestran que los SurvivalBots se comportan mejor en este sentido, presentando una varianza menor en resultados contra un rival de gran dificultad y capaz de autoadaptar su comportamiento en función del mapa y la partida. De modo que se puede concluir que los bots obtenidos mediante el método descrito en este artículo son más fiables o "robustos" en términos de comportamiento y rendimiento.

Finalmente, en la siguiente sección analizaremos la calidad o competitividad de los bots obtenidos con el método propuesto, ya que ese es el objetivo final de esta evolución.

4.3. Análisis de los bots generados

En este experimento todos los SurvivalBots obtenidos al final de las ejecuciones se han enfrentado a otros bots del estado del arte. Para ello, en primer lugar se han seleccionado los 30 mejores, como se hizo en el experimento anterior. Dichos bots han sido enfrentados contra un bot básico (BullyBot) y 4 del estado del arte. Los bots y sus configuraciones se pueden ver en la Tabla 2.

Los enfrentamientos se han realizado en los 100 mapas de ejemplo que proporcionó Google en el campeonato, los cuales no han sido usados durante la evolución.

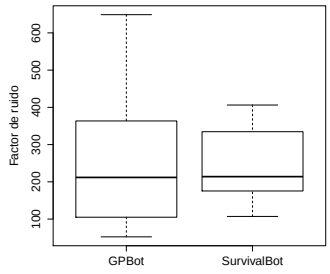

Figura 5. *Factor de ruido*, calculado como el máximo score obtenido - el mínimo en cada mapa, para cada uno de los 30 GPBots y SurvivalBots. 30 combates por mapa y 10 mapas diferentes.

Bot	Referencia	*Combates en el entrenamiento*	*Max. Turnos*
BullyBot	Web del Google AI Challenge	No	No
SurvivalBot	propuesto aquí	8000	1000
GeneBot	[16]	32000	1000
ExpGeneBot	[6]	32000	1000
GPBot	[8]	8000	1000
HoFBot	[19]	180000	500

Tabla 2. Bots disponibles en la literatura que hemos usado para medir la calidad de los SurvivalBots

La Figura 6 muestra los *boxplots* del porcentaje de victorias obtenido por los SurvivalBots contra los demás rivales (no se consideran los empates).

El resultado más interesante es que GPBot es ampliamente superado, dado que el número de combates en la evolución es el mismo y la base de sus motores de IA, basada en PG es similar. El HoFBot, que también se obtuvo usando un algoritmo co-evolutivo, ha sido vencido más del 50 % de los combates por la mayoría de los mejores SurvivalBots. Sin embargo, bots ampliamente entrenados (4 veces más evaluaciones) y basados en una IA diseñada por un experto, como son GeneBot y ExpGeneBot, han resultado difíciles de vencer, como era de esperar.

En cualquier caso, este es un punto de partida para SurvivalBot, que tiene mucho margen de mejora, empezando por un mayor número de iteraciones/combates en su entrenamiento, que llevarían a obtener bots más competitivos.

5. Conclusiones y trabajo futuro

Este artículo presenta una implementación de un algoritmo de Programación Genética (PG) co-evolutivo simple para la generación de bots (NPCs) para RTSs. Éste método propone la omisión de la selección basada en fitness en el proceso evolutivo. En su

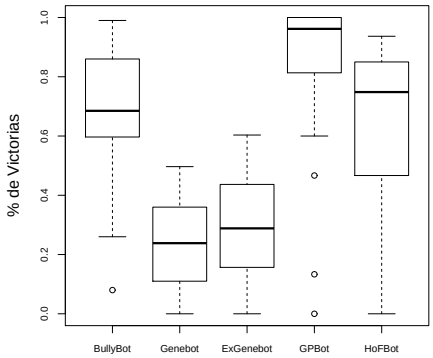

Figura 6. Porcentaje de victorias al enfrentar los mejores 30 SurvivalBots contra otros bots de la literatura.

lugar, se simula una evolución de forma más natural, puesto que se basa en la mera supervivencia de los individuos, que se enfrentan en combates contra otros en los que sólo el ganador pasa a la siguiente generación.

Esta propuesta se ha aplicado a la creación y mejora de reglas de comportamiento para la IA de un bot que juegue a Planet Wars. De forma que la selección se ha modelado como in combate en el juego, que hemos denominado *justa*, en el que el ganador será padre de la siguiente descendencia, mientras que el perdedor será reemplazado por dicha descendencia.

Según se ha demostrado en los experimentos, el algoritmo propuesto ofrece tres beneficios principales:

- Genera bots más generalistas en su comportamiento, no especializados en un rival en concreto, dado que no utiliza rivales específicos en sus evaluaciones, sino miembros de la propia población.
- Se ve menos afectado por el ruido intrínseco a las funciones de evaluación en el entorno de los videojuegos, debido a la gran presión selectiva que introduce el hecho de que un sólo combate perdido por un bot haría que éste se eliminara de la población. De modo que es muy difícil que bots que no sean realmente buenos sobrevivan.
- Requiere menos combates en la evaluación (sólo uno), en contraposición a los múltiples combates que se suelen realizar en las evaluaciones de otras propuestas. Esto hace que se reduzca el tiempo de cómputo del algoritmo.

Los bots obtenidos, llamados SurvivalBots, han sido enfrentados contra otros bots del estado del arte, obteniendo muy buenos resultados en general, considerando que el menor entrenamiento de los bots obtenidos en esta propuesta.

Como líneas de trabajo futuro se plantean muchas, ya que este enfoque es reciente en nuestras investigaciones. En primer lugar nos centraremos en obtener bots más competitivos y complejos, añadiendo más variedad de condiciones y acciones al algoritmo de PG, como por ejemplo distancias entre planetas. En la misma línea se harán pruebas sin limitar el tamaño máximo de los árboles.

Agradecimientos

Este trabajo ha sido financiado en parte por los proyectos: EPHEMECH (TIN2014-56494-C4-3-P) y KNOWAVES (TEC2015-68752-R), Ministerio de Economía y Competitividad, Fondos FEDER, PROY-PP2015-06 (Plan Propio 2015 UGR), y el proyecto MOSOS (PRY142/14), financiado por la Fundación Pública Andaluza Centro de Estudios Andaluces en la IX Convocatoria de Proyectos de Investigación.

Referencias

1. Cardona, A., Togelius, J., Nelson, M.: Competitive coevolution in Ms. Pac-Man. In: Proceedings of the 2013 IEEE Congress on Evolutionary Computation. pp. 1403–1410 (2013)
2. Cook, M., Colton, S., Gow, J.: Initial results from co-operative co-evolution for automated platformer design. In: Applications of Evolutionary Computation, EVOApplications 2012, LNCS 7248. pp. 194–203 (2012)
3. Cotta, C., Fernández-Leiva, A.J., Sánchez, A.F., Lara-Cabrera, R.: Car setup optimization via evolutionary algorithms. In: Rojas, I., Joya, G., Cabestany, J. (eds.) Advances in Computational Intelligence, LNCS, vol. 7903, pp. 346–354. Springer Berlin Heidelberg (2013)
4. Darwin, C.: On the Origin of Species by Means of Natural Selection. Murray, London (1859)
5. Esparcia-Alcázar, A., Moravec, J.: Fitness approximation for bot evolution in genetic programming. Soft Comput. 17(8), 1479–1487 (2013)
6. Fernández-Ares, A., García-Sánchez, P., Mora, A.M., Merelo, J.J.: Adaptive bots for real-time strategy games via map characterization. In: 2012 IEEE Conference on Computational Intelligence and Games, CIG 2012. pp. 417–721. IEEE (2012)
7. Fernández-Ares, A., Mora, A.M., Merelo, J.J., García-Sánchez, P., Fernandes, C.: Optimizing player behavior in a real-time strategy game using evolutionary algorithms. In: Evolutionary Computation, 2011. CEC '11. IEEE Congress on. pp. 2017–2024 (June 2011)
8. García-Sánchez, P., Fernández-Ares, A., Mora, A.M., Castillo, P.A., González, J., Merelo, J.: Tree depth influence in genetic programming for generation of competitive agents for RTS games. In: Applications of Evolutionary Computation - EvoApplications 2012, Granada, Spain, April 23-25, 2014, Proceedings. LNCS, Springer (2014)
9. Goldberg, D.E.: Genetic Algorithms in search, optimization and machine learning. Addison Wesley (1989)
10. Jaśkowski, W., Krawiec, K., Wieloch, B.: Winning ant wars: Evolving a human-competitive game strategy using fitnessless selection. In: O'Neill, M.e.a. (ed.) Genetic Programming, LNCS, vol. 4971, pp. 13–24 (2008)
11. Kim, Y., Kim, J., Kim, Y.: A tournament-based competitive coevolutionary algorithm. Applied Intelligence 20(3), 267–281 (2004)
12. Koza, J.R.: Genetic Programming: On the programming of computers by means of natural selection. MIT Press, Cambridge, MA (1992)

13. Lara-Cabrera, R., Cotta, C., Fernández-Leiva, A.J.: On balance and dynamism in procedural content generation with self-adaptive evolutionary algorithms. Natural Computing 13(2), 157–168 (2014)
14. Livingstone, D.: Coevolution in hierarchical ai for strategy games. In: IEEE Symposium on Computational Intelligence and Games (CIG'05). IEEE (2005)
15. Merelo-Guervós;, J.J.: Using a Wilcoxon-test based partial order for selection in evolutionary algorithms with noisy fitness. Tech. rep., GeNeura group, university of Granada (2014), available at http://dx.doi.org/10.6084/m9.figshare.974598
16. Mora, A., Fernández-Ares, A., Guervós, J.M., García-Sánchez, P., Fernandes, C.: Effect of noisy fitness in real-time strategy games player behaviour optimisation using evolutionary algorithms. Journal of Computer Science and Technology 27(5), 1007–1023 (2012)
17. Mora, A.M., Fernández-Ares, A., Merelo, J.J., García-Sánchez, P.: Dealing with noisy fitness in the design of a RTS game bot. In: Proc. Applications of Evolutionary Computing: EvoApplications 2012. pp. 234–244. Springer, LNCS, vol. 7248 (April 2012)
18. Nogueira, M., Cotta, C., Fernández-Leiva, A.J.: An analysis of hall-of-fame strategies in competitive coevolutionary algorithms for self-learning in rts games. In: Learning and Intelligent Optimization - 7th International Conference, LION 7, Catania, Italy, January 7-11, 2013, Revised Selected Papers. LNCS, vol. 7997, pp. 174–188 (2013)
19. Nogueira, M., Cotta, C., Fernández-Leiva, A.J.: Virtual player design using self-learning via competitive coevolutionary algorithms. Natural Computing 13(2), 131–144 (2014)
20. Paredis, J.: Coevolutionary computation. Artif. Life 2(4), 355–375 (Aug 1995)
21. Pollack, J.B., Blair, A.D.: Co-evolution in the successful learning of backgammon strategy. Machine Learning 32, 225–240 (1998)
22. Rosin, C.D., Belew, R.K.: New methods for competitive coevolution. Evol. Comput. 5(1), 1–29 (Mar 1997)
23. Runarsson, T.P., Lucas, S.M.: Co-evolution versus self-play temporal difference learning for acquiring position evaluation in smallboard go. IEEE Transactions on Evolutionary Computation 9(6), 628–640 (2005)
24. Togelius, J., Burrow, P., Lucas, S.M.: Multi-population competitive co-evolution of car racing controllers. In: Proceedings of the IEEE Congress on Evolutionary Computation. pp. 4043–4050 (2007)
25. Whitley, D., Kauth, J.: GENITOR: A different genetic algorithm. In: Proceedings of the 1988 Rocky Mountain Conference on Artificial Intelligence. pp. 118–130. Computer Science Department, Colorado State University (1988)
26. Ziółko, B., Kruk, M.: Automatic reasoning in the planet wars game. Annales UMCS, Informatica 12(1), 39–45 (2012)