UNIVERSIDAD COMPLUTENSE DE MADRID

MÁSTER DE INGENIERÍA DE SISTEMAS Y CONTROL

PROYECTO FINAL DE MÁSTER

BÚSQUEDA DE OBJETIVOS ESTÁTICOS EN TIEMPO MÍNIMO MEDIANTE PROGRAMACIÓN GENÉTICA

Alumno: Fernando Pilar Arce de la Plaza

Directores: José Antonio López Orozco y Eva Besada Portas

Curso 2014-2015

Convocatoria: Febrero

Índice de Contenido

Índice de Figuras

Resumen

Dentro de las investigaciones de búsqueda de objetivos bajo incertidumbre se han desarrollado varios sistemas capaces de optimizar las rutas de los agentes encargados de detectar dichos objetivos, bien sea maximizando la probabilidad de detección o minimizando el tiempo de búsqueda, siendo la combinación de ambos objetivos de resolución compleja.

En estos problemas no se conoce la posición exacta de los objetivos (existe incertidumbre) por lo que se divide la región de búsqueda en celdas a las que se asigna una probabilidad de que un objetivo se encuentre en cada una de ellas. Esta probabilidad se obtiene de estudios sobre el área de búsqueda, los factores físicos que puedan afectar a la posición del objetivo como puedan ser corrientes en zonas marinas o vientos, indicaciones de testigos que vieron desaparecer el objeto que se busca, u otro tipo de información que permita delimitar la posición del objeto a buscar. En el caso de usar sensores ideales para detectar los objetos, la probabilidad de las celdas visitadas se anula al ser directamente observadas por un agente que aún no ha detectado el objetivo. En el caso de usar sensores reales, la incertidumbre asociada a los sensores hace que la probabilidad de las celdas se modifique según las posibles observaciones que puedan realizar los agentes sobre la celda.

Se han aplicado diferentes técnicas para solucionar este tipo de problemas probabilísticos, entre las que se encuentran heurísticos bayesianos, algoritmos estocásticos como el de Entropía Cruzada (CEO) o algoritmos genéticos, entrenamiento de redes neuronales, y otras técnicas de aprendizaje. En el presente trabajo se analiza la viabilidad de utilizar Programación Genética (PG) para resolver el problema de búsqueda en tiempo mínimo, de tal forma que se obtenga un programa que genere las señales de control sub-óptimas utilizadas por el programa controlador responsable de guiar a los agentes hacia el objetivo. En el mejor de los casos, para evitar la optimización de un programa por cada mapa de probabilidad y obtener de estas formas las señales de control de un modo rápido, la PG deberá obtener un programa genérico, valido para todo mapa o para un tipo de mapas.

El programa generado por la PG estará formado por diferentes elementos condicionales que realicen cálculos sobre el mapa de probabilidad y por diferentes tipos de movimientos básicos que ejecute el agente en función de los valores obtenidos por los elementos condicionales. Dicho de otra forma, la PG combinará un conjunto de condiciones y movimientos, especialmente definidos para el problema de búsqueda en este trabajo, en un programa que, al ser ejecutado, generará las órdenes de control que guíe a los agentes durante la búsqueda.

Se valorará la calidad del controlador obtenido comparándolo con CEO, que será tomado como referente con el que evaluar los resultados de PG. Se realizará un análisis estadístico de los resultados obtenidos por ambos métodos, valorando el porcentaje de mapas en los que el programa generado por PG puede considerarse como un método aceptable. Además, se mostrarán algunas soluciones en las que se observan comportamientos particulares de los programas producidos por PG y que permiten proponer opciones de mejora del método. También se compararán los tiempos que PG y GEO necesitan para obtener, respectivamente, un programa genérico para cualquier mapa Finalmente, se realizarán algunas propuestas para mejorar los resultados a partir de los comportamientos detectados.

CAPÍTULO 1. Introducción

En el Departamento de Arquitectura de Computadores y Automática de la Universidad Complutense se han desarrollado diferentes trabajos que abordan el problema de búsqueda en tiempo mínimo, con el objeto de desarrollar un sistema de control multi-agente que genera las acciones que guían a vehículos no tripulados con el objetivo de encontrar náufragos. En la Tesis de Pablo Lanillos (Lanillos Pradas, 2013) se diseña un sistema completo, basado en un algoritmo de optimización de Entropía Cruzada (Cross Entropy Optimization, CEO[1]) y diferentes funciones objetivo probabilísticas, en el que se optimiza, para cada escenario y mapa de probabilidad, las señales de control que se deben aplicar a los agentes. En el Trabajo Fin de Máster de Judith Manso (Manso Vergara, 2013), en un escenario simplificado con un mapa de probabilidades estático con sensor ideal[2] y un único agente, se obtiene un controlador genérico entrenando una red neuronal con las rutas sub-optimas obtenidas por el algoritmo de optimización de entropía cruzada propuesto en la Tesis de Pablo Lanillos (Lanillos Pradas, 2013), sobre diferentes tipologías de mapa. En el Trabajo Fin de Máster de Francisco Javier Yañez (Zuluaga, 2014), se utilizan la funciones objetivo propuestas en la Tesis de Pablo Lanillos (Lanillos Pradas, 2013) para obtener las trayectorias de los agentes ante cada mapa de probabilidades por medio de algoritmos de optimización bayesiana y genéticos.

Buscando una alternativa a estos trabajos, y con el fin de encontrar un controlador que, una vez optimizado, actúe de un modo rápido sobre los mapas, el presente trabajo pretende producir, para el escenario simplificado del Trabajo Fin de Master de Judith Manso (Manso Vergara, 2013), las señales de control sub-óptimas obtenidas a partir de las rutas generadas por un programa optimizado mediante PG utilizando las funciones objetivo propuestas en la Tesis de Pablo Lanillos (Lanillos Pradas, 2013).

La ventaja de utilizar un programa para la generación de rutas es que su ejecución es rápida: una vez obtenido el código tras el proceso evolutivo de entrenamiento con diferentes mapas, solo hay que ejecutar dicho programa suministrándole como entrada el mapa de probabilidades y la posición del agente.

1.1 Objetivos

El objetivo es introducir la PG en la metodología de búsqueda de rutas sub-óptimas valorando las dificultades encontradas, los retos, los tiempos con los que se optimiza, y la calidad de los resultados obtenidos en comparación con otros métodos empleados en los trabajos ya comentados del Departamento de Arquitectura de Computadores y Automática.

Como se ha mencionado, si se determina que PG es de utilidad para este tipo de problemas, su valor añadido consiste en la rapidez de ejecución una vez obtenido el controlador por optimización. Esto hace que los tiempos de optimización no sean importantes, puesto que es

[1] CEO es un proceso iterativo de aprendizaje donde, en vez de calcular las mejores acciones directamente, se aprende la mejor distribución de las mismas de la que se obtienen muestras sub-óptimas.

[2] En este escenario, las probabilidades no se ven modificadas por las condiciones físicas del entorno, actualizándose únicamente aquellas celdas que son observadas (detectadas o no detectadas) desde la posición del agente en cada instante.

un proceso que se puede realizar de modo previo a su aplicación. De esta forma, el objetivo final de la PG se convierte en generar un programa, o un conjunto de programas, válido para obtener las trayectorias del agente sobre diferentes tipos de mapa.

1.2 Antecedentes

La Teoría de Búsqueda trata de cómo distribuir los recursos de la forma más eficaz con el fin de encontrar algo que se desconoce dónde está, pero de lo que se tiene una idea acerca de dónde podría estar y de cómo se mueve. Es una teoría genérica con la que se pueden buscar todo tipo de objetivos, como pueden ser un submarino en el Pacífico, un misil Scud en el desierto, un artefacto explosivo improvisado en una carretera, las llaves del coche en un estacionamiento gigante, o un paciente de Alzheimer perdido en el bosque.

En (Beckhusen, http://www.defensenews.com/, 2013) se puede encontrar una revisión histórica de la teoría de búsqueda nacida en la Segunda Guerra mundial con el objetivo de detectar submarinos alemanes causantes de grandes pérdidas en la contienda. En sus fases iniciales fue necesario refinar diferentes técnicas matemáticas, sobre todo probabilísticas, e identificar las dificultades de enfrentarse a un problema de optimización sometido a restricciones, todo lo cual se veía entorpecido por las limitaciones en la capacidad de computación disponible en las primeras décadas de la teoría. En dicho trabajo, también se revisan algunos de los hitos más relevantes que han impulsado el avance de la teoría hasta los métodos actuales, donde su ámbito de aplicación se ha extendido a diferentes tipos de problemas como son el tratamiento de grandes volúmenes de datos (Big Data), la búsqueda de terroristas insurgentes, o los restos de accidentes en el océano.

1.3 Una Aplicación real de la Teoría de Búsqueda

El 1 de Junio de 2009 el avión AF 447 de las línea Air France, con 228 ocupantes incluyendo pasajeros y tripulación, desapareció durante una tormenta sobre el Atlántico en un vuelo desde Río de Janeiro a Paris. El BEA (French Bureau d'Enquêtes et d'Analyses) organizó una búsqueda internacional por aire y con barcos de superficie que buscaban señales acústicas del avión caído y los posibles supervivientes. Sin embargo la búsqueda no tuvo éxito, por lo que se comisionó a un grupo de expertos oceanógrafos de Metron Inc. en Reston ((Lawerence D. Stone, 2011) para que estimaran las corrientes en el área en el momento del accidente, junto con información de las localizaciones donde habían aparecido restos y algunos cuerpos, o los mensajes recibidos desde el avión vía satélite antes del accidente.

Los analistas de Metron Inc. en Reston comenzaron su búsqueda incorporando todo lo que se conocía antes del accidente: dinámica de vuelo del avión, los vientos y las corrientes de la zona y otros accidentes de aviación con pérdida de control, asignando un 70% de probabilidad a estos datos por la credibilidad que se les concedía. Con las corrientes en el momento del estudio, estimaron las que existían al ocurrir el desastre, realizando una simulación hacia atrás para producir un número de trayectorias que terminaban en una localización estimada del accidente. De ellas se eliminaron las que se consideraban atípicas. También se incorporaron a estas probabilidades las posiciones y los tiempos de recuperación de cuerpos que se encuentran a la deriva en la superficie del océano, pero a estos datos se les asignó sólo un 30% de probabilidad debido a las turbulencias de las aguas ecuatoriales.

Estimaron un error normal bidimensional de cada estimación residual de la localización del accidente, obteniendo una media y una distribución normal del error. Con esta distribución se obtuvo un rectángulo centrado en la media con el 95% de probabilidad de contener la localización de los restos. Sin embargo, la búsqueda con vehículos submarinos en el rectángulo, extendida luego hacia el sur y el este del mismo, fue infructuosa. Se realizó una cuidadosa y metódica consideración de todos los datos disponibles con sus incertidumbres para formar áreas de alta probabilidad, produciendo finalmente una distribución bayesiana.

A partir de aquí se utilizó el sistema SAROPS (Kratzke T.M., 2010), utilizando la regla de Bayes para ir actualizando la información previa a la búsqueda con los datos que se iban obteniendo. De este modo se construyó una ruta de búsqueda que comenzaba en el punto de mayor probabilidad y exploraciones hacia las zonas de alta probabilidad, seguidas de las zonas de probabilidades intermedias, y por último por las áreas de baja probabilidad. Los restos se localizaron un año después a 14000 pies de profundidad. Evidentemente el objetivo de esta búsqueda no era encontrar las soluciones en tiempo mínimo, como se pretende este trabajo, sino la de generar caminos que maximizaban la probabilidad de detección del objetivo. Sin embargo, ambos métodos se fundamentan en construir mapas de probabilidades y, aplicando heurísticos sobre los mismos, obtener trayectorias de búsqueda para la detección de los objetivos.

1.4 Revisión Histórica

Como se señala en la revisión recogida (Lanillos Pradas, 2013), se pueden destacar tres períodos en la teoría de búsqueda. En el primero se empleaba conjuntamente la teoría probabilística bayesiana, la teoría algorítmica y una nueva disciplina desarrollada en la segunda guerra mundial conocida como investigación operacional, empleada en la toma de decisiones con base a métodos matemáticos. Estos modelos matemáticos y algoritmos se fueron desarrollando siendo Stone la figura más destacada del período inicial.

Posteriormente algunos autores se dieron cuenta de que la búsqueda era un subproblema de otro más general conocido como Procesos de Decisión Parcialmente Observables de Markov (PDPOM), ya que en realidad lo que se está realizando es una observación parcial de un estado desconocido. En ese período apareció el problema de la maldición de la dimensionalidad debida a la explosión del número de estados que aparecían en los problemas. A su vez, Eagle descubrió que los métodos de Stone no funcionaban si se introducían restricciones en los caminos y, por otro lado, la capacidad de computación no era suficientemente, por lo que se llegó a concluir que no se conocía ningún algoritmo capaz de resolver el problema de generar caminos óptimos de búsqueda.

A finales de la década de los 90 se diseñan algoritmos de ramificación y poda, y aparece una idea fundamental en la teoría: hasta que no se encuentre el objetivo todas las observaciones son de no detección. Por tanto, el problema es un problema de optimización determinista con una sola observación: no hay detección.

Al comienzo del siglo XXI se reactivó la investigación de búsqueda óptima desde la perspectiva de la fusión sensorial. Las líneas de investigación, que se centraron en aplicaciones reales y

configuraciones multiagente (lo que aumentaba la complejidad), se dividieron en tres perspectivas:

1. Se continuó con los algoritmos PDPOM, pese a que presentaban ciertas desventajas.
2. Apareció una aproximación discreta al problema de búsqueda junto con la apertura de la metodología a problemas más generales.
3. Se desarrolló una aproximación continua con fuerte base en la teoría de fusión sensorial.

El problema resultaba muy complejo y la respuesta debía obtenerse en un tiempo de computación razonable, de modo que debía restringirse utilizando aproximaciones y suposiciones. Entre ellas, la restricción más importante consiste en reducir el horizonte del agente o la ventana de decisión, con lo que su visión global del escenario desaparece. Este hecho hace que se corra el riesgo de tomar decisiones sesgadas con el peligro de caer en óptimos locales. Sin embargo, al ser la información incierta, su valor informativo puede perderse en el futuro. Otra alternativa consiste en dividir el sistema de búsqueda en un proceso de fusión sensorial y un controlador que guía los agentes. Para simplificar la toma de decisiones del controlador se asume que la información sobre la posición del objetivo entre los agentes está sincronizada.

En la mayoría de los casos, los investigadores han abandonado el objetivo original de minimizar el tiempo para encontrar el objetivo y se han centrado en maximizar la probabilidad de detectarlo. Sin embargo, en la Tesis de Pablo Lanillos (Lanillos Pradas, 2013) se propone una solución al problema de la búsqueda en tiempo mínimo (Minimum Time Search, MTS) como un proceso de decisión donde: 1) cada vez que se realiza una acción se produce una nueva observación que cambia la creencia sobre la posición del objetivo; 2) importa el orden en el que se ejecutan las acciones y se visitan las celdas; y 3) el proceso termina cuando el objetivo se detecta.

1.5 Planteamiento del Problema

Como se ha comentado, se va a utilizar la PG para obtener un programa controlador que, manteniendo las mismas premisas y función que determina el valor para la optimización empleados la Tesis de Pablo Lanillos (Lanillos Prada, 2013), genere la trayectoria de búsqueda en tiempo mínimo de un único agente para un mapa probabilístico dado. El mapa, desde la perspectiva del agente, es el mundo, y consiste en un enrejado de celdas a las que se ha asignado una probabilidad de localización del objetivo por celda a partir del conocimiento previo del mundo. La suma de todas las probabilidades será 1, es decir, la certeza de localizar el objetivo está distribuida entre todas las celdas del mapa.

La búsqueda será realizada por un cuatrirrotor, el agente, guiado en su movimiento por la secuencia de señales generadas por el programa controlador obtenido mediante PG. La secuencia de señales se proporciona como un vector de movimientos individuales codificados según el ángulo de desplazamiento entre las cuadrículas, en una de las 8 orientaciones cardinales (Norte, Nordeste, Este, Sudeste, Sur, Suroeste, Oeste, Noroeste).

El número de señales en la secuencia de control será N=10, ya que con este valor se consigue un avance significativo en la trayectoria del cuatrirrotor, tras el cual el agente puede volver a

tomar las probabilidades de su entorno, actualizadas con las medidas de su sensor, y calcular una nueva secuencia de señales de control.

Como se comentó, la búsqueda se va a restringir a objetivos estáticos, pues el mapa de probabilidades no se verá modificado por el movimiento del objetivo aunque sí por las observaciones realizadas por el agente. Se considera que el agente no detecta al objeto hasta que finaliza la búsqueda. El comportamiento del sensor es ideal, de manera que la probabilidad de una celda observada por el agente en la que no se detecta el objeto pasa a tener valor nulo.

1.6 Organización de la Memoria

El presente trabajo se estructura con el objetivo de encontrar una gramática en PG útil para obtener controladores que generen rutas óptimas en mapas de diferentes topologías. Se realiza una revisión teórica de la PG, centrándose en la PG basada en gramática, para posteriormente plantear los elementos de la gramática que puedan resolver el problema MTS. Se describen las pruebas realizadas con la gramática obtenida, una mejora de esta gramática, las pruebas realizadas para obtener 1) un controlador por mapa y 2) un controlador por tipo de mapa o genérico. Se valora la calidad de los controladores obtenidos (utilizando como referencia las soluciones obtenidas por CEO), se analizan los elementos de la gramática que aparecen en los programas resultantes y se estudian casos concretos sobre las trayectorias generadas para ciertos mapas, que muestran el comportamiento de PG y que dan una idea de cómo mejorar el método. Se muestran también los tiempos necesarios para realizar las optimizaciones, el tiempo de ejecución del programa controlador, y la evolución de las soluciones durante la optimización. Así, los capítulos que conforman esta memoria se han estructurado de la siguiente manera:

El capítulo 2 se centra en la exposición de la PG, repasando los conceptos fundamentales y la modalidad de PG que utiliza gramática. Se comentan los dos aspectos más importantes a los que se enfrentan los sistemas de PG, el de cierre y el de suficiencia. Por un lado, debe mantenerse la coherencia entre los parámetros y valores devueltos por las funciones que forman parte de la gramática y, por otro lado, que la gramática o genotipo debe contener, al menos, los elementos necesarios que permitan resolver el problema.

En el capítulo 3 se definen los elementos de la gramática propuestos para el problema MTS, siendo estos los tipos de movimientos que pueden realizar los agentes y las condiciones que determinarán cuál de esos movimientos aplicar en función de ciertos cálculos realizados sobre la probabilidad. Se plantea la gramática utilizada desde los nodos terminales hacia los nodos internos y los elementos que los agrupan, y se define la función de aptitud que valora la calidad de los programas o controladores obtenidos.

El capítulo 4 hace referencia a los resultados comparándolos con los obtenidos por el sistema CEO. Se muestran los resultados de obtener un programa controlador por mapa, mostrándose PG en este caso competente con CEO en cuanto a la calidad de las señales de control, pero no en cuanto al tiempo de optimización, y se comentan a continuación los resultados cuando el objetivo es obtener un controlador genérico por Tipo de Mapa o para todos los mapas, mostrándose el porcentaje de casos en los que PG daría resultados aceptables referidos a CEO.

Estos porcentajes muestran que la generalización en PG presenta ciertas deficiencias que son comentadas y de las que se proponen mejoras; aun así PG se muestra como un heurístico con soluciones aceptables al problema. Respecto a los tiempos de optimización, estos son notablemente superiores a los de CEO, no siendo sin embargo este el objetivo del método, sino el de obtener un controlador genérico cuya ejecución pueda realizarse de un modo casi instantáneo al aplicarse sobre un mapa. Se analizan los movimientos que aparecen en los controladores genéricos respecto al controlador individual por mapa, comprobándose que desaparecen algunos tipos de movimientos que podrían ser resolutivos en los mapas individuales, y se muestran las trayectorias obtenidas en algunos mapas que reflejan los comportamientos del controlador generado por PG.

Finalmente, en el capítulo 5, se recogen las conclusiones más importantes y se plantean algunas líneas futuras de investigación.

CAPÍTULO 2. Programación Genética

2.1 Introducción

La programación genética (Riccardo Poli W. B., 2008) es una metodología basada en algoritmos evolutivos con el fin de desarrollar automáticamente programas de computadoras que realicen una tarea definida por el usuario. Es una especialización de los algoritmos genéticos donde cada individuo de la población es un programa de computadora y una técnica de aprendizaje automático utilizada para optimizar un conjunto de programas de acuerdo a una función de ajuste, o aptitud, que evalúa la capacidad de cada individuo (programa) para llevar a cabo una tarea determinada.

En (Weise, 2011) se especifican dos aspectos importantes de lo que es la programación genética:

1) Algoritmos que evolucionan basados en población, teniendo como genotipos estructuras de datos construidas en forma árbol. Las estructuras en árbol son naturales en programación puesto que muchos compiladores las construyen al realizar el análisis de los programas. Ahora bien, estas estructuras en árbol son manejadas internamente por el sistema que controla las operaciones evolutivas, ya que desde la perspectiva del usuario no es necesario que este conozca o especifique la estructura o la forma de la solución, siendo una técnica sistemática e independiente del dominio (Riccardo Poli W. B., 2008). Pese a este desconocimiento, en (Koza, 1998) se indica que los programas se encuentran entre las estructuras más complejas creadas por el hombre, lo da idea de su potencial para resolver diferentes problemas.

2) Engloba todos los algoritmos posibles que engendran programas. Si bien la Inteligencia Artificial contiene métodos de aprendizaje automático como son las redes neuronales, las gramáticas formales, los árboles de decisión, las máquinas de soporte vectorial, etcétera, estos métodos involucran estructuras especializadas para facilitar el hallazgo de la solución e incluso el posterior análisis matemático. Sin embargo, estas estructuras no son naturales y deben programarse explícitamente, lo que restringe la forma de resolver los problemas. Si un humano programase con ese tipo de estructuras no lo haría de una forma natural ni dispondría de la flexibilidad necesaria. Además, en los lenguajes de programación se pueden realizar operaciones de forma jerárquica, emplear iteraciones y recurrencias, realizar cálculos sobre variables de diferentes tipos, o crear variables intermedias y subprogramas (Koza, 1998), y todos estos aspectos aumentan tanto su potencial como su "compresibilidad" para los humanos.

Las características más importantes de la PG son:

- Material Genético no lineal y generalmente estructurado en árbol (históricamente se han utilizado otro tipo de representaciones que en el fondo son trasladables a árboles utilizando "mapeos" del tipo genotipo-fenotipo, como es el caso de la representación en LISP en forma de una cadena de elementos equivalentes a un árbol)
- Material Genético de longitud variable, si bien se imponen ciertas limitaciones al tamaño del árbol para evitar el problema del crecimiento incontrolado, ya que a

medida que avanzan las generaciones el tamaño de los individuos crece sin que necesariamente mejore la aptitud. De hecho, habitualmente comienza a haber programas más grandes con la misma aptitud que otros de menor tamaño. Además, se trata de un crecimiento rápido, prácticamente exponencial. Este problema se debe al operador de cruce el cual permite que los hijos tengan más profundidad que los padres. A pesar de esta problemática, el que la estructura arbórea pueda crecer de manera considerable posibilita una búsqueda más amplia en el espacio de soluciones.

- Material Genético Ejecutable. Las estructuras son interpretadas, bien sea por un lenguaje de computación existente o por un lenguaje diseñado específicamente para el problema.
- Cruce Genético que preserva la sintaxis. El cruce es la operación genética más importante en PG, a la vez que la más arriesgada puesto que un nodo del árbol de una estructura, a cualquier nivel, es sustituido por otro nodo de otra estructura y no necesariamente en el mismo nivel. Realizada esta operación el programa resultante debe ser correcto. Si hay implicadas funciones con un determinado número de argumentos, o de aridad, y esos argumentos proceden de cruces con otros nodos, la consistencia entre parámetros de funciones y valores devueltos por las mismas debe preservarse, es decir, el número de parámetros que reciben debe ser adecuado y los valores devueltos deben poderse utilizar como parámetros. Como se comentará esta propiedad se llama “Cierre” y existen técnicas para preservarla.

2.2 Elementos para Crear un Programa con PG

¿Cómo se organiza la información para que un ordenador programe sin decirle exactamente qué hacer? En PG la creación de programas es aleatoria y se llega a la solución partiendo de un estado inicial y aplicando mecanismos de selección a las estructuras intermedias que se van encontrando. Según (Koza, 1998), el programa emerge como consecuencia de la aptitud, ya que esta es la responsable de generar la estructura del programa necesaria para resolver el problema en cuestión. Además, son muchos los tipos de problemas (control, planificación, estrategias de juego, etcétera) que pueden reformularse para ser resueltos en PG. Para que esta creación de programas ocurra deben definirse cinco puntos importantes:

1. Conjunto de Terminales, o las hojas de los árboles. Típicamente son variables atómicas (entradas, sensores, detectores, variables de estado) o constantes atómicas (un número, un valor booleano, null). Ocasionalmente pueden ser funciones sin argumentos que producen efectos sobre el estado del sistema.
2. Funciones que reciben un número de parámetros y devuelven un tipo de datos (entradas y salidas compuestas por otras Funciones o por elementos Terminales)

   El conjunto de Terminales y Funciones deben atenerse al principio de **Cierre**: cada función debe aceptar como argumento cualquier valor y tipo de datos, posiblemente devuelto por alguna otra función, así como utilizar como elementos terminales los retornos de otras funciones. Existen técnicas para mantener esta propiedad de cierre, que se aplicarán en la operación de cruce de individuos, como por ejemplo incluir en las funciones código de salvaguarda para los casos en los que falte algún parámetro.

3. La forma de medir el comportamiento del programa se lleva a cabo por la medida de aptitud y el tipo de medida utilizado depende de la naturaleza del problema (ponderando un error, minimizando el tiempo en control óptimo, por una combinación de aciertos positivos y negativos y falsos positivos en reconocimiento de patrones, etcétera) En el presente problema será un parámetro relacionado con la generación de rutas sub-óptimas en tiempo mínimo.
4. La manera de efectuar operaciones genéticas sobre la población actual se basa en la aptitud y en parámetros que controlan la ejecución, siendo el más importante el tamaño de la población, así como las probabilidades asignadas a las operaciones genéticas (cruce, mutación), la estrategia de selección del individuo que pasa a la siguiente generación, y el tamaño máximo de los programas obtenidos o individuos.
5. El criterio de finalización y la selección del programa resultado. Habitualmente, la ejecución finaliza tras un número máximo de generaciones y el programa elegido es el mejor individuo encontrado a lo largo de todas las iteraciones.

Finalmente, es importante destacar que el conjunto de todas las funciones y terminales forman lo que se denomina primitivas del sistema PG. Estas primitivas tienen que ser suficientes, es decir, el problema puede expresarse mediante su combinación (el conjunto de todas las posibles composiciones recursivas de las primitivas debe incluir al menos una solución). Desafortunadamente la suficiencia no puede garantizarse en algunos problemas, y hay que recurrir a alguna teoría o a experiencias realizadas con otros métodos para entender qué tipo de primitivas podrían utilizarse.

2.3 Representación, Inicialización y Operaciones en PG

A continuación se describen las características de los elementos más relevantes de la PG.

2.3.1 Representación

Como se ha comentado, para la representación de los individuos o programas no se utilizan líneas de código sino árboles sintácticos. Por ejemplo, si un programa tuviera como código la línea “max(x*x,x+3y)”, (Riccardo Poli W. B., Genetic programming an introductory tutorial and a survey of techniques and applications, 2007), los elementos terminales serían las variables x e y, junto con la constante 3, y los nodos internos o funciones, son los operadores +,* y max. Su representación como Árbol Sintáctico se muestra en la Ilustración 1.

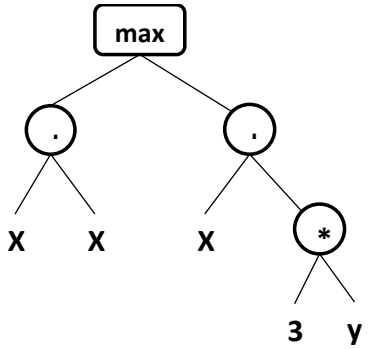


Ilustración 1. Representación de max(x*x,x+3y) en un Árbol Sintáctico

Pueden crearse estructuras más complejas agrupando un conjunto de nodos que posteriormente será tratado como un componente más en un solo nodo. Diseños más complejos permiten pasar como parámetros funciones completas, como son las expresiones lambda. Sin embargo en el presente trabajo no se realizan este tipo de ampliaciones.

El proceso evolutivo debe inicializarse con una población de partida. Esta inicialización es aleatoria, sin embargo, existen tres modos básicos de realizarla:

- **Completa**. Genera árboles de un tamaño y forma predeterminado. Se escogen nodos de tipo función hasta que se ha alcanzado la máxima profundidad, y más allá de dicha profundidad solo se escogen nodos terminales.
- **Creciente**. Permite crear árboles de forma y tamaño variados. Se cogen nodos funciones y terminales hasta alcanzar la profundidad máxima, momento a partir del cual solo se permiten nodos terminales.
- **Mixta con rangos.** La mitad de la población inicial es construida de forma completa y la otra mitad con el método creciente. Además, utiliza un rango para los límites de profundidad, asegurando una variedad de formas y tamaños.

En los dos primeros métodos el tamaño y la forma son muy dependientes de la profundidad escogida. El método creciente genera árboles muy cortos y en el caso de que haya muchas más funciones que terminales es similar al método completo. Además, cambios en el número de funciones o terminales afectan a las formas de los árboles. El tercer método permite generar un amplio rango de programas con diferentes características. En todo caso se debe procurar preservar la diversidad en las siguientes generaciones, consiguiendo que la ramificación y profundidad del árbol inicial tenga la suficiente variedad.

2.3.2 Operaciones en PG

Las operaciones utilizadas para construir nuevos programas en base a los programas ya existentes se presentan a continuación.

2.3.2.1 Selección

Determinados individuos o programas se seleccionan para la siguiente generación en base a su aptitud (función objetivo o criterio de optimización), de modo que los más aptos tengan mayor probabilidad de ser seleccionados. El método más habitual es la selección por torneo, así como la selección proporcional a la aptitud.

2.3.2.2 Cruce

La forma más habitual de realizar el cruce es mediante la combinación de subárbol. Dados dos padres, se selecciona aleatoriamente un nodo como punto de cruce en cada uno de los árboles. La selección de los nodos es aleatoria y no tienen por qué encontrarse al mismo nivel del árbol. El descendiente se crea reemplazando el sub-árbol encabezado en el punto de cruce de una copia del primer padre con una copia del sub-árbol encabezado en el punto de cruce en el segundo padre:

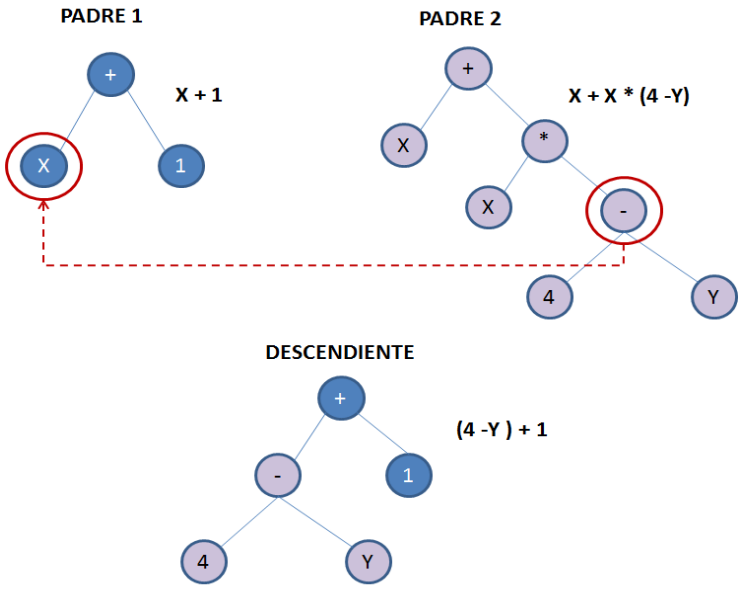


Ilustración 2. Cruce de Subárbol.

Existen muchos otros tipos de cruces, como el cruce de punto que se realiza seleccionando un punto de cruce común que tengan los padres e intercambiando los correspondientes sub-árboles. Con el fin de mantener la diversidad estructural puede realizarse un análisis de los árboles para seleccionar solo aquellas regiones comunes con las que realizar el cruce. En el cruce con preservación del contexto los puntos de cruce se restringen a los que tengan las mismas coordenadas. En el cruce de tamaño imparcial, el primer punto de cruce se selecciona aleatoriamente, como en un cruce estándar, para calcular luego el tamaño del subárbol del primer padre a ser escindido de tal modo que la selección del segundo punto de corte no seccione un subárbol demasiado grande.

2.3.2.3 Mutación

Para el caso de la mutación, la forma más habitual es la selección aleatoria de un punto de mutación y la sustitución del sub-árbol por otro generado aleatoriamente. También puede realizarse como el cruce de un programa con otro generado aleatoriamente. En la mutación de punto, se selecciona aleatoriamente un nodo y la primitiva o función almacenada allí se reemplaza por otra generada aleatoriamente con la misma aridad. Si no hay otras primitivas de esa aridad, no ocurre la mutación.

2.4 PG basada en Gramáticas

En la Programación Genética basada en gramática (PGBG), el genotipo y el fenotipo son entidades separadas: el genotipo se ha de trasladar a un programa y la gramática asegura que el resultado de la traslación produzca un programa sintácticamente correcto. La PGBG no impone restricciones sobre la operación genética de cruce, siendo ahora esta operación similar a la realizada en los Algoritmos Genéticos, es decir, el cruce puede ocurrir en cualquier lugar del genotipo. La gramática también permite expresar restricciones, cumpliéndose la propiedad de cierre de una forma natural, con lo que queda asegurado que las funciones reciban los argumentos correctos y sus retornos puedan pasarse a otras funciones o ser utilizados como elementos terminales.

Una gramática está constituida por elementos no terminales, elementos terminales y reglas de producción. Los elementos terminales y no-terminales son del mismo tipo que las funciones en PG. La diferencia entre PG y PGBG aparece con las reglas de producción, puesto que estas reglas se encargan del cumplimiento de la sintaxis.

Para ilustrar el funcionamiento de las reglas presentamos el siguiente ejemplo, en el que puede verse como cada una de las líneas de la gramática se corresponde con una regla de producción. Cada regla está formada por una parte izquierda que identifica el elemento de la gramática al que afecta, una flecha que indica que se trata de una producción, y una parte derecha con todos los elementos terminales y no terminales de la gramática que pueden elegirse en la regla y que se encuentran separados por una barra vertical (símbolo de disyunción).

S → B

B → and BB | or BB | not B | if B B B | T

T → a0 | a1 | d0 | d1 | d2 | d3

En la primera regla, S será transformado en B; en la segunda, B puede ser transformado aleatoriamente en cualquiera de las 5 opciones separadas por una barra; y en la tercera, T será transformado en una de las 6 opciones separadas por la disyunción.

2.4.1 Gramáticas Libres de Contexto (GLC)

En estas gramáticas la parte izquierda de las producciones sólo puede tener un símbolo no terminal. Una gramática libre de contexto está constituida por la tupla (**N**,**∑**,**P**,**S**), donde **N** es el alfabeto no terminal(también denominado variable), **∑** es el alfabeto terminal (los símbolos del lenguaje que se esté definiendo), **P** es el conjunto de producciones, y **S** es el símbolo designado como inicio (Whigham, 1995).

Las producciones, se separan por comas, y pueden ser como alguna de las siguientes: **x → y**, **x → z, x → y | z** donde **x Є N**; **y, z Є {∑ U N}**, (U es el símbolo de unión). La primera regla indica que dado x se produce y, la segunda que dado x se produce z, y la tercera regla aglutina ambas producciones mediante el símbolo de disyunción **|**. Por tanto, se responde a la siguiente forma **"variable → cadena de variables y terminales".**

Las ideas básicas tras las GLC son:

1) Utilizar variables para generar conjuntos de cadenas.

2) Definir estas variables recursivamente en términos de otras variables.

3) Definir reglas recursivas (o producciones) que suponen una concatenación de las cadenas de variables.

Por lo tanto, la cadena se va produciendo con varias aplicaciones de las reglas y, cada vez que se aplica el símbolo "→" se produce un "**paso de derivación**", una transformación de la cadena, que se va ampliando.

El término libre de contexto se refiere al hecho de que el elemento no terminal, x en los ejemplos, puede ser sustituido por y o z sin tener en cuenta el contexto en el que pueda ocurrir dicha sustitución.

2.4.2 Gramáticas Libres de Contexto y PG

De (Whigham, 1995) se pueden sacar las siguientes conclusiones respecto de la adaptación de las GLC a la PG:

- Para generar los individuos de la población inicial se utilizan las reglas de manera aleatoria.
- Resulta más complicado generar individuos de una longitud prefijada puesto que es más difícil utilizar el método mixto por rangos, al tener que considerar la profundidad del árbol.
- En la operación de cruce, para generar individuos correctos se eligen dos puntos de cruce que puedan ser generados por la misma regla.
- Es necesario especificar cuántas veces se puede utilizar una regla para limitar el tamaño final del árbol.

2.4.3 Notación de las GLC en la Forma de Backus Nahur (BNF)

Una notación habitual para especificar las gramáticas en PG, y que es la utilizada en este trabajo, es la Forma de BNF, un metalenguaje usado para expresar gramáticas libres de contexto. Una especificación en BNF consiste en un sistema de reglas de derivación escrito como un conjunto de expresiones del tipo:

<simbolo> ::= <expresión con símbolos>

donde "símbolo" es un elemento no terminal y "expresión con símbolos" una secuencia de símbolos u otras secuencias (de símbolos), cada una separada por la barra vertical (operador de disyunción '|'), constituyendo así el conjunto de posibles sustituciones posibles a asignar al símbolo a la izquierda. Los símbolos que nunca aparecen en el lado izquierdo de las expresiones son los elementos terminales.

Un ejemplo de cómo se expresa la gramática mediante BNF, que es la gramática utilizada en este trabajo, puede encontrarse en el apartado 3.1.4.

2.5 Sistema Jeco: Herramienta Java de PG

Jeco, “a Java Evolutionary COmputation library” (Adaptive And Bioinspired Systems Research Group), es una librería de Computación Evolutiva construida en Java y desarrollada por investigadores del Departamento de Arquitectura de Computadores y Automática de la Universidad Complutense de Madrid.

Como herramienta de optimización de programa, es necesario proporcionar a Jeco la gramática (definida en la forma BNF) propia del problema que se desea resolver.

Desde el punto de vista de la programación en Jeco, los elementos que se le deben facilitar son:

1. El fichero con la gramática.
2. La función objetivo utilizada para determinar qué programas son considerados más aptos.
3. El tamaño de la población para la evolución genética.
4. El número de generaciones en las que se aplicará la evolución. Una vez alcanzado este número, Jeco terminará la optimización y devolverá el conjunto de mejores soluciones encontradas.

La herramienta suministrará en cada generación un programa a partir de la gramática en forma de cadena de caracteres. Este programa se prueba para valorar su aptitud, que será devuelta a Jeco para que valore las mejores soluciones. Adicionalmente se le puede indicar a Jeco el número de errores si en el problema que se está optimizando estos pueden cuantificarse.

Estos son los aspectos con los que se ha configurado Jeco dejando sus valores por defecto para el proceso evolutivo: probabilidades para seleccionar individuos a la siguiente generación, probabilidad de mutación, tipo de inicialización, tipo de operación de cruce, etcétera. Por el alcance del presente trabajo no se ha entrado en modificar estos valores puesto que afectan a la forma de construir las nuevas generaciones de individuos y, en principio, los programas obtenidos por la configuración presente en Jeco ha servido para valorar los elementos diseñados en la gramática y sus comportamientos.

Es importante destacar que actualmente existe una versión más actualizada de Jeco con mejor rendimiento en cuanto a tiempos de ejecución, pero que no ha sido utilizada en este Trabajo Fin de Master al presentar una interfaz de funciones diferentes a la versión utilizada.

CAPÍTULO 3. Representación del Problema de Búsqueda en Tiempo Mínimo Mediante PG

En este capítulo se describe la gramática elegida para obtener un programa controlador que minimice el tiempo de búsqueda de un objetivo por un agente. Es decir, se presentan los elementos terminales y no terminales de la gramática, así como las estructuras que se incluirán en el programa controlador (bucles, reglas "IF-THEN-ELSE", funciones y subrutinas) que han sido definidos teniendo en cuenta las características del problema. También se especifica la función objetivo elegida para medir la aptitud de los individuos o programas controladores generados en cada generación.

3.1 Definición de la Gramática

A la hora de elegir las funciones del problema de búsqueda en tiempo mínimo se tiene en cuenta que, según (Koza, 1998), las estructuras que aparecerán en PG son todas las posibles funciones que resultan de combinar recursivamente el conjunto de funciones $\mathbf{F = \{f_1,f_2,...,f_{Nfunc}\}}$ y el conjunto de terminales de $\mathbf{T = \{a_1,a_2,...,a_{Nterm}\}}$. Además, cada función $\mathbf{f_i}$ tendrá una aridad, (o conjunto de argumentos de la función en un número, orden y tipo determinados) prefijada y será definida bien como una:

- Operación aritméticas (+, -, *, etc).
- Función matemática (sin, cos, exp, log, etc.).
- Operador booleano (AND, OR, NOT, etc).
- Operador condicional (como IF-THEN-ELSE).
- Función que cause iteración (como while, for).
- Función que cause recursión.
- Función propia del problema que se desea resolver. Éstas últimas son las más importantes, ya que nos permitirán definir los comportamientos básicos del agente, que la herramienta combinará para obtener el programa controlador buscado.

Por otro lado, el conjunto de elementos terminales y funciones primitivas debe ser suficiente para expresar la solución del problema. Conseguir esta identificación de variables no siempre es evidente y puede requerir cierta perspicacia o conocimiento del problema. Además, la elección del conjunto de funciones y terminales afecta directamente al carácter y a la forma que tomarán las soluciones. Para la mayoría de problemas, el conjunto de funciones no incluirá únicamente el mínimo suficiente que se emplearía al resolver el problema a mano, pues ante un proceso de aprendizaje pueden incluirse funciones "extrañas/ajenas" al mismo. El efecto en el funcionamiento de estas funciones es complejo, pero por norma general, el tener numerosas funciones extrañas/ajenas degrada el rendimiento de la PG aunque el problema siga resolviéndose. De igual modo, la existencia de alguna función que en principio pueda parecer extraña puede aportar un comportamiento adicional que mejore considerablemente el rendimiento de la PG. Es decir, incluir funciones "extrañas" adicionales no impide que se obtenga una solución, aunque es conveniente aproximarse al conjunto mínimo de funciones adicionales suficientes que no provoquen opacidad en la solución o degraden su rendimiento.

3.1.1 Hipótesis Sobre la Estructura de los Programas Generados por PG

Recordemos que el agente habita en un entorno de probabilidad sobre el que se podrán calcular ciertas condiciones, sean estas valores de medias, varianzas, gradientes, u otros cálculos, y estará situado en una determinada celda desde la que podrá realizar algún tipo de movimiento.

Se plantea la hipótesis de que el movimiento completo del agente será el resultado de la composición de movimientos simples que se irán concatenando a partir de las condiciones de probabilidad que encuentre el agente al alcanzar cada celda. Por una parte, estos movimientos simples, realizables desde cualquier celda con un determinado número de casillas (definidos en el apartado 3.1.2), pueden consistir, por ejemplo, en avanzar en dirección axial (en una de las ocho direcciones cardinales) un pequeño número de celdas, realizar una L (con un cierto tamaño en celdas de su base y altura), ir hacia la celda de más probabilidad de una determinada fila o columna, etcétera. La concatenación de estos movimientos simples construirá la ruta completa. Por otra, las condiciones de probabilidad que se comprueban tras alcanzar una celda pueden dar como resultado continuar con el movimiento simple anterior o comenzar uno diferente. Los movimientos simples serán los nodos terminales de la gramática, las condiciones de probabilidad serán las funciones no terminales.

Al desconocer a priori que tipo de nodos terminales y funciones de condición van a resultar más útiles, se ha diseñado una gramática en la que ambas pueden tener dos naturalezas diferentes:

a) Local o cercana, en la que solo se consideran las celdas más cercanas a aquellas donde se encuentra el agente.

b) Más genérica, en la que se considera el conjunto total de celdas del enrejado de 20 x 20, y no sólo las celdas cercanas, con el fin de evitar centrarse en probabilidades cercanas. Esto intenta evitar la posible miopía considerando toda la probabilidad del enrejado de celdas.

Existen movimientos en los que se debe especificar el tamaño en número de celdas del movimiento, por ejemplo, en los de tipo L la base o altura de la misma. PG deberá resolver el valor correcto del parámetro en cada caso y en la gramática se ha planteado como la elección aleatoria en un rango de valores posibles. Esto amplía el espacio de búsqueda de soluciones, lo que ralentiza el proceso de optimización.

En principio se desconoce qué tipo de movimiento es el adecuado para cada mapa, incluso qué condiciones se deben definir para resolver el problema. Durante las pruebas ciertos tipos de movimientos y sobre todo de condiciones se han descartado porque no aportan conocimiento problema (los valores que devuelven las condiciones no varían prácticamente al ser evaluadas y no son útiles para discriminar).

3.1.2 Tipos de Movimientos Simples

La trayectoria total del agente tendrá un total de 10 movimientos de casilla y estará constituida por movimientos simples, que serán de un tamaño menor de 10, n. Por otra parte, muchos de los movimientos simples pueden realizarse en una de las 8 orientaciones

cardinales. Para determinar cuál es la mejor, se prueba el movimiento simple de tamaño n en cada una de esas orientaciones y se escoge la orientación que obtenga mejor LET.

Sin embargo puede haber empates en algunos LETs de las orientaciones y, en ese caso, se elige la orientación que continúe con el movimiento anterior, es decir, la orientación que tenga alguna componente igual al movimiento anterior. Por ejemplo, si el movimiento anterior era norte, otro movimiento norte continuaría el movimiento, pero si el movimiento norte no tiene el mejor LET, se escogería entre los movimientos nordeste y noroeste. Si estos tampoco tuvieran mejor LET, se preferirían los movimientos este, oeste a los que tengan alguna componente sur. Esto puede producir miopía pues, en función del tamaño n del movimiento, se están eligiendo mejores opciones locales.

Si la casilla donde se encuentra el agente se encuentra cercana a los bordes del mapa de probabilidades de 20x20 celdas, quizás no puedan considerarse los n movimientos en todas las direcciones. Esto implica que las comparaciones por LET no tendrían el mismo número de casillas n, y no serían equivalentes, con lo que:

- Si n=0 en una dirección no se considera la dirección para mover el agente.
- Se reduce la comparación al tamaño n del movimiento en la dirección que tenga el mínimo número de casillas. Es decir, el n para comparar el movimiento en las 8 direcciones se reduce al de la dirección que tenga menor valor por topar con un borde.

No se ha planteado normalizar el LET dividiendo por el número de movimientos para comparar los LETs en las diferentes direcciones, puesto que si hay mucha diferencia en el número de celdas de cada movimiento, no serían comparables.

3.1.2.1 Movimiento Axial Local (MoveLocalAxial)

Este movimiento parte de la casilla actual y se mueve consecutivamente N-pasos en alguna de las 8 direcciones correspondientes a los puntos cardinales: norte, sur, este, oeste, noreste, noroeste, sureste, suroeste.

En la Ilustración 3, como ejemplo, se muestran estas posibilidades para un movimiento de tamaño n=3. Como se observa, desde la casilla donde se encuentra el agente se valorarán 8 rutas de tres casillas en cada una de las 8 direcciones cardinales, calculando el LET de cada una.

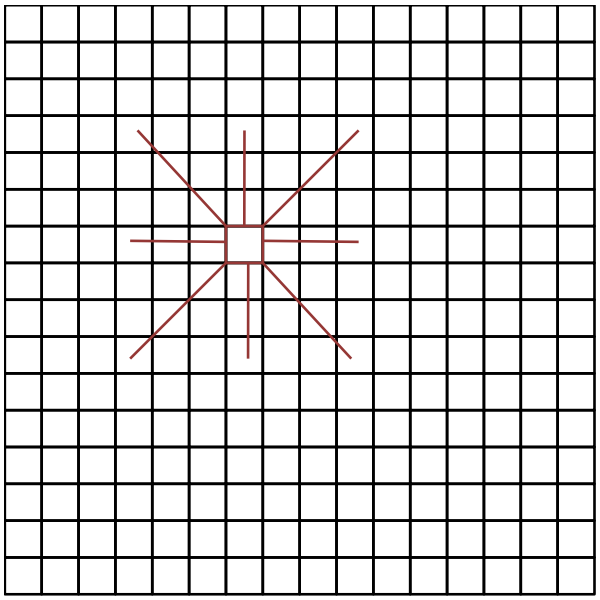


Ilustración 3 Movimiento Axial Local

Se elegirá la dirección con menor LET (resolviendo los empates como se comentó anteriormente). A la ruta total que se está componiendo se le añadirían las tres casillas de la orientación seleccionada.

En la gramática el tamaño n del movimiento está parametrizado en un rango de 1 a 10. El proceso de optimización elegirá en cada aparición de este movimiento que número de celdas asignarle al movimiento.

3.1.2.2 Movimiento Axial Global (MoveAxial)

Este movimiento es idéntico al anterior, solo que el tamaño del movimiento es el máximo posible desde la casilla hasta los bordes (en realidad n sería la distancia al borde más cercano). Si la celda donde está el agente se encuentra cerca de los bordes es similar a MoveLocaLAxial, puesto que n tomará un valor pequeño. Si no está cerca de los bordes, tendrá en cuenta muchas casillas, respondiendo así a situaciones en las que el agente se encuentre en zonas de poca probabilidad con la necesidad de encontrar mejores regiones.

3.1.2.3 Movimiento por Cuadrantes Local (MoveByLocaLQ)

A partir de la casilla actual se construyen cuatro cuadrados de un tamaño determinado (en la Ilustración 4 se muestran los cuadrados con tamaño 3). Si un cuadrado tiene una media de probabilidad mayor que los otros, se escoge en dicho recuadro la casilla con mayor probabilidad. Si existe más de una casilla con igual probabilidad a la máxima, se escoge la de mínima distancia (se entiende como distancia entre dos casillas el número de casillas que las separan) y, si hay varias, se escoge aquella que, al dirigirse hacia ella, se obtenga como primer movimiento el más parecido al movimiento anterior.

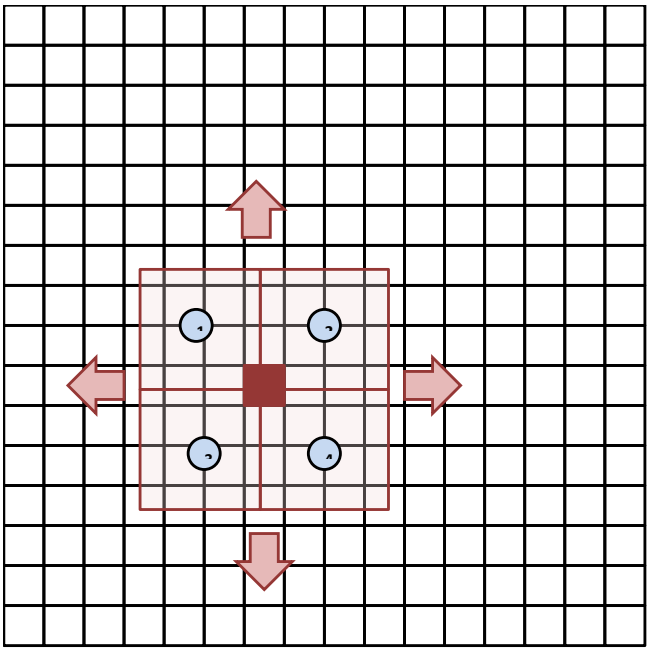


Ilustración 4 Movimiento por Cuadrantes Local

El problema principal de este procedimiento es que pueden aparecer bastantes empates, caso en el que se procede del siguiente modo:

1. Establecer el tamaño de los cuadrados a un tamaño mínimo (comenzando por 2, pues se incluye la celda actual como un vértice de los cuadrados a considerar). Se ha comprobado que si el tamaño de los cuadrados es muy pequeño el método resulta "demasiado local" y el comportamiento del agente se vuelve muy miope.
2. Calcular, si existe, un cuadrado dominante de los cuatro en media de probabilidad. De existir, componer la ruta a la casilla de máxima probabilidad de dicho cuadrante.
3. En caso de que haya más de un cuadrado dominante, aumentar en uno el tamaño de los cuadrados y repetir consecutivamente los pasos 1 y 2 hasta encontrar uno que domine (al incluir más casillas, y por tanto más valores para obtener la media de probabilidad, es esperable que aumente la diferencia en las medias de probabilidad de los cuadrados)
4. Si el tamaño de los cuadrados alcanza el máximo de casillas horizontales o verticales y no se ha encontrado un cuadrado dominante, se vuelve a establecer el tamaño al inicial, se calculan los cuadrantes dominantes, sus casillas de máxima probabilidad, y se escoge aleatoriamente entre las casillas dominantes.

3.1.2.4 Movimiento por Cuadrantes Global (MoveByGlobalQ)

Desde la casilla actual se divide toda la cuadrícula 20 x20 en cuatro cuadrados hasta las casillas extremas, y se calcula la media de probabilidad de cada uno de esos cuadrados. A partir de ahí, el proceso de selección es idéntico al del caso anterior, es decir, se escoge como destino la casilla de mayor probabilidad y se compone una ruta de mínimo LET hacia ella.

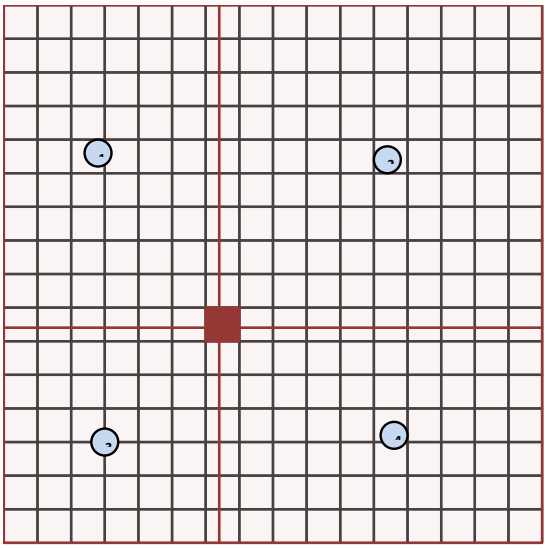


Ilustración 5 Movimientos por cuadrantes global

3.1.2.5 Movimiento por Filas y Columnas (MoveByRowCol)

Es similar al movimiento por cuadrante local (MoveByLocalQ), en el que se rodeaba a la casilla actual con cuadrados, de los que se calculaba la probabilidad media, escogiendo la casilla de máxima probabilidad del cuadrado con mayor media. En este caso, en lugar de rodear a la casilla actual con cuadrados, se la rodea con un perímetro rectangular de casillas. Entonces se valora la probabilidad media de cada una de las aristas del perímetro cuadrado, llamadas filas si son horizontales y columnas si son verticales, y se escoge la casilla de máxima probabilidad de la arista de mayor probabilidad media, tratando los empates y componiendo un camino de mínimo LET hacia ella.

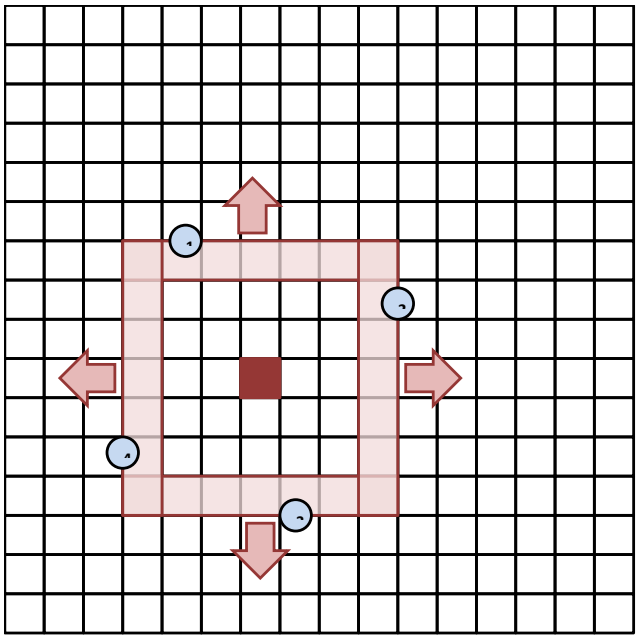


Ilustración 6. Movimiento por Filas y Columnas

Como se comentó con MoveByLocalQ, se parte de un tamaño inicial, en este caso la distancia a la que estará el perímetro (en la figura aparece a una distancia de 3 casillas) y, si no se encuentra una arista del perímetro dominante, se aumenta la distancia del perímetro para incluir más casillas con las que calcular la probabilidad media. Los cálculos se realizan del mismo modo.

3.1.2.6 Movimiento en L a Derechas o Izquierdas (LRight y LLeft)

Este tipo de movimiento intenta componer una trayectoria en forma de L desde la casilla actual. La L se considera formada por una base, una altura y un sentido de la L, hacia la izquierda o hacia la derecha (de ahí que se haya dividido en dos tipos LRight y LLeft), así como la orientación hacia una de las 8 orientaciones cardinales. Al calcular las trayectorias para las 8 orientaciones cardinales de la L, el conjunto de casillas que en conjunto evaluará el método se aproximará a un círculo en torno a la casilla actual (ver Ilustración 8). En la Ilustración 7 se muestran las dimensiones de la L.

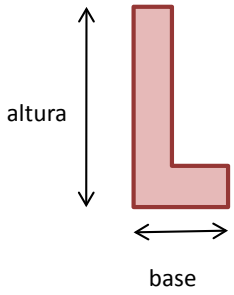


Ilustración 7 Dimensiones de la L

Los tamaños en número de casillas que puedan darse a la altura y la base pueden variar y están parametrizados en la gramática, ambos en un rango de 1 a 5. Esto aumenta el espacio de búsqueda de las soluciones, sin embargo es necesario, porque el comportamiento de este movimiento está muy influido por estas dimensiones y en muchos mapas los programas generados presentan diferentes combinaciones de las mismas, siendo un método que, en la optimización por mapa puede ser resolutivo para mejorar la solución. En la Ilustración 8 se muestra, para una L a derechas, los movimientos en las ocho orientaciones.

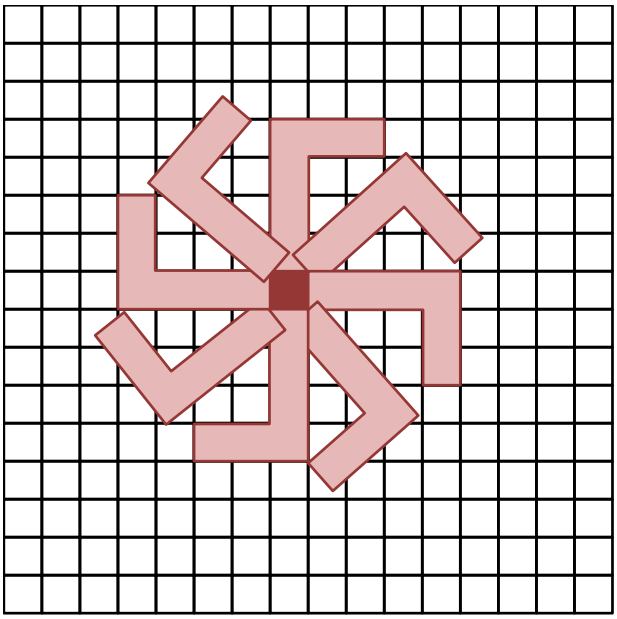


Ilustración 8 Movimiento L Derecha en las 8 Direcciones Cardinales

Este movimiento puede formarse a partir de otros movimientos más básicos, pero se ha observado que introducir un movimiento como este mejora la convergencia de la solución del PG.

3.1.2.7 Movimientos Moverse a la Celda con Mayor Probabilidad (MoveMostImportantCell) y Moverse al Entorno de Mayor Probabilidad (MostImportantRoundCell)

Selecciona la casilla de máxima probabilidad y compone un movimiento de mínimo LET hacia ella. Si hay varias casillas empatadas en probabilidad máxima se escoge la más cercana y si persiste el empate, se opera como en cualquier empate. En la Ilustración 9 se observa una ruta desde la celda del agente hacia la celda de máxima probabilidad. Dado que el método MoveLocalAxial[1] (entre corchetes el tamaño del movimiento) es bueno al moverse por la máxima probabilidad que rodea la celda, es el que se utiliza para obtener una trayectoria entre las dos celdas (si se construyera una ruta por la mínima distancia entre ambas celdas, no se aseguraría que se pasase por zonas de buena probabilidad).

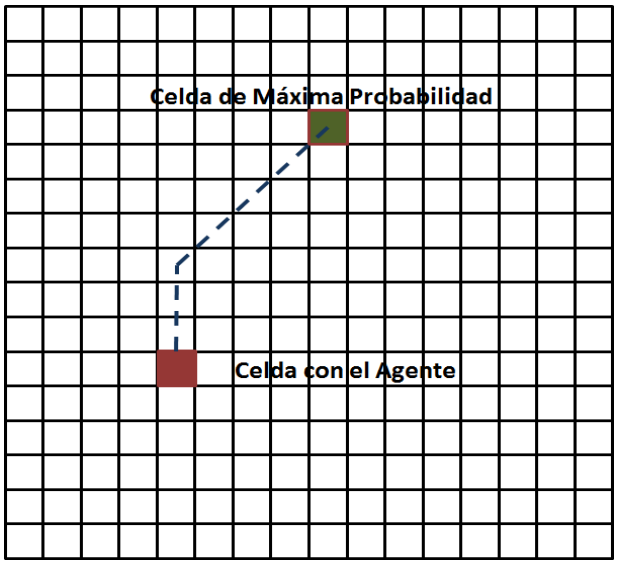


Ilustración 9. Moverse a la Celda de Máxima Probabilidad (MoveMostImportantCell)

Como basar el método en una única casilla puede no ser representativo de una región de probabilidad, el método MoveMotImportantRoundCell considera un cuadrado de un determinado tamaño en torno a la celda de máxima probabilidad, y las operaciones para detectar la celda destino se realizan considerando la media de probabilidad de las esas casillas, en lugar del valor de probabilidad de una única celda.

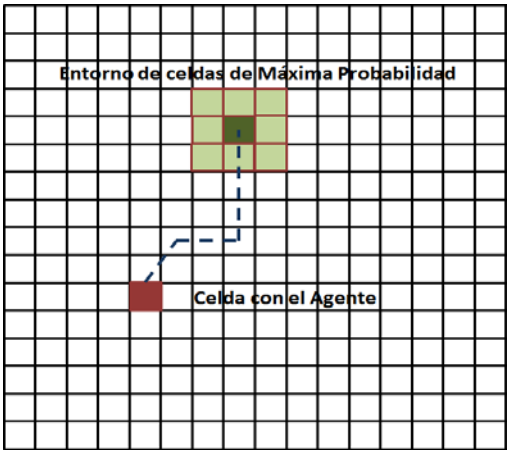


Ilustración 10. Movimiento hacia al Entorno de Mayor Probabilidad (MoveMostImportantRoundCell)

3.1.3 Tipos de Condiciones

Las condiciones realizarán operaciones sobre las probabilidades de las casillas devolviendo un valor que aparecerá en los programas evaluado dentro de una expresión lógica. Tras una expresión lógica habrá anidada otra expresión o, al final, un movimiento seleccionado.

Como se ha comprobado con los movimientos, y de forma más evidente con las condiciones, se trata de métodos diseñados siguiendo ciertas “intuiciones”, sin tener ningún conocimiento previo sobre su operatividad final. Es claro que existen muchas alternativas posibles que podrían haberse planteado, y que dar con las más apropiadas es la clave para conseguir la

resolución del problema. Las condiciones devuelven un rango de valores y, durante las pruebas, se verificó que algunas condiciones retornaban prácticamente siempre el mismo valor de ese rango sin que aparecieran los otros valores, por lo que su capacidad discriminativa era escasa (por ejemplo, medias de cuadrados de distinto tamaño en torno a la celda, probabilidad aún no recogida del mapa comparada respecto a un determinado valor, rotaciones de filas y columnas antes de aplicar MoveLocalAxial, evaluación de la simetría en líneas que atraviesan la celda actual, o estar a una determinada distancia de una concentración de probabilidad, etcétera). Por otro lado, el exceso de condiciones, sobre todo cuando el rango de los valores que devuelven es grande, amplia el espacio de búsqueda. Como ya se ha comentado, el uso de funciones aparentemente "extrañas" es un procedimiento habitual en la PG, aunque puede dificultar y ralentizar el proceso de optimización del programa controlador.

3.1.3.1 Condición Media de Probabilidad en Torno a la Celda Respecto al Resto de Probabilidad (CondLocalGlobal)

Se cogen las celdas que rodean en un cuadrado a la celda actual, y se calcula la probabilidad media del cuadrado. Después se calcula la probabilidad media del resto de las casillas no incluidas en el cuadrado anterior. Finalmente, se comparan ambas medias, pudiéndose obtener 2 valores diferentes como resultado de dicha comparación:

1. Si la probabilidad del cuadrado interior es mayor o igual al exterior, devuelve un 1
2. Si es menor, un 2.

Actualmente un tamaño del cuadrado de 3 casillas se ha obtenido realizando pruebas como el de mejores resultados y no se ha parametrizado este valor para evitar aumentar el espacio de búsqueda.

3.1.3.2 Condición Variabilidad de la Probabilidad en Torno a la Celda Respecto a la Variabilidad Fuera de ese Entorno (CondSAroundGlobal)

Desde la celda actual se componen 4 cuadrados teniendo como esquina la celda y se calcula la varianza de dichos cuadrados. Se devuelve el cuadrado de mayor varianza. Se evalúan los cuadrados por orden Nordeste, Noroeste, Suroeste y Sudeste, y en caso de empates, si uno de los ganadores no sigue el movimiento anterior, se devuelve el primero que se obtuvo.

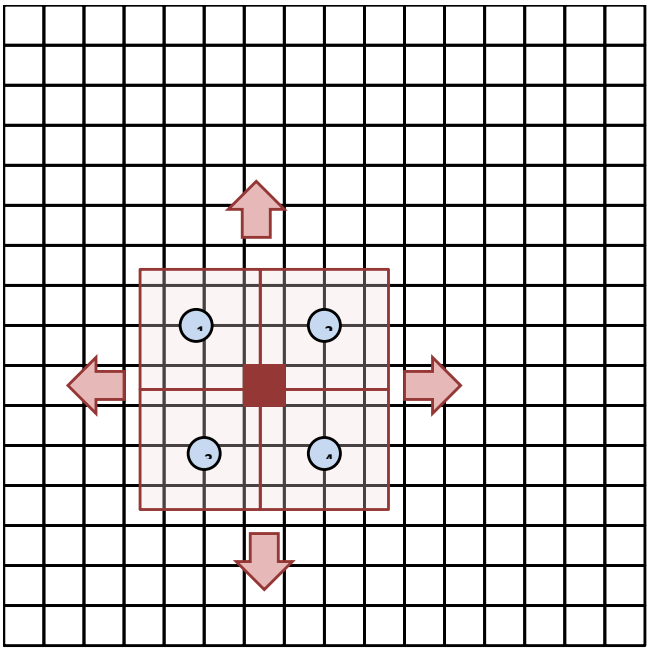


Ilustración 11. Movimiento CondSAroundGlobal.

El objetivo pretendido es dirigirse a zonas con variabilidad de probabilidad.

3.1.3.3 *Condición Celda con Mayor Incremento de Probabilidad Respecto a sus Vecinas Situada en el Perímetro de Distancia 1, 2 o 3 (CondBestIncDec)*

De cada celda de la cuadrícula se calcula el incremento de probabilidad que se produce en cada una de sus celdas vecinas en perímetros alejados 1, 2 y 3 celdas. Se escoge la celda con mejor incremento en valor absoluto (incremento o decremento) en cada uno de estos tres perímetros.

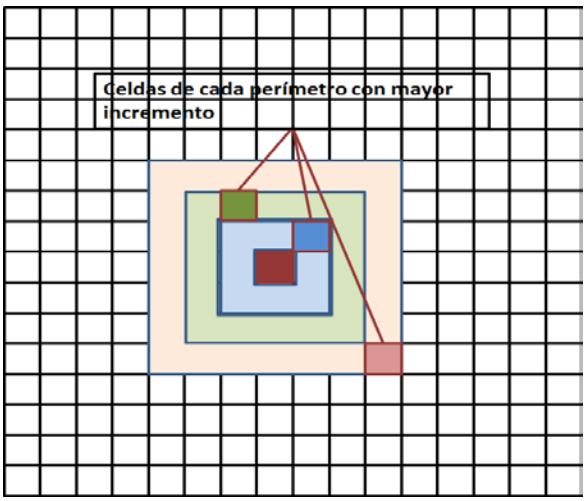


Ilustración 12 Condición CondBestIncDec

El método devolverá 1 si el mejor incremento-decremento se encuentra en una celda del perímetro a distancia 1, 2 si lo está en el perímetro a distancia 2, y 3 si lo está en el de distancia 3.

El objetivo es focalizar el siguiente movimiento hacia una distancia de 1, 2 o 3 de la celda actual.

3.1.3.4 Condición Mejor Tipo de Movimiento Simple Mínimo en Torno a la Celda (CondBestLet)

Probando los movimientos se ha comprobado que concatenando movimientos del tipo MoveLocalAxial de tamaño 1 (moverse una casilla), pueden obtenerse todas las rutas posibles. Un movimiento LRight o LLeft de tamaño 1 de base y cero de altura, sería equivalente al anterior. Si se combinan tamaños de base y altura de la L se pueden hacer barridos de celdas en torno a la celda local puesto que, como se comentó, un movimiento en L al valorar las 8 direcciones está explorando una zona prácticamente circular del entorno.

La idea es utilizar los movimientos con tamaños pequeños para explorar los alrededores de la celda, comprobando cuál de estos mini-movimientos da mejor resultado. Esta forma de proceder se asemeja a realizar una búsqueda miope en torno a la celda actual. Los movimientos escogidos son:

- MoveLocalAxial de tamaño 1. Si fuera el mejor respecto de los demás se devolvería 1.
- MovelLocalAxial de tamaño 2. Idem, devolviendo 2.
- LRight de base 1 y altura 1. Idem, devolviendo 3.
- LLeft de base 1 y altura 1. Idem, devolviendo 4.
- LRight de base 1 y altura 2. Idem, devolviendo 5.
- LLeft de base 1 y altura 2. Idem, devolviendo 6.
- LRight de base 1 y altura 1. Idem, devolviendo 7.
- LLeft de base 2 y altura 1. Idem, devolviendo 8.

Las rutas que se obtienen en los métodos anteriores tienen diferente tamaño (por ejemplo MoveLocalAxial contiene una casilla, mientras LRight de base 1 y altura 2 contiene 3 casillas) por lo que para poder comparar los LETs asociados a cada movimiento, éstos son normalizados dividiéndolos por el número de casillas. Está normalización no es correcta, puesto que el valor obtenido de LETlleva implícito un orden en el que las probabilidades de las primeras casillas son más importantes que las probabilidades de las casillas sucesivas. Sin embargo, y dado que el número de casillas es pequeño, se admite esta normalización de forma excepcional.

3.1.3.5 Movimiento Distancia Entre las Celdas que Darían una Mejor Ruta de Tamaño Tres Uniendo Celdas Vecinas de Máxima Probabilidad, Situadas en los Perímetros a Distancia 1, 2 y 3 (CondBestIncByCell)

Puesto que se han calculado los incrementos de probabilidad de cada celda respecto de todas sus celdas vecinas en las 8 orientaciones, se puede componer un movimiento de tamaño 3 a partir de cada celda que siga los mejores incrementos.

Los cálculos se realizan para todas las celdas en los perímetros a distancia 1, 2 y 3 de la celda actual. Es decir, para cada una de las celdas de esos perímetros se calcula un movimiento que siga las 3 mejores vecinas, seleccionándose por perímetro aquella celda que obtenga mejor LET para ese movimiento.

Habrá una celda de mejor LET en el perímetro a distancia 1, otra en el perímetro a distancia 2, y otra en el perímetro a distancia 3.

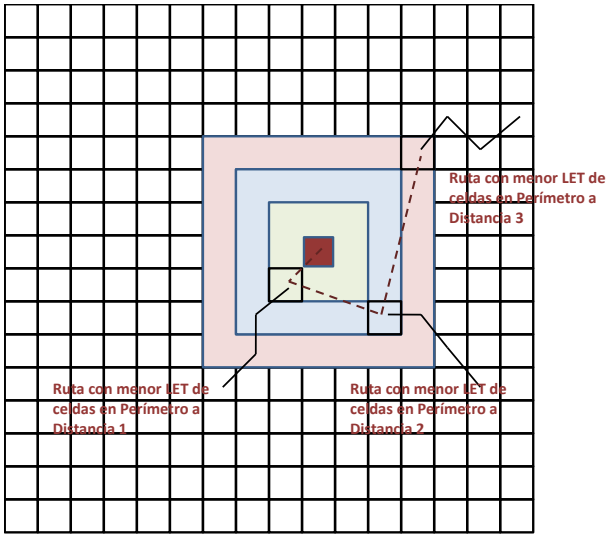


Ilustración 13 Condición CondBestIncByCell

Está condición devuelve la distancia entre las tres celdas mejores de los tres perímetros. El cálculo de la distancia está motivado porque si la distancia entre las tres celdas de los tres perímetros es pequeña (mínimo 3), las condiciones que dan rutas óptimas podrían ser consecutivas, muy cercanas, y un movimiento como MoveLocalAxial (que avanza de celda en celda sin cambiar la dirección) podría componer esas rutas con facilidad. Si la distancia es grande (1 + 3 + 5 como máximo), el intentar ir a la mejor celda vecina podría no ser una buena solución, puesto que el agente podría estar alejándose de celdas posteriores con más probabilidad. Pese a utilizarse 3 perímetros no deja de ser un método con cierta miopía puesto que ir a la vecina con más probabilidad no anticipa que se seguirá encontrando posteriormente celdas con buena probabilidad.

3.1.3.6 Condición Orientación con Mayor Covarianza (CondMoveCov)

Como se comenta en (Murphy, 2012), haciendo referencia a una figura que aparece en la wiki[3], los valores de las covarianzas sobre probabilidades conjuntas están asociados a las diferentes formas que pueden adoptar los valores de las posiciones (x,y), es decir, las formas de la probabilidad, el perfil de como varía la probabilidad en torno a una celda (x,y). El perfil de variación de probabilidad sobre una celda puede influir en el tipo de movimiento que pueda ser óptimo.

Se calculan covarianzas entre filas o columnas consecutivas de las que rodean a la fila actual. Estas covarianzas se calculan para las 8 orientaciones. Puesto que se necesitan dos conjuntos para calcular una covarianza, se seleccionan las celdas a una distancia 1-2 y distancia 2-3 de la celda actual. En la figura se ven todas las situaciones (la celda actual se ha repetido en cuatro posiciones diferentes para dejar claro con que celdas se calculan las covarianzas), y no porque existan cuatro agentes. Por ejemplo, las celdas para la covarianza norte a distancia 1-2 lo forman las tres celdas de la fila sobre la celda actual junto con las otras tres celdas sobre esta

[3] http://en.wikipedia.org/wiki/File:Correlation_examples.png

misma fila. Para la covarianza a distancia 2-3, se procede de igual modo entre las filas de 3 celdas a una distancia norte de 2 y de 3. Equivalentemente se calcularían para los conjuntos de tres celdas de todas las orientaciones mostradas en la figura.

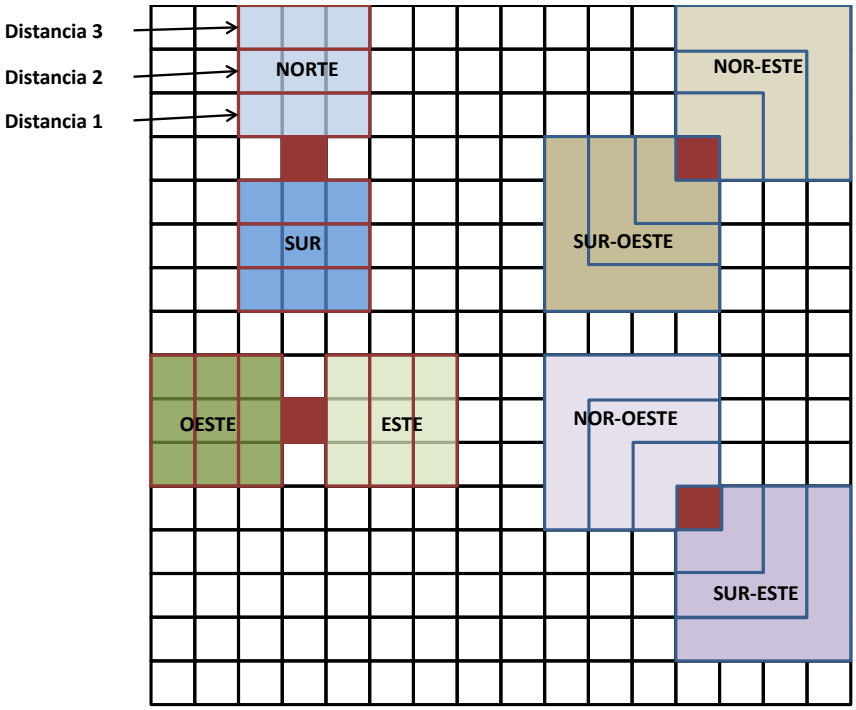


Ilustración 14 Condición CondMoveCov

Con el fin de realizar las comparaciones, las covarianzas se transforman en coeficientes de correlación variantes entre 1 y -1 (-1 para una correlación negativa, lo que implica que si la probabilidad sube en un conjunto bajará en el otro). Cuanto menores sean los coeficientes de correlación menor relación existirá entre los conjuntos comparados. La condición calculará 8 coeficientes de correlación para las 8 orientaciones de los conjuntos de tres celdas a distancias 1-2 y otros ocho a la distancia 2-3.

El valor devuelto por la condición considera la diferencia entre coeficientes de correlación de distancia 1-2 y 2-3, para cada orientación. Se restan, en cada orientación, ambos coeficientes y el que tenga mayor diferencia valor absoluto (que supondría un cambio de situación de probabilidad) es el que se considera.

3.1.4 Gramática Para Generar el Programa Controlador.

La estructura de la gramática producirá programas con movimientos simples en función de las condiciones de probabilidad, de tal modo que las condiciones formarán parte de expresiones con operadores lógicos y los movimientos serán el resultado tras cumplirse las condiciones.

Tanto las condiciones como los movimientos son funciones programadas en Java. Los movimientos serán nodos terminales. Las condiciones formarán parte de los nodos internos en expresiones, y serán las funciones de la gramática.

La gramática define 3 rangos de valores de los que recoger parámetro que hay que asociar a los movimientos que lleven tamaños de celdas asociados, así como para dar valor en las expresiones lógicas a los valores devueltos por las condiciones.

La gramática incluirá reglas de tipo IF<antecedente> THEN <consecuente> ELSE <consecuente> y bucles tipo "while" y "for". Los bucles “for” se utilizarán para aplicar repetidamente el mismo tipo de movimiento un número determinado de veces. El bucle while incluirá bloques y estructuras con reglas, es decir, conjuntos de reglas a aplicar repetidamente hasta conseguir una trayectoria de 10 movimientos.

En las siguientes tablas se exponen los elementos de la gramática. Se irá presentado la gramática desde sus elementos terminales y se comentarán los elementos de la misma que posteriormente sufrieron algún cambio.

3.1.4.1 Elementos Terminales

RANGOS DE ATRIBUTOS
<digit> ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 <range_1_2> ::= 1 | 2 <range_1_3> ::= 1 | 2 | 3 <range_1_4> ::= 1 | 2 | 3 | 4 <range_1_5> ::= 1 | 2 | 3 | 4 | 5 <range_1_8> ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8

OPERADORES LÓGICOS
<logical_operator> ::= <eq> | <neq> | <l> | <le> | <s> | <se> <eq> ::= "==" <neq> ::= "!=" <l> ::= ">" <le> ::= ">=" <s> ::= "<" <se> ::= "<="

ELEMENTOS TERMINALES – TIPOS DE MOVIMIENTOS
<MoveAxial> ::= MoveAxial ";" <MoveLocalAxial> ::= MoveLocalAxial [<digit>] ";" <MoveByGlobalQ> ::= MoveByGlobalQ ";" <MoveByLocalQ> ::= MoveByLocalQ [<range_1_3>] ";" <MoveByRowCol> ::= MoveByRowCol ";" <LRight> ::= LRight [<range_1_5>,<range_1_5>] ";" <LLeft> ::= LLeft [<range_1_5>,<range_1_5>] ";" <MostImportantCell> ::= MostImportantCell ";" <MostImportantRoundCell> ::= MostImportantRoundCell ";"

ELECCIÓN DEL TIPO DE MOVIMENTO DE LOS TERMINALES

```
<move_block> ::= "{" <moves> "}"

<moves> :: = <move_expression> ";"
   | <move_block>

<move_expression> ::= <MoveByGlobalQ>
   | <MoveByLocalQ>
   | <MoveByRowCol>
   | <LRight>
   | <LLeft>
   | <MoveLocalAxial>
   | <MoveAxial>
   | <MostImportantCell>
   | <MostImportantRoundCell>
```

3.1.4.2 Funciones

Las condiciones son las expresiones lógicas con las condiciones anteriormente descritas.

CONDICIONES - FUNCIONES QUE DEVUELVEN UN VALOR A EVALUAR

```
<CondLocalGlobal> ::= CondLocalGlobal [<range_1_3>]
<CondSAroundSGlobal> ::= CondSAroundSGlobal
<CondBestLetMethod> ::= CondBestLetMethod
<CondBestIncDec> ::= CondBestIncDec
<CondDistCellsBestLet> ::= CondDistCellsBestLet
<CondMoveCov> ::= CondMoveCov
<CondBeWithinAGroup> ::= CondBeWithinAGroup
```

Operaciones lógicas con las condiciones.

COMPOSICIÓN DE CONDICIONES

```
<block_logical> ::=  <conditional_expression>
     | ( "!" <conditional_expression> )
     | <or_conditional>
     | <and_conditional>
     | <block_logical>

<and_conditional> ::= ( <logical_expression> "&&" <logical_expression> )
<or_conditional> ::= ( <logical_expression> "||" <logical_expression> )

<conditional_expression> ::= (<CondLocalGlobal> <logical_oepartor_eq> <range_1_2>)
   | (<CondBestLetMethod> <logical_operator> <range_1_8>)
   | (<CondSAroundSGlobal> <logical_oepartor_eq> <range_1_4>)
   | (<CondBestIncDec> <logical_operator> <range_1_3>)
   | (<CondDistCellsBestLet> <logical_operator> <range_1_3>)
   | (<CondMoveCov> <logical_operator> <range_1_8>)
```

En una primera versión de la gramática las reglas que hay que aplicar no se encuentran anidadas, declarándose simplemente tipos de sentencias con condiciones lógicas.

SENTENCIAS POSIBLES EN JAVA
<while_statement> ::= "while " "(" <logical_expression> ")" <statement> <if_statement> ::= " if " "(" <logical_expression> ")" <statement> " else " <statement> <for_statement> ::= "for(" <logical_for> ")" <statement> <logical_for> ::= "int ifor=0;ifor<" <upper_for_size> ";++ifor"

AGRUPACIÓN DE SENTENCIAS
Tipos de Declaraciones <statement> ::= <move_expression> | <if_statement> | <while_statement> | <for_ifs_statement>

NODO RAÍZ
<program> ::= <statement>

Esta gramática se ha diseñado con el objeto de agrupar movimientos en bloques, según condiciones, sin utilizar reglas anidadas, incluyendo estos bloques en bucles “while” con el fin de que pueda repetirse la aplicación de las reglas hasta conseguir N=10 movimientos .

3.2 Cálculo de la Aptitud de los Individuos o Programas

El objetivo es que la ruta generada por el programa minimice el tiempo necesario para detectar un objetivo. El parámetro utilizado es el mismo que en los trabajos previos realizados en el Departamento de Arquitectura de Computadores y Automática: el **Tiempo Local Esperado o LET (Local Expected Time)**. Este parámetro se calcula a partir de las probabilidades de las celdas visitadas durante la trayectoria generada por el programa controlador.

Puesto que el alcance de este estudio está limitado al caso estático (los objetivos no se mueven) con un sensor ideal, el cálculo del LET puede realizarse a partir de las probabilidades de las celdas por las que pasa el agente. Siendo así, el LET puede expresarse como la probabilidad de no encontrar el agente en el primer paso, más la probabilidad de no encontrarlo en el segundo paso, y así hasta completar todas las celdas de la trayectoria. Si la probabilidad de encontrar el agente es la de la celda, la de no encontrarlo es (1 - probabilidad celda) en el primer paso. En los pasos sucesivos, será (1 - probabilidad acumulada sobre las celdas por las que ha pasado el agente).

Se analizarán distintos tipos de mapas en busca del programa adecuado para resolverlos. También se analizarán conjuntos de mapas pues, como se comentará, estos pueden agruparse por responder a la misma tipología de distribución de probabilidad.

3.3 Comentarios Adicionales sobre la Gramática Utilizada

Al describir la gramática ya se ha comentado que en una primera versión se utilizó un conjunto de condiciones LET de “IF-THEN-ELSE”, que formaban reglas que se incluían en bloques de tipo “while” y tipo “for”. Como se verá en el Capítulo 4, los resultados con esta gramática no fueron

suficientemente buenos y se realizó una sencilla modificación de la gramática en la que las condiciones “IF-THEN-ELSE” se han anidado dentro de otras condiciones “IF-THEN-ELSE”, hasta terminar la anidación con un tipo de movimiento. La idea es la misma, aplicar un conjunto de reglas en un bucle “while” o “for”, pero ahora estas reglas estarán formadas por anidaciones “IF-THEN-ELSE”.

Además, durante las pruebas se comprobó que en algunos casos el programa resultante generado a partir de la gramática no producía movimientos. Esto es así porque las reglas comienzan por unas condiciones y, en el caso de que ninguna se cumpla, no existirá movimiento a aplicar. Con el fin de evitar este problema se ha añadido al final de la gramática una condición que evalúa si se ha producido en esa pasada del bucle general algún movimiento y, de no ser así, se genera un nuevo bloque de condiciones finalizando con una re-evaluación de si hubo movimiento, que de no haberse producido, hará que se ejecuté una última condición insertada en la gramática que obligue al agente a realizar un movimiento eligiendo al azar entre los tres tipos de movimientos (MoveLocalAxial de tamaño 1, MoveByLocalQ, o MoveByRowCol) que más aparecen en los programas generados y que, individualmente, han producido mejores LETs.

CAPÍTULO 4. Ejemplo Ilustrativo y Resultados

Se comenzará con unos ejemplos ilustrativos sobre el que se aclaran los puntos más relevantes de las pruebas realizadas, se describen los tipos de mapas utilizados, las optimizaciones probadas y como se determinan y comparan los resultados. Se realiza la comparación entre CEO y PG para un controlador individual por mapa mediante la prueba de Wilcoxon (Knowles, 2006) para lo que se necesitan dos conjuntos de muestras de CEO y PG que se obtienen repitiendo 20 optimizaciones sobre cada uno de los mapas.

Tras mostrar los ejemplos ilustrativos, se presentan los resultados del presente trabajo. Se comprueba que para la optimización de un único mapa CEO y PG tienen resultados similares. Sin embargo la optimización de CEO es mucho más rápida que la de PG.

El siguiente paso es mostrar los resultados de PG en la obtención de un controlador genérico para todos los mapas de un mismo tipo, o de cualquier mapa en general. Esta generalización no puede producirse en CEO puesto que este método se limita a optimizaciones mapa a mapa. Además, tras el proceso de optimización, el programa obtenido con PG estaría disponible para aplicar a cualquier mapa y su ejecución sería muy rápida. Durante este estudio, se realizan 20 optimizaciones obteniendo 20 programas-controladores sobre los que se aplicar la prueba de Wilcoxon, con resultados peores en el caso de PG que en el caso de. Sin embargo, pueden valorarse conjuntamente los resultados de los 20 programas y escoger la mejor solución propuesta por los 20 programas.

Como se ha comentado, ciertos tipos de movimiento planteados en la gramática tienen atributos cuyos valores deben encontrarse en la optimización genérica. Por ejemplo, en los movimientos en L, debe definirse la altura y anchura de la L, y en otros tipos de movimientos como “MoveLocalAxial”, debe definirse el número de casillas a abarcar por ese movimiento. Cuando se optimiza mapa a mapa, PG puede resolver los valores de esos atributos concretando los valores para cada mapa, pero en el proceso de optimización genérica los movimientos en L (LLeft y LRight) desaparecen en las soluciones obtenidas, lo que hace pensar que no existen valores para los atributos de esos movimientos que puedan aplicarse de modo genérico a todos los mapas.

A continuación se mostrarán algunas trayectorias generadas por los controladores genéricos que puedan dar ideas de los comportamientos de las soluciones encontradas ante ciertas características de los mapas, y se mostrarán los tiempos y características del proceso de optimización.

4.1 Ejemplos Ilustrativos del Proceso de Optimización.

Como primera aproximación para encontrar una solución de PG que nos permita obtener un programa que sea capaz de controlar al UAV en el problema de búsqueda en tiempo mínimo se han clasificado los mapas de prueba en tres grupos. Las tres tipologías de mapa contempladas sobre las que se desea obtener un controlador genérico son:

1. Mapas con una única concentración de probabilidad (OneRandomStatic). Este tipo de mapas genera una distribución de probabilidad en el enrejado 20x20 que la concentra en una única zona.

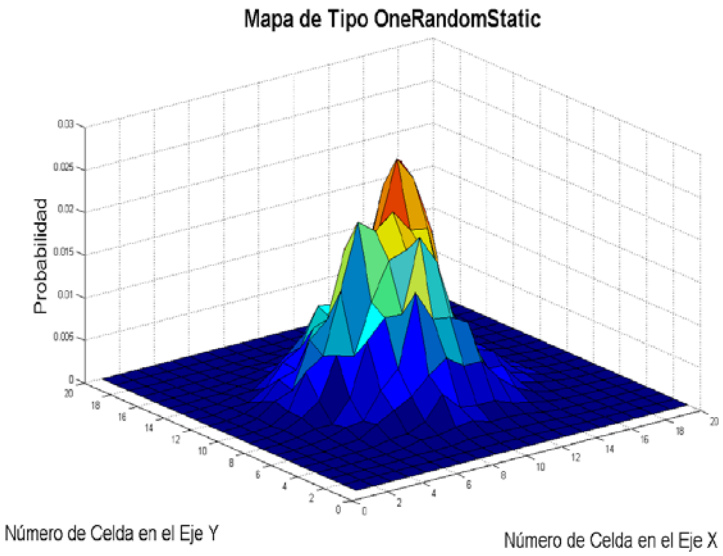


Ilustración 15. Mapa con una Única Concentración de Probabilidad (OneRandomStatic)

2. Mapas con dos concentraciones de probabilidad (TwoGaussStatic). Estos mapas presentan dos concentraciones de probabilidad, cada una de ellas según una gaussiana.

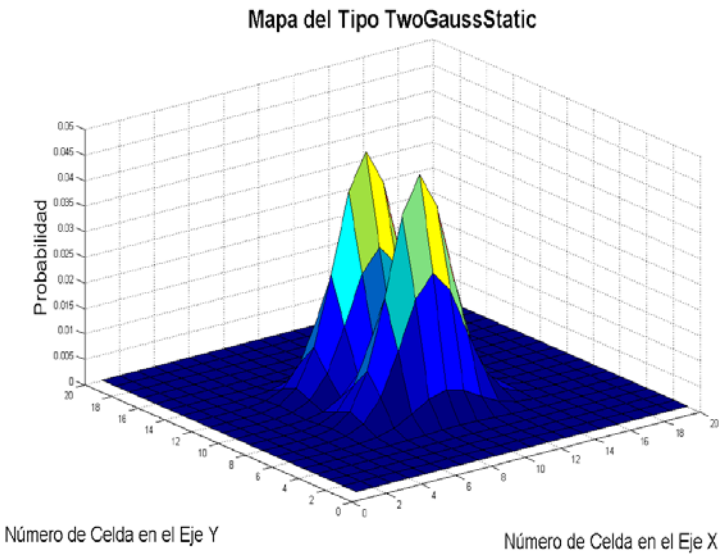


Ilustración 16. Mapa con Dos Concentraciones de Probabilidad (TwoGaussStatic)

3. Mapas con varias concentraciones de probabilidad (SeveralGaussStatic). Presentan varias concentraciones de probabilidad gaussiana:

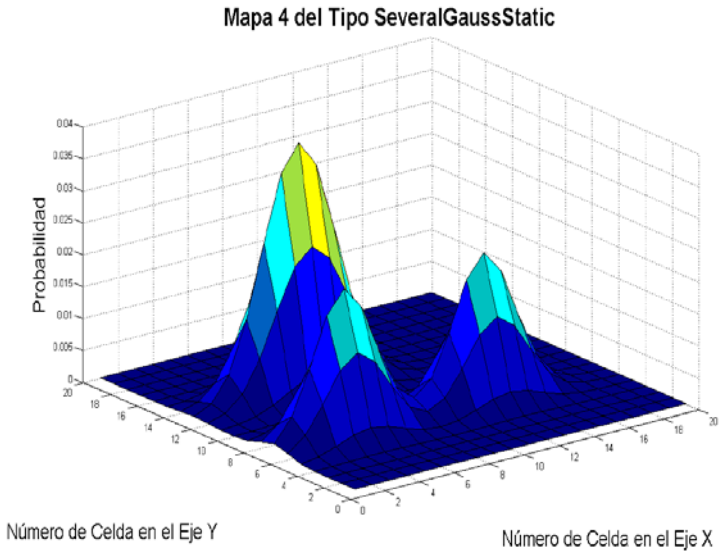


Ilustración 17. Mapas con Varias Concentraciones de Probabilidad (SeveralGaussStatic)

Para todos los mapas, la celda de posición inicial del agente se genera aleatoriamente.

El valor de LET como función de aptitud es apropiado para encontrar el programa controlador óptimo para un mapa. Es decir, la búsqueda de trayectorias de agentes en tiempo mínimo sobre un mapa de probabilidad dado se puede formular con un problema de minimización del LET, en el que los mejores programas-controlador son aquellos que obtienen trayectorias de bajo LET.

4.1.1 Cálculo del LET sobre Cada Mapa.

La primera prueba consiste en realizar una optimización PG en cada uno de los tres mapas ejemplo por separado, lo que devolverá un programa solución y su LET por mapa.

1. Optimización PG para un mapa del tipo **OneRamdonStatic**. El programa obtenido tras realizar la optimización del mapa ejemplo es:

```
while (true) {
        if ((!(CondBestIncDec>3)))
                for(int ifor=0;ifor<5;++ifor){
                        {MoveLocalAxial[1];;}}
        else
                if (((!(CondDistCellsBestLet==2))&&(!(CondBeWithinAGroup==1))))
                        for(int ifor=0;ifor<5;++ifor){
                                {LRight[3,1];;}}
                else
                        for(int ifor=0;ifor<2;++ifor){
                                {{MoveByRowCol;;}}}}
```

Ejecutando el programa sobre el mapa se obtiene un LET de 9.046251197. Este LET coincide con el de CEO, tomado como referencia para evaluar los resultados. En la

Ilustración 18 se muestra la probabilidad inicial del mapa y la trayectoria común ([NE][NE][NE][E][NE][E][NE][NW][N][E]), obtenida por ambos métodos.

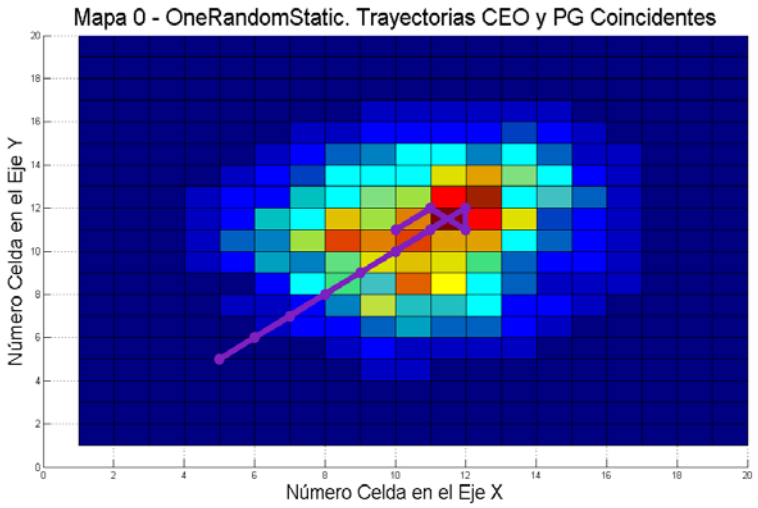


Ilustración 18. OneRandomStatic, Mapa y Trayectoria Resultado con PG y CEO (Coincidente)

2. Optimización PG para el ejmplo del mapa tipo **TwoRamdonStatic**. El programa obtenido es:

```
while (true) {

        if ((CondMoveCov==3))

                for(int ifor=0;ifor<5;++ifor){

                        {MostImportantRoundCell;;}}

        else
                for(int ifor=0;ifor<2;++ifor){{{{
                        {MoveLocalAxial[1];;}}}}}}
```

Analizando el programa se observa que aplica el movimiento MoveLocalAxial de tamaño 1 diez veces, lo que le lleva a seguir la misma trayectoria que la obtenida mediante CEO ([SW][SW][SW][SW][W][W][W][W][W][W])

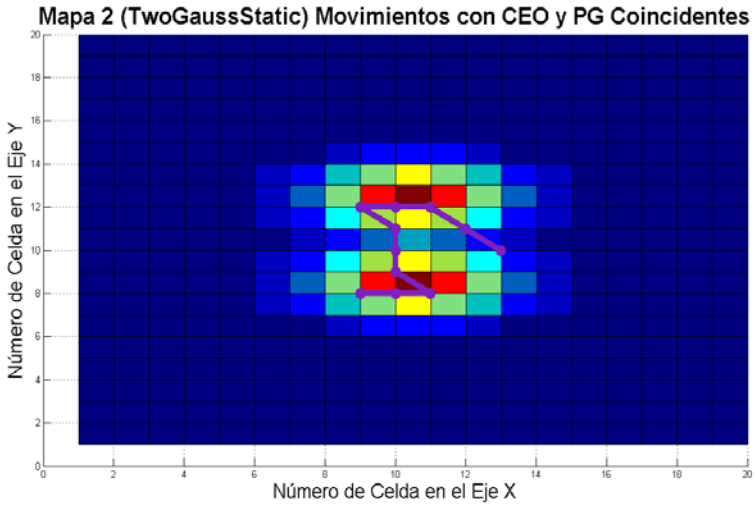


Ilustración 19. TwoGaussStatic, Mapa y Trayectoria con PG y CEO (Coincidentes)

3. Optimización PG para el ejemplo del mapa tipo **SeveralRamdonStatic**. El programa obtenido es:

```
while (true) {
        if
        (((CondDistCellsBestLet>2)||((!(CondBeWithinAGroup<=2))&&((!(CondLocalGl
        obal[1]==2))&&(!(CondBestIncDec>3))))))
                for(int ifor=0;ifor<2;++ifor){{MostImportantRoundCell;;}}
        else
                for(int ifor=0;ifor<1;++ifor){{{MoveByLocalQ[3];;}}}
```

En este caso la trayectoria obtenida por CEO y PG difieren. Además, el LET obtenido por PG es peor que el obtenido por CEO (8.87834 de PG frente a 8.89972 de CEO).

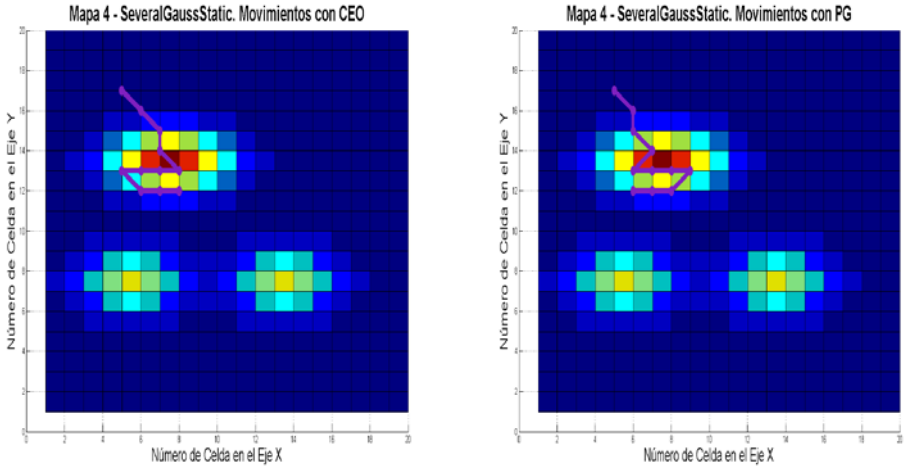


Ilustración 20. SeveralGaussStatic, Mapa y Trayctoria PG (verde) y CEO (rojo)

Si se estudia la estructura de los tres programas, se puede observar que éstos están compuestos por un bucle “while” principal y dentro condiciones “IF-THEN-ELSE” que pueden estar anidadas, aunque en las soluciones obtenidas en estas optimizaciones no se ha producido esta anidación. Dentro de la estructura “IF-THEN-ELSE”, pueden aparecer otros bucles “while” o “for”. Adicionalmente los movimientos se envuelven en bucles “for”, cuyo número de iteraciones se determina en el proceso de iteración, puesto que podría ser conveniente aplicar el mismo tipo de movimiento un número determinado de veces. Para mapas individuales los programas no suelen tener gran tamaño.

4.1.2 Comparación Estadística de las Soluciones entre PG y CEO

Comparar PG y CEO con una única optimización por mapa no es estadísticamente representativo. Al realizar nuevas optimizaciones sobre el mismo mapa podría ocurrir que los resultados obtenidos en la optimización fueran diferentes, ya que tanto PG como CEO son métodos de optimización no deterministas, que no tienen por qué converger a la misma solución.

Por lo tanto, se deben realizar varias optimizaciones sobre cada mapa, tanto para PG como para CEO, para comparar estadísticamente los resultados que se obtienen y comprobar si la solución obtenida es equivalente en cada optimización. La prueba elegida para realizar esta comparación es el método Wilcoxon de Ordenación de Diferencias (Adaptive And Bioinspired Systems Research Group). Esta prueba no realiza ninguna hipótesis sobre la forma de las distribuciones, es aplicable a pocos datos, y permite determinar estadísticamente si la mediana de las muestras es la misma o una es menor que la otra. Las dos muestras son los valores de los LETs obtenidos como resultado que las trayectorias obtenidas por las optimizaciones repetidas sobre los mismos mapas realizadas con PG y con CEO. Puesto que la mediana es el valor del LET en las posiciones centrales, si una de las dos muestras tiene menor mediana, la distribución estará centrada hacia un valor de LET menor, y por tanto el resultado será mejor.

Se ha escogido realizar 20 optimizaciones seguidas tanto para PG como CEO. Por ejemplo, para el primer mapa del tipo OneRandomStatic, los valores de las dos muestras de LETs han sido:

	LETs de 20 optimizaciones sobre un mapa tipo OneRandomStatic
PG	9.046251197,9.046251197,9.046251197,9.046251197,9.046251197,9.046251197,9.046251197,9.046251197,9.046251197,9.046251197,9.046251197,9.046251197,9.049478911,9.047223037,9.046251197,9.046251197,9.046251197,9.046251197,9.046251197,9.046251197
CEO	9.046251,9.046251,9.046251,9.046251,9.046251,9.046251,9.047223,9.046251,9.046251,9.046251,9.046251,9.046251,9.046251,9.046251,9.046251,9.046251,9.046251,9.046251,9.046251,9.046251

Si se aplica Wilcoxon a estas dos muestras, resulta que CEO obtiene mejor resultado que PG , mientras que en el ejemplo que se expuso con una sola optimización sobre el mismo mapa se obtuvo un empate. Por tanto, CEO está funcionando mejor que PG para este mapa, al menos con resultados obtenidos sobre 20 optimizaciones.

La siguiente tabla resume los datos de Wilcoxon para los tres mapas del ejemplo:

	Gana CEO	Gana PG
OneRandomStatic	**X**	
TwoGaussStatic	**X**	
SeveralGaussStatic		**X**

Como puede verse, de tres mapas, CEO se muestra mejor que PG en dos. De este modo y tomando como referencia a CEO se puede valorar la calidad de los resultados para PG.

4.2 Obtención de un Controlador Diferente para cada Mapa

Siguiendo el mismo proceso que en los ejemplos descritos, se realiza una comparativa estadística de los resultados obtenidos por PG y CEO para un mapa dado con las diferentes gramáticas propuestas en este trabajo. Para corroborar la validez del estudio, la comparativa de los resultados obtenidos por PG y CEO pada un mapa dado, se realiza sobre un conjunto de 400 mapas de prueba diferentes, de forma que tanto PG como CEO se utilicen para optimizar el camino del agente de forma independiente sobre cada uno de los mapas.

4.2.1 Resultados con la Gramática Inicial

Siguiendo el mismo proceso que en los ejemplos descritos en el apartado 4.1.2, se han realizado 20 optimizaciones repetidas con un conjunto de 400 mapas de prueba de los tres tipos anteriormente expuestos, tanto con la heurística PG como con CEO, y sobre los resultado obtenidos sobre cada uno de los mapas se ha aplicado la prueba de Wilcoxon. Los resultados obtenidos se recogen en la siguiente Tabla, donde se observa que CEO se muestra considerablemente superior en un número mayor de mapas de prueba.

EMPATES	GANA PG	GANA CEO
84	96	221

Con el fin de mejorar estos resultados iniciales, se buscó una gramática alternativa para hacer que las reglas “IF-THEN-ELSE” tuvieran una estructura anidada con otras reglas “IF-THEN-ELSE”.

4.2.2 Resultados con la Gramática Modificada

La gramática modificada incluye un único bucle while externo, conteniendo reglas anidadas y que se ejecutará hasta que se produzcan 10 movimientos sobre el mapa. Repitiendo el proceso de realizar 20 optimizaciones por mapa para obtener los 20 LETs asociados a PG y los 20 LETs asociados a CEO, y aplicando la prueba de Wilcoxon sobre los dos conjuntos de datos, se obtienen los resultados siguientes.

EMPATES	GANA PG	GANA CEO
69	183	149

Como se observa, los resultados de PG ha mejorado considerablemente, incluso superando a los obtenidos por CEO. Sin embargo, como se mostrará al analizar los tiempos de los procesos

en el apartado 4.4, la optimización mediante PG es muy lenta comparada con CEO. Pero el que PG sepa optimizar individualmente los mapas lo convierte en un método válido de optimización de LET y permite preguntarse si también funcionará adecuadamente en la optimización de un controlador genérico por tipo de mapa, o para todos los mapas.

4.3 Generalización de Resultados. Búsqueda de un Controlador Único por Tipo de Mapa o un Controlador Genérico.

El proceso de encontrar los programas controladores por tipología se realizará del modo habitual en las heurísticas de aprendizaje, dividiendo el conjunto de mapas de cada tipo en dos subconjuntos, uno para aprendizaje y otro para las pruebas.

Usar el LET como función de aptitud es adecuado únicamente para encontrar el mejor programa en mapas individuales. Sin embargo, puesto que ahora se está optimizando un conjunto de mapas que responden a una tipología, es necesario encontrar una función de aptitud que permita comparar los programas-controladores generados por la gramática sobre un conjunto de mapas del mismo tipo.

Al probar un programa sobre el conjunto de mapas pertenecientes a una tipología, se obtiene un LET diferente para cada uno de los mapas. Para optimizar el controlador usando los LETs calculados sobre todos los mapas no se pueden comparar/combinar directamente los valores de todos los LETs, puesto que habrá mapas que tengan un valor de LET mayor que otros y que sin embargo sea un LET óptimo para ese mapa. Es decir, el valor mínimo de todos los LETs de los mapas del mismo tipo no es indicativo de que se trata del mejor LET. Por lo tanto, es necesario combinar/comparar las discrepancias existentes entre el LET obtenido por PG sobre cada mapa con el LET óptimo correspondiente al mismo mapa. Como el valor óptimo del LET es un parámetro desconocido a priori, se sustituye su valor por el obtenido por CEO cada uno de los mapas.

Se proponen dos formas de optimizar el LET por tipo de mapas, siempre en base a la diferencia entre el LET del mapa devuelto por PG y el obtenido por CEO:

- Minimizando la discrepancia máxima obtenida para todos los mapas analizados entre el LET obtenido por PG y por CEO. Es decir, planteando un problema tipo MiniMax, como una minimización (del tiempo esperado) de máximos (diferencia de LET PG y LETCEO sobre cada mapa).
- Sumando todas las diferencias entre el LET de PG y de CEO. El programa que minimice estas diferencias será el mejor controlador.

Finalmente, se elige la primera opción, ya que con ella se obtendrá un controlador que mejora el LET del caso peor, en vez de un programa que mejore en promedio el LET sobre todos los mapas.

4.3.1.1 Resultados de la Optimización en la Obtención de un Controlador Genérico.

Como se hizo anteriormente, para poder aplicar Wilcoxon se necesitan los resultados de varias optimizaciones repetidas. Por tanto, en el aprendizaje, se van a realizar 20 optimizaciones,

obteniendo 20 controladores por tipo de mapa. Cada uno de esos controladores o programas se aplicará sobre los 71 mapas de prueba del mismo tipo, obteniendo como resultado la muestra de 20 LETs de PG necesarios para la prueba de Wilcoxon. Los 20 LETs de CEO se obtienen de forma similar a los casos anteriores: ejecutando 20 veces CEO para cada uno de los 71 mapas del conjunto de prueba.

Para el caso del controlador genérico para cualquier tipo de mapa, se sigue el mismo procedimiento, utilizado 213 mapas de aprendizaje (71 de cada tipo) y 213 de prueba.

En la siguiente tabla se muestran los resultados de la prueba de Wilcoxon para los controladores obtenidos por tipo de mapa y genérico.

	Empates	**Gana PG**	**Gana CEO**
OneRandomStatic	4	0	67
TwoGaussStatic	1	0	70
SeveralGaussStatic	2	1	68
Controlador Genérico	9	0	204

Como puede comprobarse, los resultados demuestran que CEO obtiene mejores resultados siempre. Esto se debe a que CEO no se calcula para un tipo genérico de mapa, sino que se calcula para cada uno de los mapas propuestos.

Para compensar este hecho se ha realizado una sencilla mejora, ya que se dispone de los 20 programas por tipo de mapa procedentes de las 20 optimizaciones, y por lo tanto, en lugar de utilizar un único programa controlador, se pueden aplicar los 20 programas controladores sobre cada mapa de prueba y elegir la solución que proporcione los mejores resultados en cada.

La comparativa en este caso se realiza sobre el mejor resultado obtenido por los 20 programas de PG y el mejor resultado obtenido por CEO para cada mapa. Los resultados se recogen en la siguiente tabla, en la que se observa una notable mejora de los resultados obtenidos mediante PG, que se valorarán por medio de otras técnicas en el apartado 4.3.3.

	Empates	**Gana PG**	**Gana CEO**
OneRandomStatic	22	17	32
SeveralGaussStatic	24	24	23
TwoGaussStatic	25	9	37
Controlador Genérico	69	51	93

4.3.2 Revisión de los Programas Obtenidos en la Optimización Genérica

Para entender las diferencias de resultados obtenidos para los programas-controlador por mapa y el programa-controlador genérico es conveniente analizar los elementos de la gramática que aparecen en los programas obtenidos mediante PG.

Un aspecto importante a resaltar es que en los controladores genéricos los tipos de movimiento LLeft y LRight no aparecen. Estos dos movimientos se definen con dos atributos que dan dimensión al número de casillas de la base y de la altura de la L. Estos valores tienen

un rango de variabilidad de 1 a 5 y es PG, en el proceso de optimización, quien debe encontrar que valores son los más adecuados. Cuando se optimiza para un único mapa estos movimientos aparecen en la solución y, en ocasiones, son con los que consiguen mejorar el resultado respecto a CEO. Sin embargo, al buscar un programa general por tipo de mapa, PG no puede determinar un par de valores de base y altura de la L válidos para todos los mapas del mismo tipo, y por este motivo, estos movimientos desaparecen de los programas-controladores genéricos.

Otros tipos de movimientos parametrizados como MoveAxialLocal si aparecen en los programas-controladores genéricos, con lo que el proceso es capaz de asignar valores al parámetro del movimiento en los lugares en que aparece. Lo mismo ocurre con MoveByLocalQ. Sin embargo la estructura de los programas es compleja, con muchas expresiones lógicas que contienen también muchas condiciones, que se encuentran además anidadas. Además, en algunos casos ocurre que los movimientos que aparecen en los programas nunca se llevan a cabo, ya que las condiciones no permiten alcanzar su ejecución. Por lo tanto, una tarea que hay que realizar en el futuro será incorporar un analizador de código capaz de simplificar la estructura de los programas obtenidos por PG. Además, un analizador de este tipo podría utilizarse para extraer conclusiones que permitiesen incluir nuevos elementos en la gramática propuesta, y mejorar, al igual que ocurrió con la modificación de la gramática del controlador individual, los programas-controles genéricos obtenidos .

4.3.3 Viabilidad de un Controlador Genérico con PG

La prueba de Wilcoxon realizada no permite determinar completamente lo que está sucediendo, ya que solamente muestra el número de veces que un método es mejor que otro. Sin embargo, puede ocurrir que aunque PG sea peor que CEO sus resultados sean aceptables, por lo que también debe valorarse si es significativa la diferencia en la que PG es que CEO.

En los siguientes apartados se muestra una medida que pretende realizar esta valoración y se muestran los resultados obtenidos. También se muestran ciertas trayectorias que ejemplifican el comportamiento de PG en determinados casos.

4.3.4 Valoración de la Diferencia entre los Valores de PG y CEO

Siguiendo el método propuesto en el Trabajo Fin de Master de Judith (Manso Vergara, 2013), se puede valorar el **ahorro** en el tiempo de búsqueda (ahorro de LET) obtenido por aplicar un método, PG, en lugar de otro método, CEO, mediante la diferencia del LET proporcionado por ambos valores. Este ahorro de tiempo puede expresarse del siguiente modo:

$$\Delta = -[(LET_{PG} - LET_{CEO})/(10 - LET_{CEO})]$$

El peor valor de LET para trayectorias de 10 elementos es 10, que es el caso que se presenta cuando no se ha recogido ninguna probabilidad en ningún movimiento. Si el LET de PG es menor al de CEO la medida del ahorro que realiza PG respecto a CEO será positivo, en caso contrario será negativo (ahorra más CEO). A su vez este ahorro se referencia a la calidad del LET de CEO, representable por el valor $(10 - LET_{CEO})$, es decir, por lo que mejora respecto del peor caso que es 10, y que será mayor cuanto más bajo, o mejor, sea el LET de CEO.

Sin embargo, el ahorro de tiempo puede no ser suficiente para analizar las soluciones, puesto que LETS parecidos pueden producir trayectorias diferentes. Es decir, pueden existir alternativas a la hora de definir las trayectorias de uno u otro método de parecida calidad. Por tanto, además del ahorro de tiempo se estimará la similitud de las trayectorias medible por el número de movimientos coincidentes entre ambas. Este valor se denominará **Número de Aciertos**.

En base a estas dos medidas se pasa a describir el funcionamiento de los controladores genéricos obtenidos anteriormente.

4.3.5 Valoración del Ahorro Tiempo de Búsqueda y Número de Movimientos Coincidentes para los Diferentes Tipos de Controladores.

Todos los datos mostrados se han obtenido aplicando los 20 programas de las 20 optimizaciones y escogiendo como resultado el mejor. Se realizará una gráfica con los valores de ahorro de tiempo y aciertos, estableciéndose una banda o margen de ahorro de tiempo de valor 1.5, tanto positivo como negativo, dentro de la cual se considerará que el resultado de PG es aceptable. De este modo el número de mapas que se consideraría que PG ha analizado correctamente respecto a CEO se verá que es superior al 80%.

Todos los datos de las gráficas están ordenados por ahorro de LET, desde los valores negativos, en los que PG pierde respecto a CEO, a valores positivos en los que tiene mejor resultado.

4.3.5.1 Ahorro de Tiempo para el Controlador Genérico de Tipo OneRandomStatic.

En Ilustración 21 se muestran los resultados obtenidos para el caso del controlador genérico para mapas de probabilidad tipo OneRandomStatic. Como se observa, en ella existen muchos valores de ahorro concentrados en la parte izquierda de la figura, donde el comportamiento de CEO es mejor al de PG (parte derecha de la figura con valores positivos).

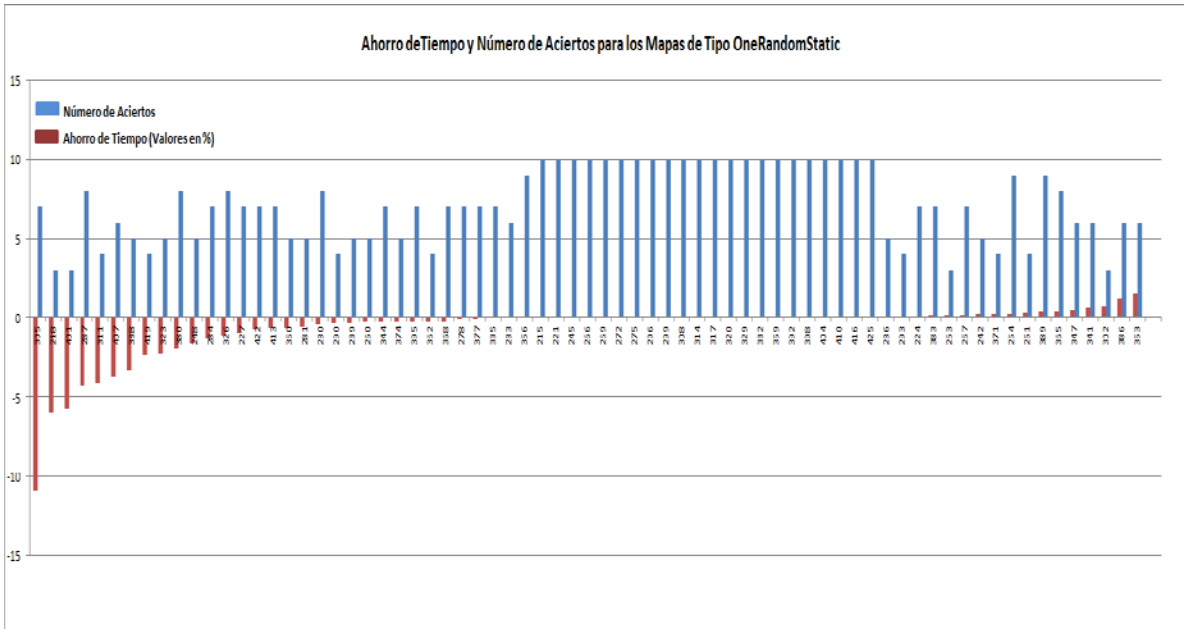


Ilustración 21. Comparativa de Métricas Relacionadas con LET y Número de Aciertos para el Controlador Generalizado de Mapas OneRandomStatic

La gráfica muestra en rojo la evolución del ahorro de tiempo. Cuando es negativo el ahorro de CEO es mayor al de PG (parte izquierda de la gráfica), y cuando es positivo el de PG es mayor al de CEO (parte derecha). Existen valores destacados de ahorro de tiempo de un método frente a otro. Los valores cero representan un empate, y por último hay casos en los que la diferencia es pequeña. En la siguiente tabla se muestran algunos valores de esa diferencia en reducción de tiempo junto al número de resultados de PG que estarían dentro de ese valor. La última columna es la suma de mapas PG dentro de la diferencia con los mapas en los que gana PG:

Reducción de Tiempo (%)	**Número de mapas en el margen de reducción**	**Número de mapas en el margen de reducción más los que gana PG**
0.5	48	53
1	56	58
1.5	59	**60**
2	62	62
2.5	64	64
3	64	64

Con un margen de reducción de 1,5%, que no supone una pérdida de tiempo significativa, el número de mapas resueltos con PG que se posicionan en dicho margen más aquellos en los que gana PG fuera del margen, es de 60 mapas sobre un total de 71. Es decir, PG aporta un 84% de resultados aceptables.

También se observa en la gráfica que, cuando los resultados de CEO y PG tienen poca diferencia, el número de aciertos o coincidencias de las trayectorias es elevado. Sin embargo en algún caso esto no ocurre, pudiendo haber poca diferencia de tiempos con pocos aciertos, mapa 362, o elevada diferencia de tiempos con bastantes aciertos, mapa 287. Como se comentará al mostrar las trayectorias, esto se debe a variaciones locales en la selección de las casillas de CEO y PG, o a realizar un recorrido por las mismas casillas en un orden diferente.

4.3.5.2 Ahorro de Tiempo para el Controlador Genérico de Tipo TwoGaussStatic

En la siguiente gráfica se muestra la evolución de resultados para el controlador TwoGaussStatic, que presenta un comportamiento bastante similar al caso anterior.

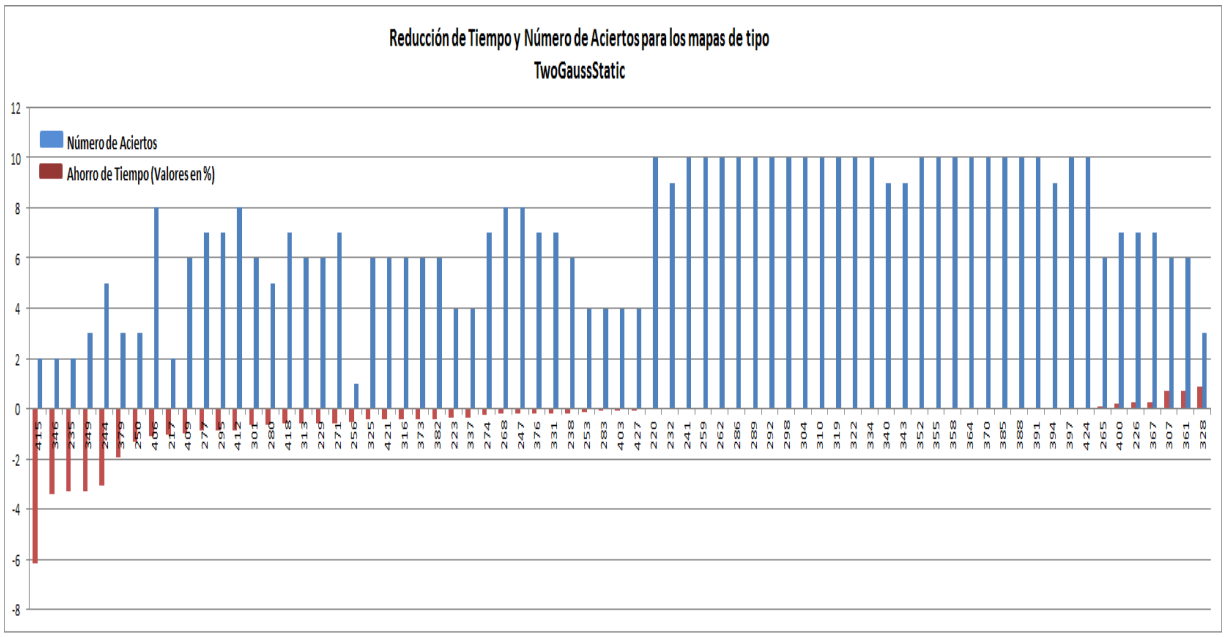


Ilustración 22. Comparativa de Métricas Relacionadas con LET y Número de Aciertos para el Controlador Generalizado de Mapas TwoGaussStatic

Los números de mapas que se encuentran en el margen de reducción y sobre los que PG obtiene resultados aceptables se muestran en la tabla. En este caso, para un valor de reducción de 1,5%, entre los 71 mapas analizados, existen 65 mapas sobre los que PG aporta resultados aceptables. Por lo tanto, los resultados de PG se pueden considerar aceptables en un 90% de los casos.

Reducción de Tiempo (%)	Número de mapas en el margen de reducción	Número de mapas en el margen de reducción más los que gana PG
0.5	48	51
1	61	61
1.5	65	**65**
2	66	66
2.5	66	66
3	70	70

4.3.5.3 Ahorro de Tiempo para el Controlador Genérico de Tipo SeveralGaussStatic.

La representación gráfica para el caso de los controladores genéricos obtenidos sobre los mapas del tipo Several GaussStatic es la siguiente:

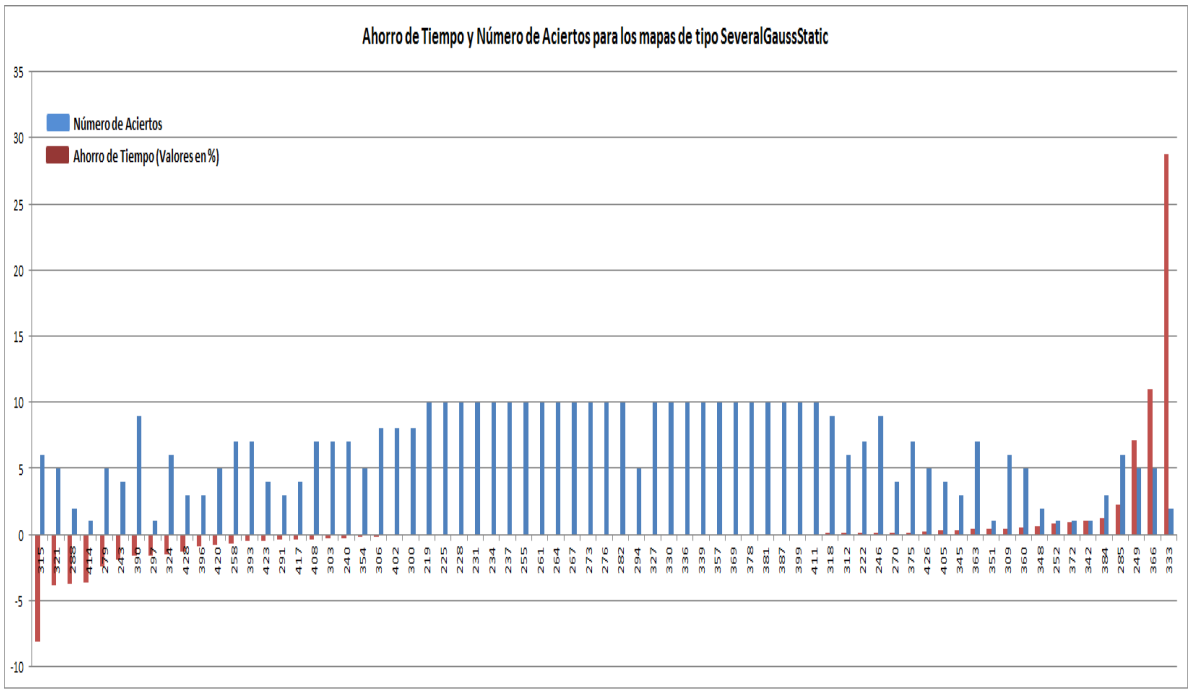


Ilustración 23. Comparativa de Métricas Relacionadas con LET y Número de Aciertos para el Controlador Generalizado de Mapas SeveralGaussStatic.

Los números de mapas que se encuentran en el margen de reducción y sobre los que PG obtiene resultados aceptables se muestran en la tabla. En este caso, con un valor de reducción de 1,5% PG aporta soluciones satisfactorias sobre 62 mapas de 71. Por lo tanto, se considera que proporciona resultados aceptables el 87% de los casos.

Reducción de Tiempo (%)	Número de mapas en el margen de reducción	Número de mapas en el margen de reducción más los que gana PG
0.5	48	58
1	55	61
1.5	59	**62**
2	62	66
2.5	64	67
3	64	67

Finalmente, es importante destacar que en la Ilustración 23 el último mapa a la derecha, 333, aparece muy destacado en reducción de tiempo y, como se verá, esto es así porque elige explorar una región donde hay más probabilidad que la decida por CEO.

4.3.5.4 Ahorro de Tiempo de un Controlador Genérico para Todos los Mapas.

Por último para el análisis de los resultados obtenidos con un controlador genérico para todos los mapas se obtiene la siguiente figura.

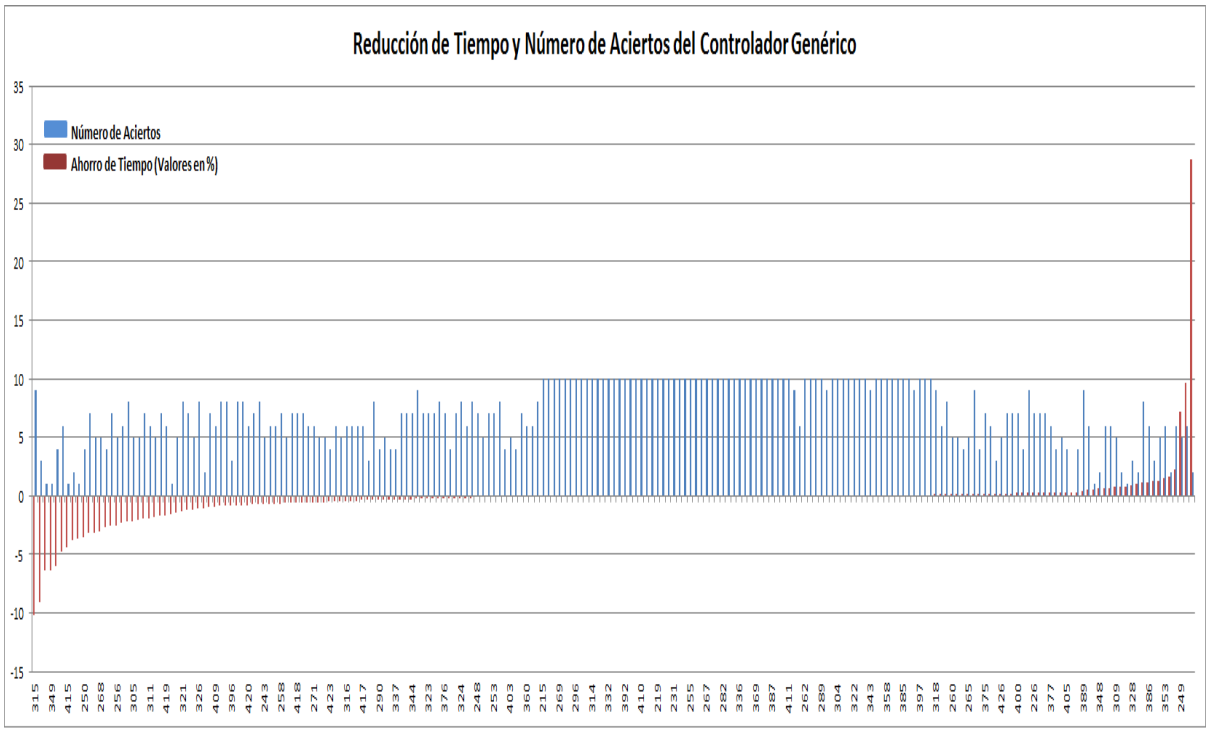


Ilustración 24. Comparativa de Métricas Relacionadas con LET y Número de Aciertos para el Controlador Genérico

Los números de mapas que se encuentran en el margen de reducción y sobre los que PG obtiene resultados aceptables se muestran en la tabla. En este caso, con un valor de reducción de 1,5% PG aporta soluciones satisfactorias sobre 187 mapas de 213. Por lo tanto, se considera que proporciona resultados aceptables el 87% de los casos.

Valor de Reducción de Tiempo (%)	Número de mapas en el margen de reducción	Número de mapas en el margen de reducción más los que gana PG
0.5	137	159
1	168	179
1.5	181	187
2	189	193
2.5	194	197
3	197	200

4.3.6 Casos Particulares Observados al Valorar las Gráficas de Ahorro de Tiempo.

En las gráficas de ahorro de tiempo aparecen casos en los que puede haber buenos valores de LETS con pocos aciertos y viceversa. Analizando este tipo de casos pueden valorarse ciertos comportamientos de los controladores obtenidos. Estos casos se muestran en los siguientes apartados.

4.3.6.1 Diferente exploración Local de la Probabilidad

Existen muchos casos en los que la diferencia entre PG y CEO se debe a tener en sus trayectorias dos o tres celdas diferentes, o las mismas celdas recorridas en distinto orden. Estos pequeños cambios producen que el LET de un método pueda mejorar al del otro. En las

siguientes figuras se muestran algunos de estos casos obtenidos de diferentes controladores genéricos.

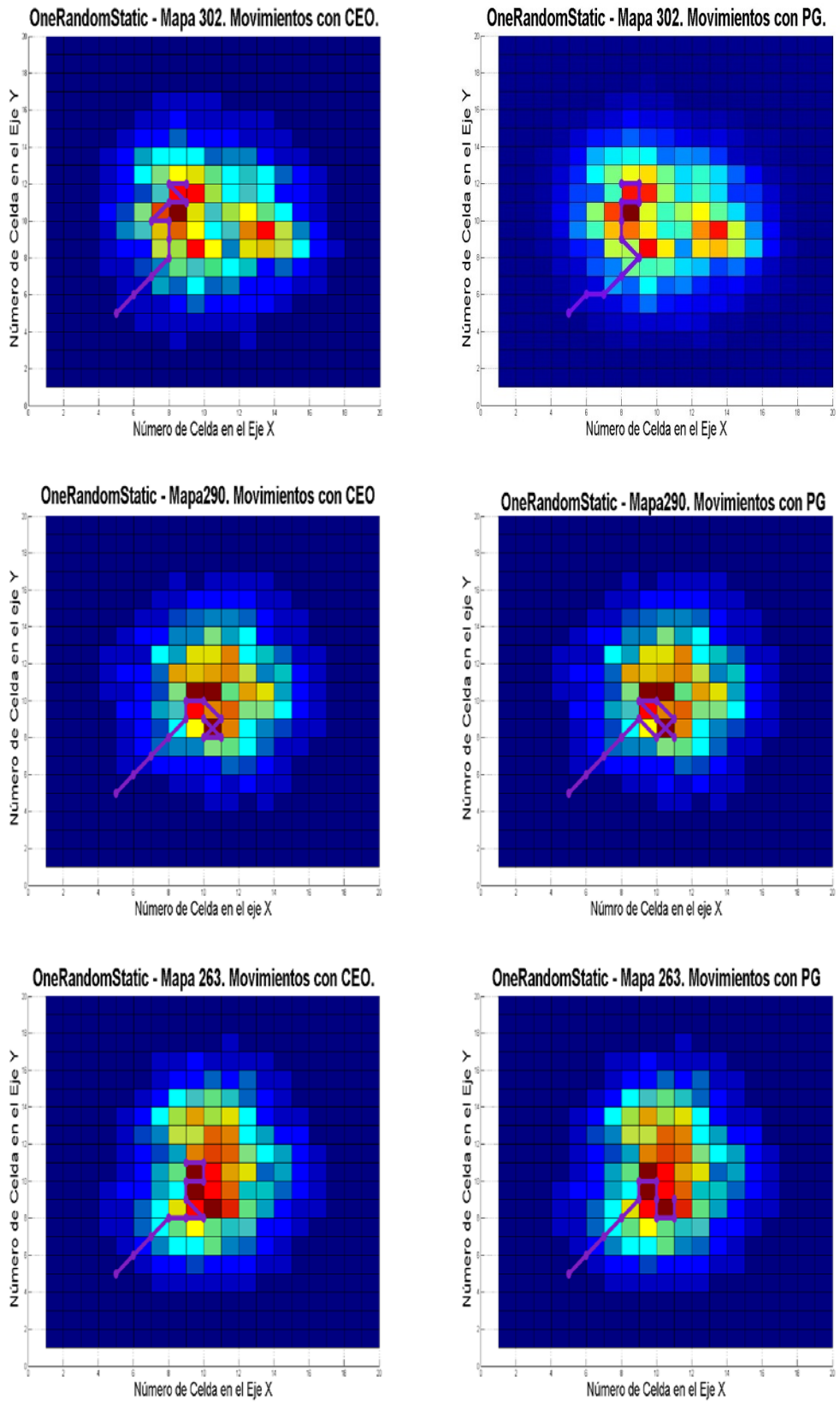
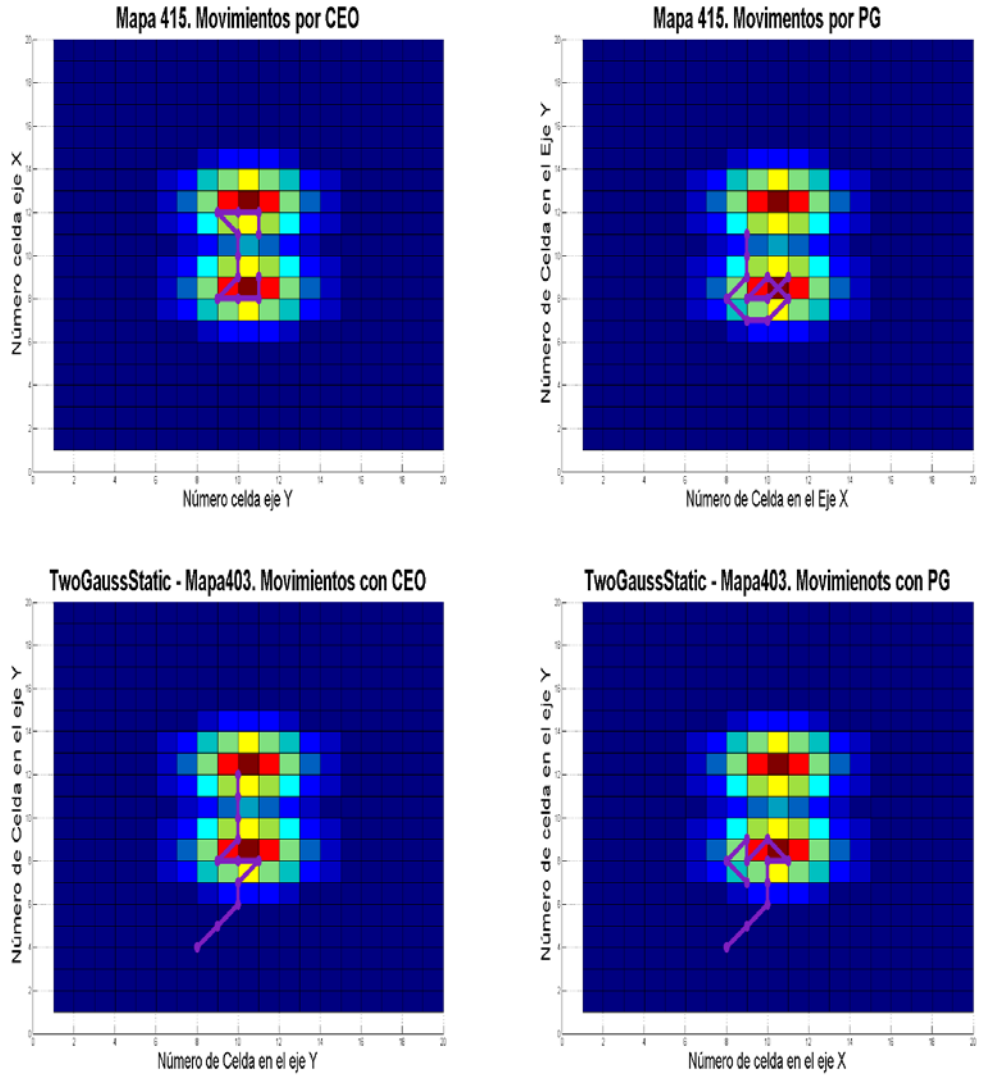


Ilustración 25. Diferencias en las Exploraciones Locales de las Trayectorias CEO y PG.

Estas diferencias en la trayectoria hacen que en unos casos PG mejore respecto a CEO, mientras que en otros empeore. Para valorar si el empeoramiento de PG se debe a que los movimientos de la gramática no están correctamente parametrizados, se optimizan los mismos mapas individualmente con PG, dando como resultado que en los mapas 302 y 263 PG mejora a CEO (en 302 aparece un LRight[3,4] y en 263 un LRight[1,1]), encontrando mejores trayectorias, y que en el mapa 290 le iguala los resultados (mediante un LRight[1,4]). Parece claro que la deficiencia de parametrización de los métodos en el controlador genérico empeora los buenos resultados que obtiene el controlador individual en los movimientos locales, y estos son importantes para determinar las diferencias de LETs.

4.3.6.2 Quedarse en una Zona Local cuando hubiera sido Preferible Moverse a Otra Región de Mayor Probabilidad

En los mapas representados en la figura siguiente, la trayectoria se queda explorando una zona mientras que haberla abandonado hacia otra de mayor probabilidad hubiera dado mejores resultados.



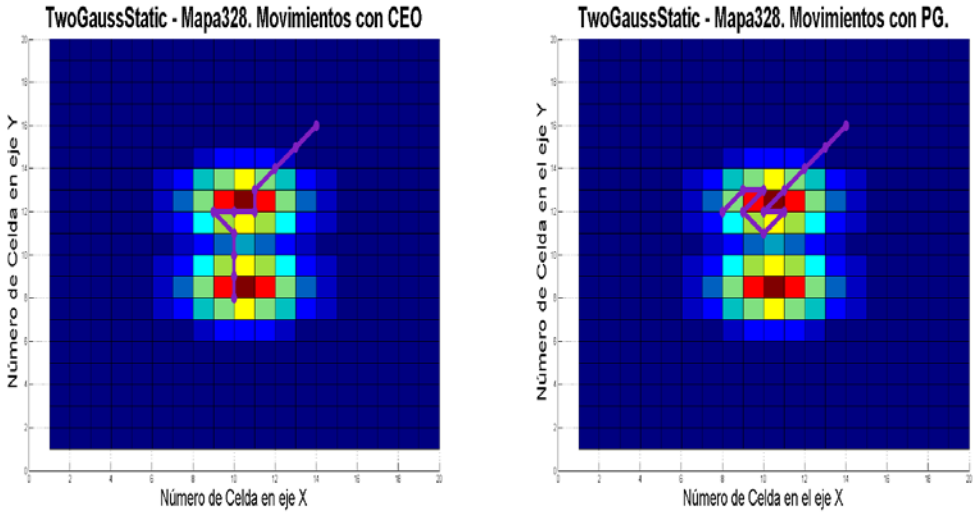


Ilustración 26. Trayectorias que Cambian la Región de Exploración Respecto de Aquellas que se Mantienen en una Zona.

Tras observar las trayectorias del método PG de la Ilustración 26 podría concluirse que PG está cayendo en mínimos locales. Procediendo como en el apartado anterior para ver si esto se debe a la mala parametrización de los métodos, a continuación se muestran las trayectorias obtenidas para los mismos mapas con una optimización individual mapa a mapa:

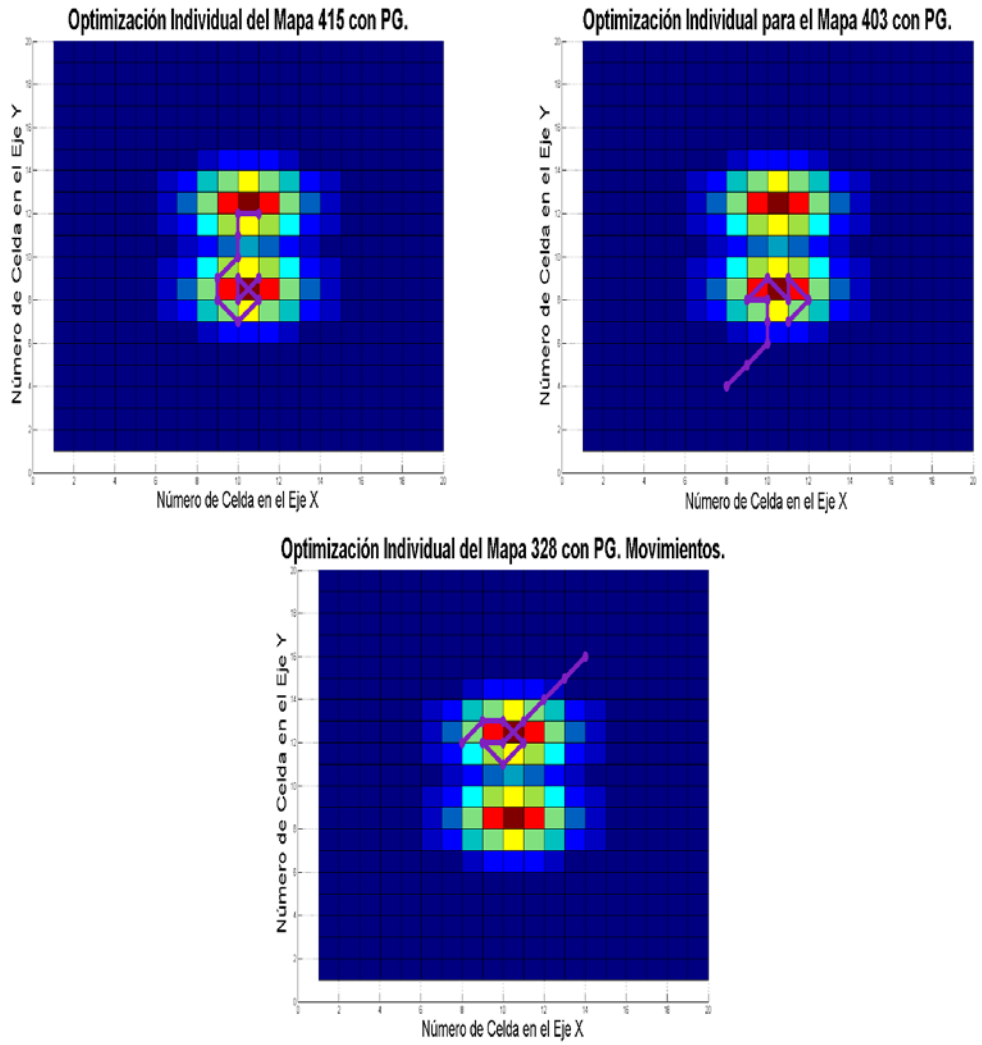


Ilustración 27. Optimizaciones Mapa a Mapa para Comparar Resultados con Los Controladores Genéricos

Los tres los casos que se producen son:

1. **Mapa 415**. PG sale del mínimo local dirigiéndose hacia otra zona donde recoger probabilidad, es decir, se aleja del mínimo local pero sin mejorar a CEO, estando la diferencia causada por los movimientos locales (el ahorro de tiempo es -5.24%).
2. **Mapa 403**. PG se queda en la zona de mínimo local obteniendo un valor de LET inferior a CEO, sobre el cual no pierde prácticamente en LET puesto que el ahorro de tiempo es de apenas un -0.05%.
3. **Mapa 328**. PG se queda en el mínimo local lo que le hace mejorar respecto a CEO (con un ahorro de tiempo de un 0.9%) Es decir, en este mapa fue mejor no abandonar la zona local.

Puede concluirse que el comportamiento de PG individual mejora respecto al genérico aunque aún persiste algún caso en el que PG queda atrapado en un mínimo local. Como se comentó en el apartado anterior, dentro del mínimo local los movimientos mejorarían si se parametrizasen los métodos en la gramática para el controlador genérico. Sin embargo, el que persista alguna condición donde hubiera supuesto una mejora abandonar la zona local hace

plantearse la alternativa de añadir en la gramática algún elemento que permita asegurar el abandono de zonas locales suficientemente exploradas.

4.3.6.3 Acercamiento a las Zonas con Máximo Valor Local de Probabilidad

En principio PG no es un método miope puesto que valora toda la probabilidad de la rejilla de 20x20 celdas, pudiendo encontrar los montículos de mayor probabilidad en la misma. Por tanto, sería esperable que se dirigiera a ellos cuando esa fuera la mejor opción, pero esto no ocurre en los casos mostrados a continuación.

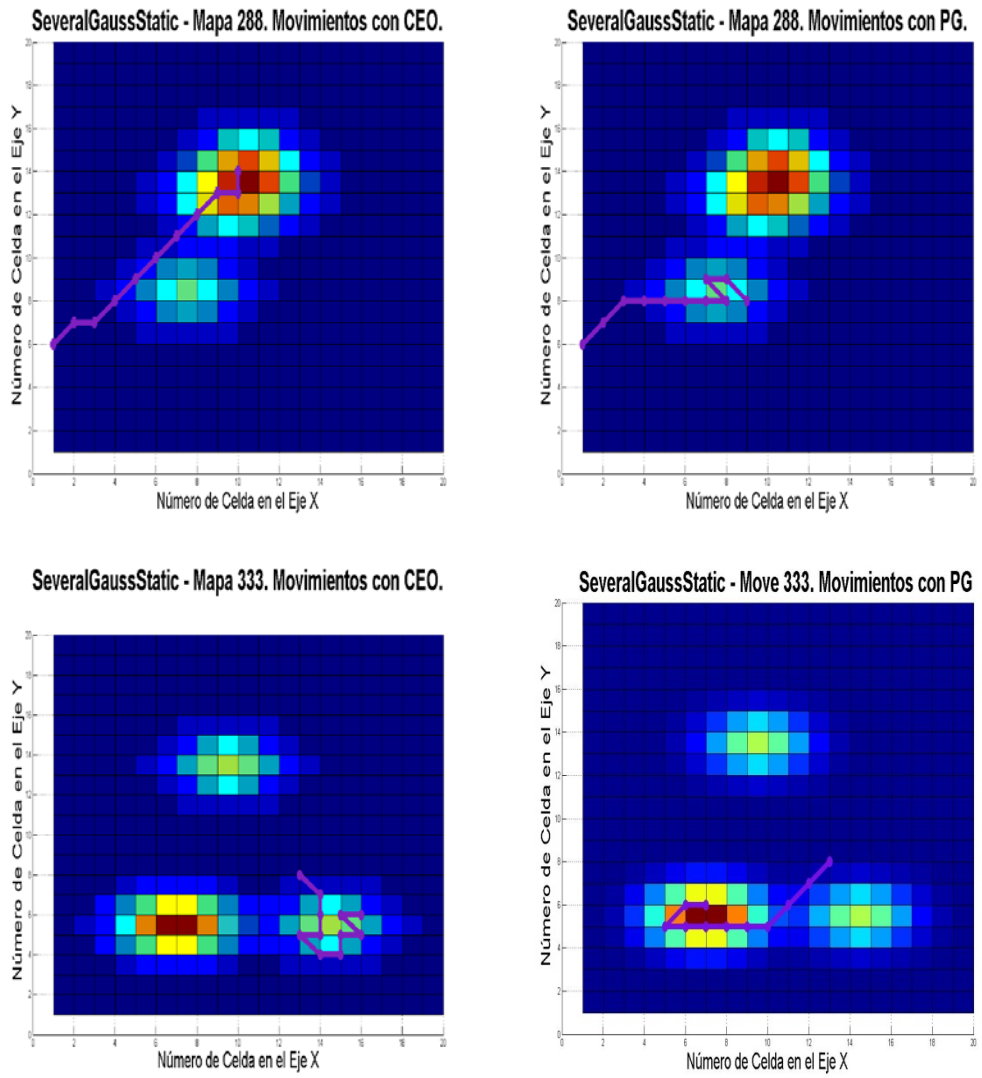


Ilustración 28. Movimientos hacia la zona de Máxima Probabilidad.

En el mapa 288 de la Ilustración 28 PG decide no ir a la zona de máxima probabilidad por encontrarse alejada y se dirige a un montículo local de probabilidad. En el mapa 333 ocurre la situación contraria, CEO se dirige a un montículo de menor probabilidad pero PG acierta al dirigirse al de mayor probabilidad. Por tanto, este mal comportamiento se observa en los resultados obtenidos con ambos métodos.

4.3.6.4 Simetrías

Existen mapas de probabilidad que presentan simetrías y, según la posición del agente, su visión del horizonte podría encontrarse con alternativas equivalentes. En la figura se muestran las decisiones tomadas por CEO y PG para un par de estas situaciones.

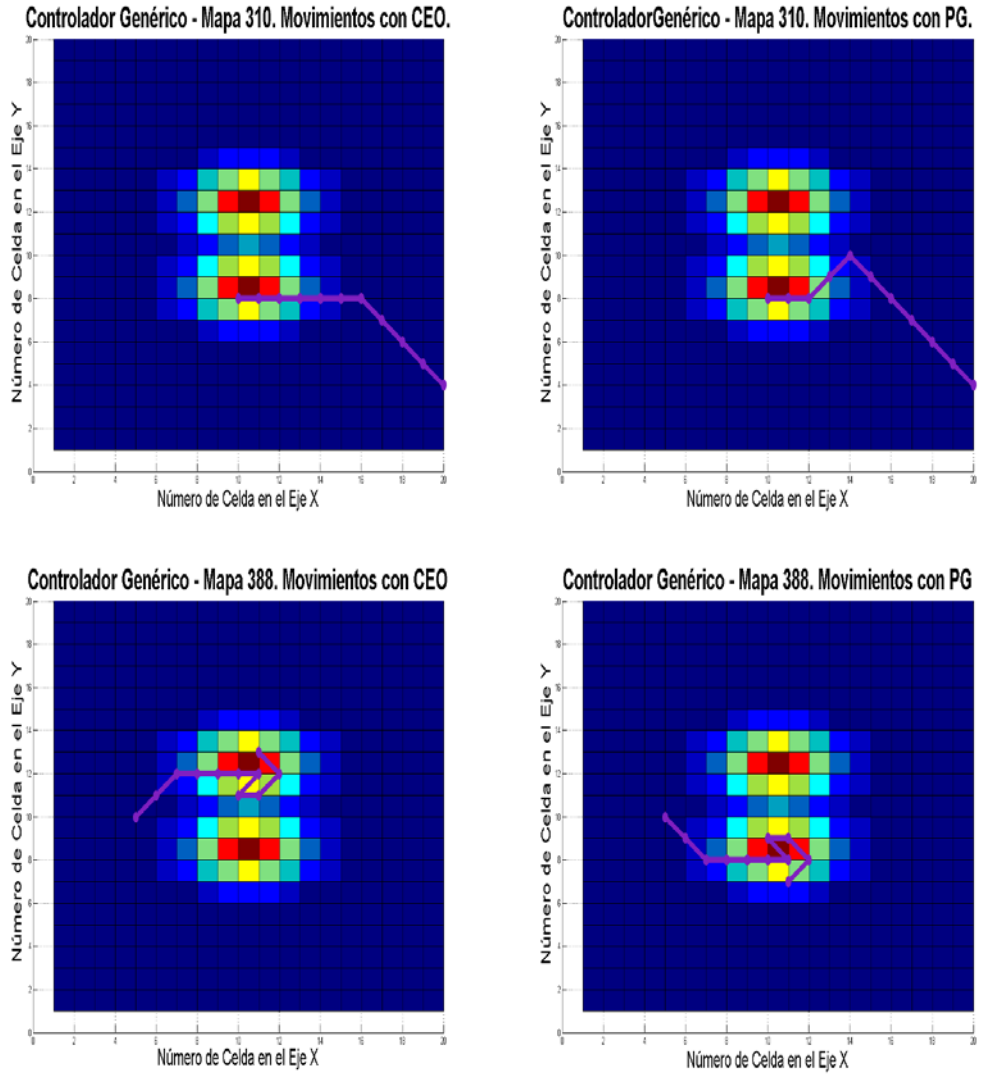


Ilustración 29. Simetrías en los Movimientos.

Como se observa en la Ilustración 29, en el mapa 310 PG resuelve la simetría con una trayectoria que parece pre-programada: se acerca a la zona donde hay concentración simétrica de probabilidad sin decidirse y, una vez cerca de ella, opta por uno de los dos montículos de probabilidad simétrica. CEO sin embargo decide en seguida ir al montículo más cercano, lo que le hace tener mejor resultado.

Este tipo de comportamientos pre-programados en los métodos (condiciones y movimientos) que intentan evitar elegir al azar entre situaciones que devuelven idénticos resultados deben evitarse, de tal modo que las valoraciones de ciertas condiciones que no puedan resolverse por sí mismas podrían depender de los resultados de otras condiciones. Para realizarlo habría de incluirse algún mecanismo en la gramática que permitiera cierta evaluación conjunta de condiciones cuando alguna condición no pueda resolverse.

En el mapa 388 existe simetría tanto en las probabilidades como en la decisión que toman CEO y PG en sus trayectorias pues son prácticamente especulares. Una u otra opción son válidas y en este caso cada método ha elegido una diferente.

4.4 Tiempos de las Optimizaciones.

En este apartado se muestran los tiempos para las optimizaciones. Los resultados son aplastantes y, mientras CEO tarda minutos en optimizar 428 mapas, PG se queda tan atrás que puede llevarle hasta 30 horas terminar el proceso con la misma cantidad de mapas.

Esto empeora para el caso de los optimizadores genéricos puesto que, con los datos utilizados para el aprendizaje durante el presente trabajo, se debe valorar la trayectoria propuesta por PG sobre un conjunto de 72 mapas o incluso de 213 mapas para un único controlador.

Lo importante es que una vez ha obtenido PF el controlador su tiempo de ejecución para cualquier mapa es inferior a 250 ms.

4.4.1 Tiempos de Optimización Mapa a Mapa

El proceso de optimización de CEO es muy estable respecto al tiempo invertido en cada mapa, es decir, no se entretiene más con unos mapas que con otros, lo que si ocurre en la optimización con PG, en la que se llegan a observar grandes diferencias entre mapas.

Para optimizar 428 mapas CEO tarda 35’ 49’’, siendo el tiempo medio de optimización por mapa de 4,96 segundos.

Para PG el tiempo medio por mapa es de 252 ms (4’ 22’). Esto se debe a que PG debe valorar 250 generaciones en el proceso evolutivo y que, según la gramática planteada, el número de nodos de cada individuo en la generación puede ser elevado, lo que le hace invertir tiempo en la obtención de cada individuo de una generación. Esto se refleja en la desviación típica que presentan los tiempos de optimización de PG, cuyo valor de 629 ms es elevadísimo, apareciendo además ciertos mapas con un tiempo de más de 10 órdenes superior al tiempo invertido en otros mapas.

No hay que olvidar que en el proceso individual se están parametrizando los métodos de la gramática buscando el mejor ajuste para cada caso, lo que amplía el espacio de búsqueda. Se verá como en la optimización genérica por tipo de mapa las desviaciones típicas disminuyen.

Se deduce que por tiempos ambos métodos no son comparables, siendo CEO considerablemente mejor.

4.4.2 Tiempos de Las Optimizaciones de los Controladores Genéricos.

En la gráfica se muestra la evolución de tiempos, en media de 20 optimizaciones, durante cada una de las 250 generaciones que se tarda en la optimización de un conjunto de mapas.

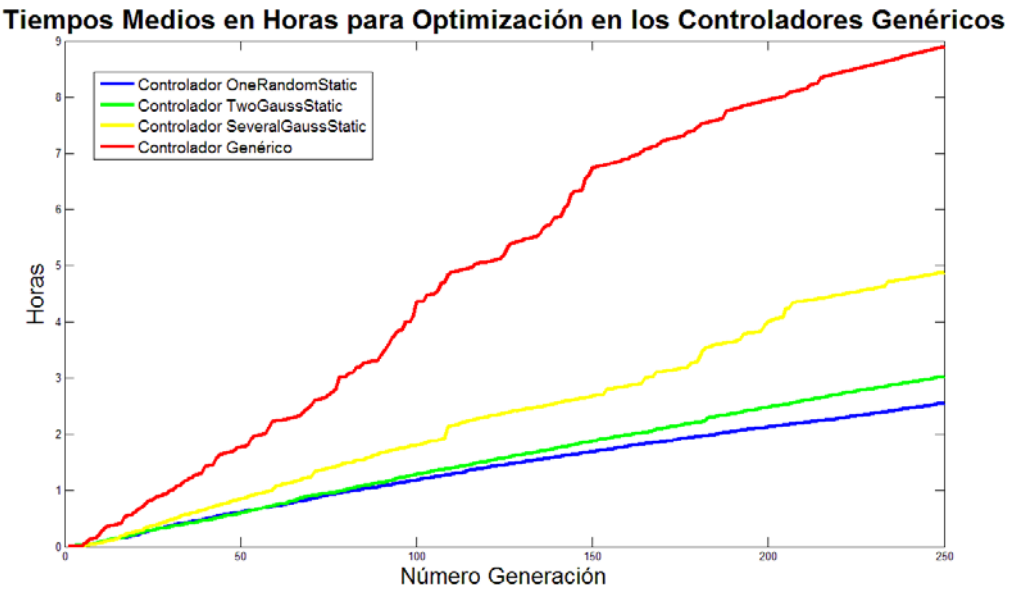


Ilustración 30. Tiempos Medios en cada una de las 250 Generaciones de los Procesos de Optimización Genérica por Tipo de Mapa

Como se observa en la Ilustración 30, para la optimización de los programas-controladores genéricos se llega a invertir un tiempo en horas. Además estos tiempos son mayores conforme aumenta la complejidad del mapa (entendida por el número de regiones que contiene) y crece el número de mapas con el que se entrena/optimiza el controlador (en el caso genérico único se utiliza el triple de mapas que en los casos genéricos por tipo de mapa). En concreto, se puede observar que:

- **OneRandomStatic** tiene un tiempo medio por generación de 37 sg.
- **TwoGaussStatic** tiene un tiempo medio por generación de 43,6 sg.
- **SeveralGaussStatic** tiene un tiempo medio por generación de 70,3 sg.
- El **Controlador Genérico** tiene un tiempo medio por de 2min 56 sg.

Es importante recordar que se han utilizado 71 mapas en el aprendizaje de los controladores por tipo y 213 en el genérico, pero no se ha realizado un estudio de qué número de mapas sería el mínimo necesario para obtener valores de ET aceptables. Es decir, se podría dar el caso de que utilizando menos mapas se obtuvieran controladores con resultados similares, lo que reduciría los tiempos del proceso.

Cuando se valoraron los tiempos de CEO se comentó que el reparto de tiempo dedicado a cada mapa estaba muy igualado. En PG existen generaciones en las que, por la estructura del individuo o programa que se está probando, se requiere más tiempo para su generación y prueba. Las diferencias del tiempo invertido entre mapas no son tan exageradas como en el caso individual, aunque la desviación típica sigue siendo elevada. En la siguiente tabla se muestran estos valores.

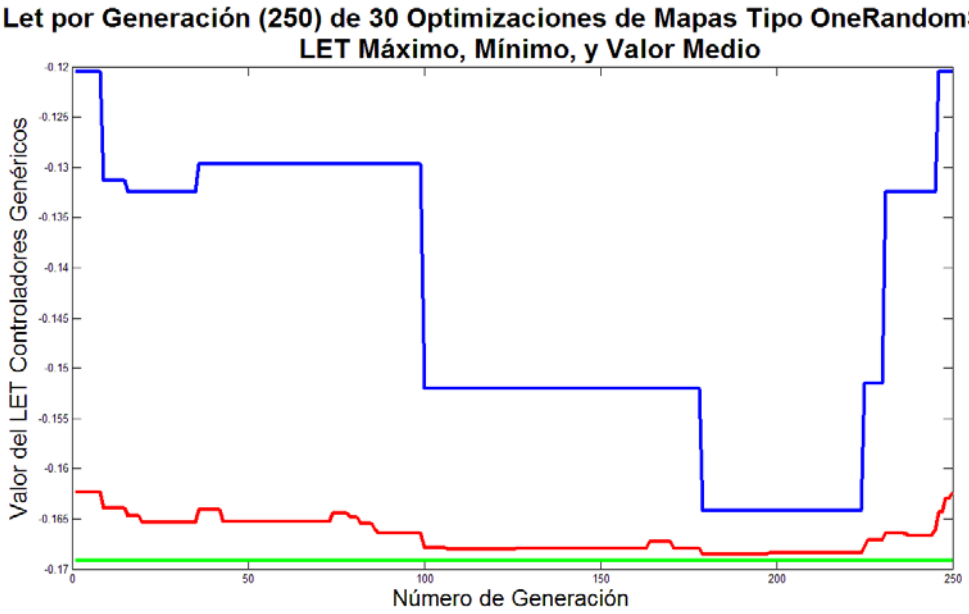
	Media tiempo por Mapa	Desviación Típica
OneRandomStatic	36''	12''
TwoGaussStatic	43''	17''
SeveralGaussStatic	1' 17''	1' 41''
Controlador Genérico	2' 13''	2' 34''

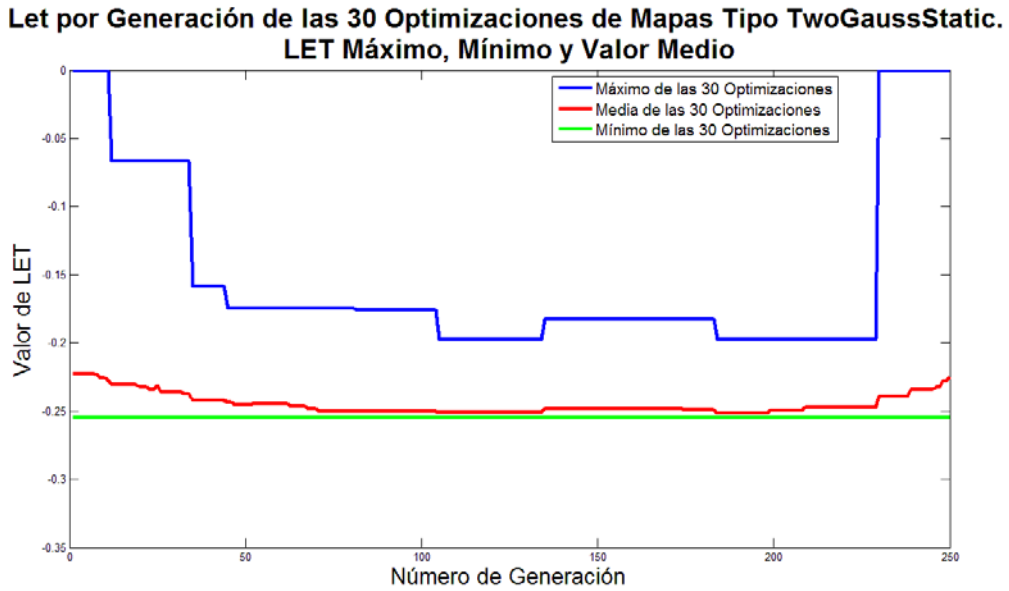
Finalmente, es importante destacar que una vez obtenido un controlador genérico, su ejecución para cualquier mapa se sitúa en una media de 241 ms.

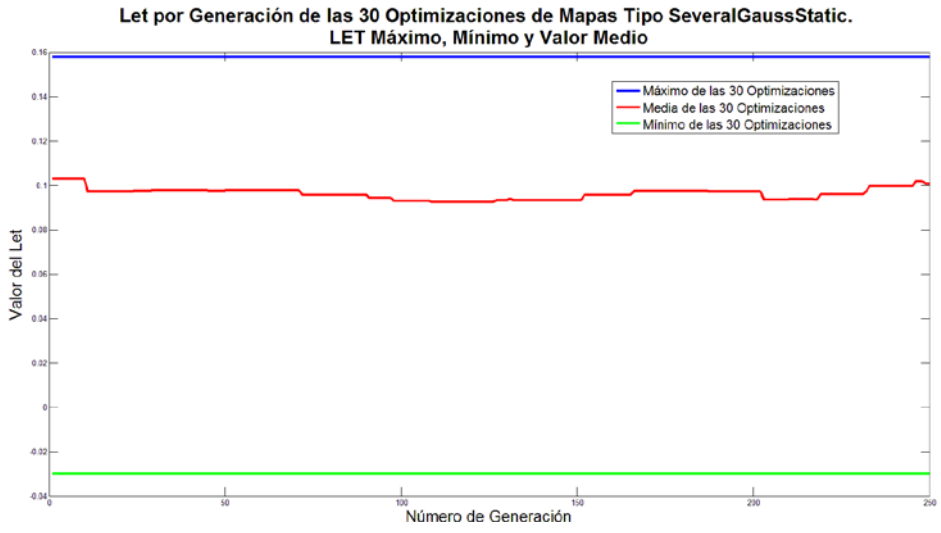
4.5 Consideraciones adicionales sobre el Proceso de Optimización

Una de las gráficas que aparece en el libro de (Koza, 1998) para representar el proceso de optimización consiste en mostrar la evolución de la aptitud en el transcurso de las generaciones. En esta gráfica se muestra, para los datos de aprendizaje, la mejor y peor evolución de la aptitud, así como la media de aptitud obtenida con los datos de aprendizaje. Otro parámetro interesante es la complejidad estructural de los programas que se van obteniendo, parámetro que se determina sumando el número de nodos del árbol generado, es decir, el número de condiciones y movimientos.

En las gráficas siguientes se muestra la evolución de la aptitud por generación. En el caso de los controladores por tipo de mapa, se muestran los valores medios, mejores y peores. En el caso del controlador genérico, únicamente se muestra el valor medio ya que no se han recogido los valores extremos durante la optimización.


Let por Generación (250) de 30 Optimizaciones de Mapas Tipo OneRandomStatic.
LET Máximo, Mínimo, y Valor Medio
Valor del LET Controladores Genéricos
Número de Generación


Let por Generación de las 30 Optimizaciones de Mapas Tipo TwoGaussStatic.
LET Máximo, Mínimo y Valor Medio
Máximo de las 30 Optimizaciones
Media de las 30 Optimizaciones
Mínimo de las 30 Optimizaciones
Valor de LET
Número de Generación


Let por Generación de las 30 Optimizaciones de Mapas Tipo SeveralGaussStatic.
LET Máximo, Mínimo y Valor Medio
Máximo de las 30 Optimizaciones
Media de las 30 Optimizaciones
Mínimo de las 30 Optimizaciones
Valor del Let
Número de Generación

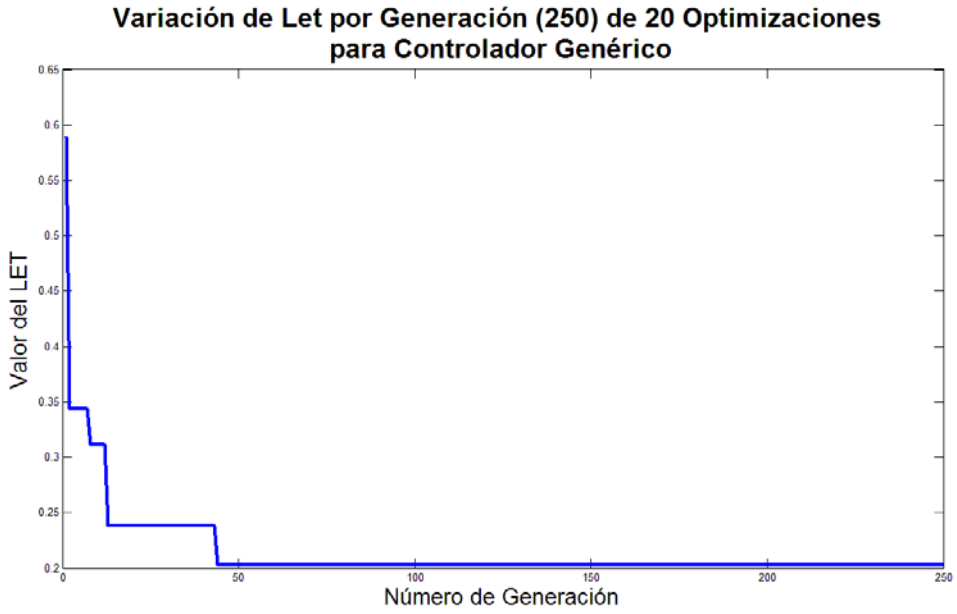
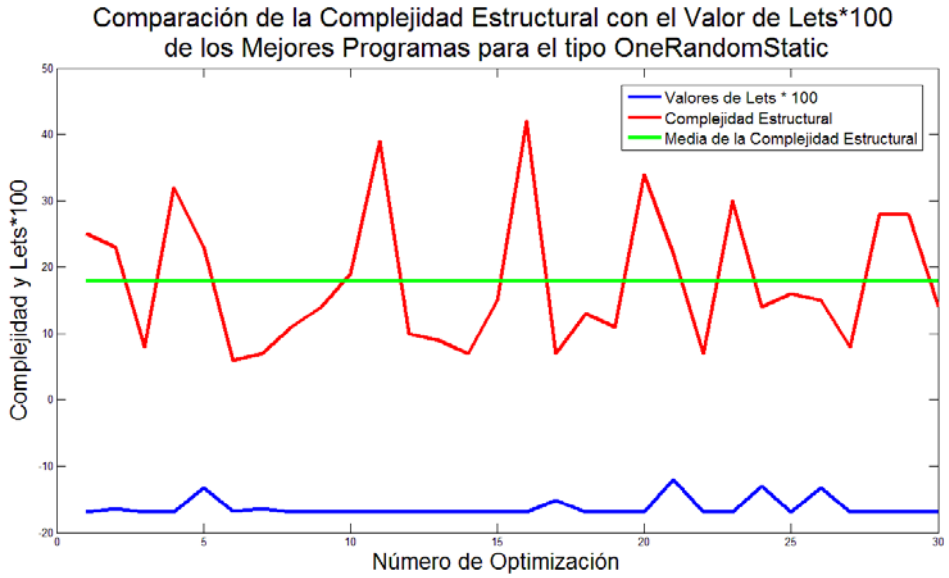
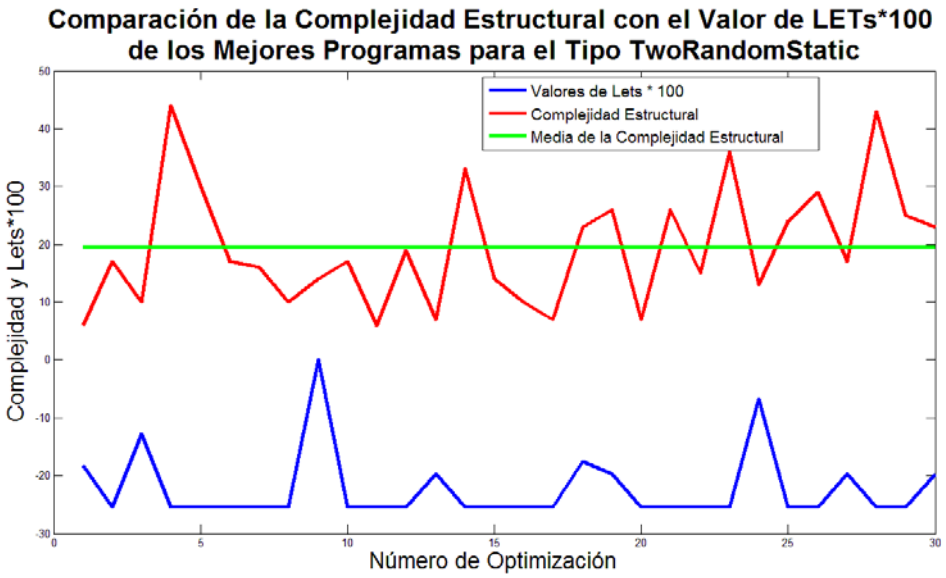


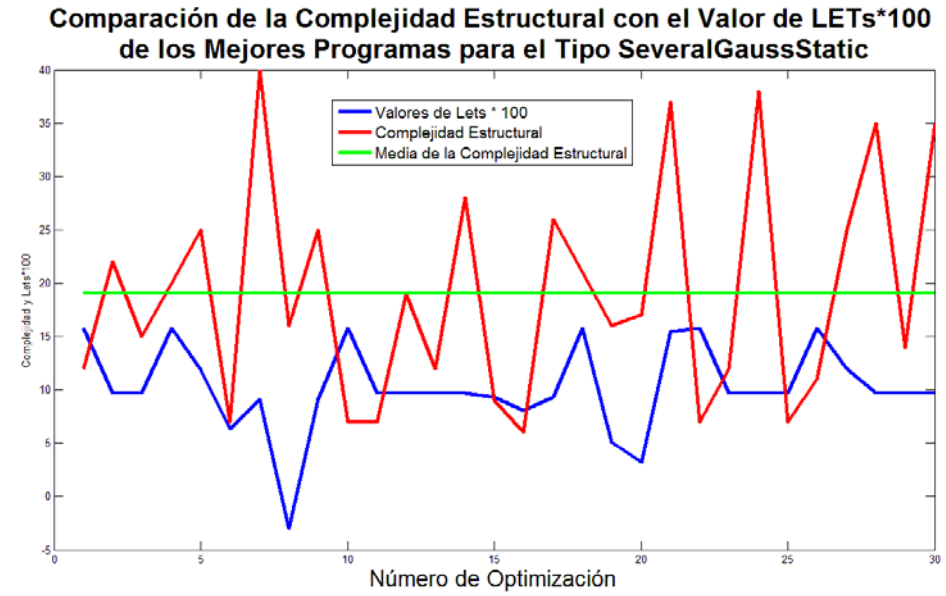
Ilustración 31. Variación del LET durante las 250 Generaciones para los Controladores Genéricos.

En todos los casos se observa que sobre la generación número 100 ya se ha encontrado una solución con el LET que será solución (50 generaciones para el controlador genérico). De este modo podemos concluir que no serían necesarias las 250 generaciones utilizadas en la prueba y que fijando este valor entre 100 y 150 los tiempos de la optimización podrían reducirse a la mitad.

Respecto de la **complejidad estructural,** entendida como el tamaño de los programas y definida como la suma de todos los nodos terminales y no terminales, en las siguientes gráficas se muestra comparada con el valor de LET multiplicado por 100 para que queden próximas visualmente.




Comparación de la Complejidad Estructural con el Valor de LETs*100
de los Mejores Programas para el Tipo TwoRandomStatic
Valores de Lets * 100
Complejidad Estructural
Media de la Complejidad Estructural
Complejidad y Lets*100
Número de Optimización


Comparación de la Complejidad Estructural con el Valor de LETs*100
de los Mejores Programas para el Tipo SeveralGaussStatic
Valores de Lets * 100
Complejidad Estructural
Media de la Complejidad Estructural
Complejidad y Lets*100
Número de Optimización

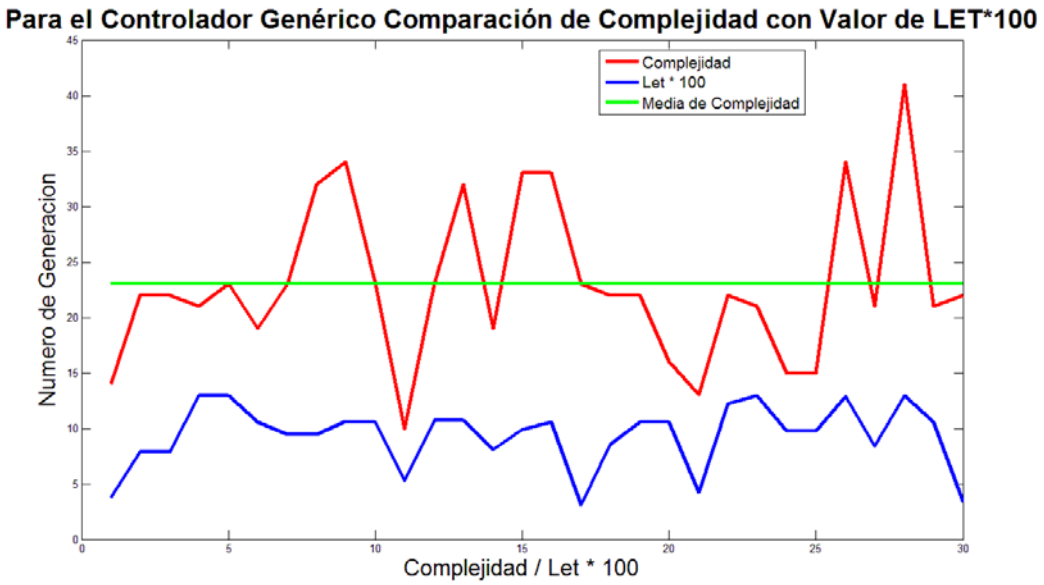


Ilustración 32. Gráficas de la Complejidad Estructural para los Controladores Genéricos.

Viendo las gráficas se observa que pueden obtenerse bajos valores de LET con una menor complejidad estructural. Es decir, los programas más complejos no siempre van a dar la mejor solución. Si se observa la evolución de las gráficas se verifica que la optimización va variando la complejidad estructural de las soluciones constantemente por lo que el valor de la complejidad podría añadirse al LET como valor de aptitud para hacer que los programas obtenidos fueran más sencillos.

Quedaría pendiente realizar un proceso de análisis de los programas con el fin de eliminar redundancias o regiones de código inalcanzables.

CAPÍTULO 5. Conclusiones y Trabajo Futuro

Se ha abordado el problema de optimización de las rutas que un agente debe seguir para minimizar el tiempo de búsqueda de un objeto desde la perspectiva de la PG, por medio de la definición de una gramática específica. Esto implica determinar los ingredientes, funciones y terminales, que formarán parte de una gramática capaz de generar los programas que controlan el desplazamiento de los agentes durante la búsqueda del objeto.

Se ha propuesto una gramática formada por ciertos tipos de movimientos que se elegirán en función de ciertas condiciones de probabilidad que puedan cumplirse desde la posición actual del agente. La misión de la PG es conjugar esas condiciones para que cada combinación de las mismas indique el tipo de movimiento a aplicar conforme a la probabilidad que rodea al agente, y al total de probabilidad restante en el enrejado de casillas.

Se ha definido una gramática inicial en forma de reglas IF-THEN-ELSE y, posteriormente, una modificación de la misma que permite el anidado de dichas reglas y que mejoró considerablemente los resultados obtenidos con la gramática anterior. Con este formato de gramática los resultados obtenidos para la optimización de mapas individuales son comparables a los obtenidos por otros métodos como CEO, hecho que es corroborado por el análisis mediante la prueba Wilcoxon realizado con dos muestras de 20 optimizaciones repetidas realizadas por ambos métodos, PG y CEO, sobre un elevado número de mapas diferentes.

A continuación, se ha utilizado la PG para obtener un controlador genérico diferente para cada tipo de mapa o un controlador genérico para todos los mapas. Para obtener estos controladores se necesita un valor de aptitud relacionado con el tiempo esperado local (LET) pero aplicable a un conjunto de mapas, definiéndose como el valor máximo de la diferencia entre el LET de PG y el LET de referencia de CEO. Los resultados obtenidos en este caso no han sido mejores que los de CEO. Esto es debido a que CEO calcula la mejor trayectoria para cada mapa y, por lo tanto, se centra en obtener las acciones más relevantes para un mapa de probabilidades particular sin generalizar. Los 20 controladores genéricos obtenidos por PG en 20 optimizaciones repetidas, para cada tipo de mapa, permiten calcular rápidamente 20 caminos adecuados para el problema de búsqueda en tiempo mínimo sobre un mapa de probabilidades especifico y escoger el que tiene mejor LET. De este modo se ha comprobado que los programas generados por PG permiten obtener trayectorias aceptables en un porcentaje superior al 80% de los casos mapas utilizados como elementos de prueba.

Se ha comprobado que, en los controladores genéricos desaparecían ciertos tipos de movimientos que contenían valores parametrizados en su definición, siendo el proceso de optimización el encargado de obtener dichos valores seleccionándolos de un rango. Estos movimientos si aparecen en la optimización individual mapa a mapa. Si esto es así, se puede deducir que las condiciones de probabilidad propuestas en la gramática trabajan conjuntamente con los atributos de estos movimientos para encontrar la solución óptima y que el proceso de generalización no puede resolver los valores en algunos movimientos que desaparecen.

Para conseguir mejorar la capacidad de generalización de PG debería ampliarse la gramática de modo que, además de plantear nuevas condiciones para abarcar más casos, se produzcan los valores de los parámetros de los movimientos a partir de los datos de probabilidad. En la nueva gramática se pueden establecer funciones adicionales que realicen estos cálculos sobre el mapa de probabilidad y pasen los resultados de los mismos a los movimientos parametrizados, los cuales podrán entonces adaptar su comportamiento a cada mapa . Es de esperar que una generalización parametrizada mejore los movimientos locales producidos por los controladores genéricos.

Se han analizado algunas trayectorias obtenidas por PG comparándolas con las presentadas por CEO. En algunas de ellas se ha visto como haría falta alguna condición para que PG seleccione mejor las zonas de probabilidad a la que dirigirse en ciertos casos, así como para evitar comportamientos pre-programados en la codificación de movimientos y condiciones, sobre todo cuando se dan alternativas equivalentes y se intenta evitar moverse al azar forzando un comportamiento.

Una tarea que hay que realizar en el futuro será incorporar un analizador de código capaz de simplificar la estructura de los programas obtenidos por PG. Además, un analizador de este tipo podría utilizarse para extraer conclusiones que permitiesen incluir nuevos elementos en la gramática propuesta, y mejorar los programas-controles genéricos obtenidos.

Los tiempos de optimización de PG son muy elevados pero, viendo cómo ha evolucionado la aptitud en las diferentes generaciones, se comprueba que el LET óptimo puede obtenerse con menos generaciones. De este modo sería posible reducir a la mitad el número de generaciones en el proceso de optimización, lo que reduciría el tiempo para obtener el controlador. De todos modos, este aspecto no es tan importante puesto que el objetivo es tener un código del controlador ejecutable para todo mapa, no importando tanto el tiempo necesario para obtener el programa como el tiempo necesario para ejecutarlo, que es muy reducido.

Finalmente, es importante destacar que los resultados para los controladores genéricos son aceptables en un número de casos superior al 80% respecto a CEO, de modo tal que PG podría convertirse en un método eficaz si se resuelven las deficiencias comentadas anteriormente.

Bibliografía

Adaptive And Bioinspired Systems Research Group. (n.d.). Retrieved 2014, from http://absys.dacya.ucm.es/doku.php: http://absys.dacya.ucm.es/doku.php?id=software

Beckhusen, R. (2013). *http://www.defensenews.com/.* Retrieved 08 23, 2014, from http://www.defensenews.com/article/20140325/C4ISRNET08/303250022/Applying-math-sank-U-boats-today-s-intel-problems

Beckhusen, R. (2013, Julio 5). *http://www.defensenews.com/.* Retrieved 08 23, 2014, from http://www.defensenews.com/article/20140325/C4ISRNET08/303250022/Applying-math-sank-U-boats-today-s-intel-problems

Hajira Jabeen, A. R. (2010). Review of Classification Using Genetic. *International Journal of Engineering Science and Technology* (pp. 94-103). Islamabad, Pakistan: National University of Computer and Emerging Sciences.

Hitoshi Iba, Topon Kumar Paul, Yoshihiko Hasegawa. (2010). *Applied Genetic Programming and Machine Learning.* United States Of America: CRC Press International Series on Computational Intelligence.

Knowles, J. D. (2006). A tutorial on the performance assessment of stochastic multiobjective optimizers, technical report.TIK-Report No. 214. Computer Engineering and Networks Laboratory, ETH Zurich, Switzerland.

Koza, J. R. (1998). *Genetic Programming. On the Programming of Computers by Means of Natural Selection.* MIT Press.

Kratzke T.M., S. L. (2010). Search and Rescue Optimal Planning System. *13th Conference on Information Fusion*, (pp. 1 - 8). Edinburgh, Scotland.

Kratzke, T. S. (2010). Search and Rescue Optimal. *13th Conference on Information Fusion.* Edinburgh, Scotland: IEEE.

Lanillos Pradas, P. (2013). Búsqueda de Objetivos Móviles en Tiempo Mínimo sobre entornos con incertidumbre. Tesis Doctoral, Universidad Complutense de Madrid, España.

Lawerence D. Stone, C. M. (2011). *Search Analysis for the Location of the 447 Underwater Wreckage.* Metron Scientific Solutions.

Lawerence D. Stone, C. M. (2011). *Search Analysis for the Location of the 447 Underwater Wreckage.* Reston, Virgina: Metron Scientific Solutions.

Lawerence D. Stone, C. M. (2011). Search Analysis for the Underwater Wreckage of Air France Flight 447. *14th International Conference on Information Fusion*, (pp. 1061, 1068). Chicago, Illinois, USA.

Manso Vergara, J. (2013). Búsqueda de Objetivos estáticos en tiempo mínimo mediante redes neuronales. Trabajo Fin de Máster, Universidad Complutense de Madrid, España.

Marconi de Arruda Pereira, C. A. (2010). A Niched Genetic Programming Algorithm for Classification Rules Discovery in Geographic. *SEAL*, 260-269.

Murphy, K. P. (2012). *Machine Learning: A Probabilistic Perspective.* MIT Press.

Pedro G. Espejo, S. V. (2008). A Survey on the Application of Genetic. (pp. 121-144). Granada: IEEE Transactions on Systems, Man and Cybernetics - Part C: Applications and Reviews, Vol 40, no 2.

Pedro G. Espejo, S. V. (2008). *A Survey on the Application of Genetic.* Granada: IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS—PART C: APPLICATIONS AND REVIEWS, VOL. 40, NO. 2.

Peter C. Bell, R. I. (1998). *COLUMBUS-AMERICA DISCOVERY GROUP and the SS CENTRAL AMERICA.* Oxford: Institute for Operations Research and Management Sciences.

Riccardo Poli, W. B. (2007). *Genetic programming an introductory tutorial and a survey of techniques and applications.* University of Essex.

Riccardo Poli, W. B. (2007). *Genetic programming an introductory tutorial and a survey of techniques and applications.* University of Essex: Technical Report.

Riccardo Poli, W. B. (2008). *A field guide to genetic programming.* http://lulu.com and freely available at http://www.gp-field-guide.org.uk.

Weise, T. (2011). *Global Optimization Algorithms. Theory and Application.* Kassel: http://www.it-weise.de/projects/book.pdf.

Whigham, P. (1995). *Grammatically-Based Genetic Programming.* California, USA: Proceedings of the Workshop on Genetic Programming: From Theory to Real World Applications.

Zuluaga, F. J. (2014). Técnicas heurísticas para la búsqueda en tiempo mínimo (Comparativa). Trabajo Fin de Máster, Universidad Complutense de Madrid, España.