Preguntas del Oponente

Pregunta 1

A pesar de que el algoritmo genético lograra un valor lo suficientemente pequeño de la función objetivo en menos iteraciones, ¿es aconsejable su uso a pesar de haber reducido el conjunto inicial a 41 casos, a diferencia de GRASP donde quedaron 163 casos?

Respuesta 1

Los escenarios de aplicación de la propuesta de la tesis son muy diversos. La elección de un método metaheurístico en particular depende en gran medida del contexto del problema. Como se expone en la memoria escrita, la tipología de los casos de prueba se reduce a un conjunto pequeño (cuatro vectores binarios de dos componentes). Esto puede provocar que una gran cantidad de casos de prueba no siempre sea deseable. Utilizar un conjunto grande (de tamaño 161 en este caso), donde haya exceso de representación por cada tipo de caso, podría traer resultados no deseados como una de las observaciones que se hacen en la aplicación del algoritmo genético con las soluciones de los estudiantes en el examen de Anagramas: el aumento del porciento de efectividad de cada solución. Sucede que, al haber más casos, es probable que en ese conjunto final haya muchos que sean resueltos de manera satisfactoria por alguna solución, la cual, sin poder resolver todavía algunos casos, se ve beneficiada en el sentido del porciento. Habría que entonces hacer un mejor ajuste de los parámetros (lo esperado para cada nota) ante este tipo de situación.

Las bondades del algoritmo genético [1] permiten que sea usualmente empleado por los investigadores en problemas de optimización combinatoria. Una ventaja clara es que operan con varias soluciones de forma simultánea, en vez de ser secuenciales. La variedad de operadores genéticos que se planteen evita que queden atrapados en óptimos locales y puedan explorar hacia uno global. Sin embargo, nada es absoluto y en dependencia del contexto podría ser más recomendable el uso del otro algoritmo como Búsqueda Tabú o Recocido Simulado, ya que puede darse el caso que se demore en converger o no converger debido a los parámetros y características de la población.

Independientemente de cuál sea el algoritmo empleado y el tamaño de la solución final y, deberían existir cantidades mínimas de cada tipo de solución para garantizar representatividad [2]. Pero esto es algo que no resultado sencillo con casos de prueba reales, pues en la práctica los de tipo $(1, 0)$ donde solo acierte la implementación de 3 y no así la de 4, no abundan.

Pregunta 2

Si fuera a resolver el problema de la generación automática de casos, ¿qué consideraciones habría que tener? ¿Qué se debería garantizar para un correcto funcionamiento de la propuesta? Exponga brevemente cómo atacaría el problema.

Respuesta 2

Un aspecto importante que se debería tener en cuenta es lograr la representación del universo de casos [3]. Para el correcto funcionamiento de cualquier algoritmo a aplicar, es necesario que reciba como entrada un conjunto representativo de casos de prueba de ese universo. Además, sería idóneo que cada tipo de caso esté representado con varios representantes, tratar de llegar a un equilibrio entre el exceso y el defecto.

Algunas fuentes [4] llevan a cabo una etapa denominada Testeo Temprano, donde en el que se intenta maximizar la cobertura de escenarios con un reducido menor número de valores. Para lo cual se generan combinaciones para cada escenario con la intención de probar una funcionalidad.

Para resolver este problema sería útil diseñar una clasificación para cada caso e intentaría generar una cantidad suficiente de cada tipo como solución final. Un tipo de clasificación podría ser los 4 grupos denominados “tipos” de casos de prueba, observación descrita en el Experimento 1 durante la ejecución de GRASP. Esto depende también de la representación del problema, si los vectores de información fuesen continuos en el intervalo [0, 1] se podrían asignar mejores etiquetas a cada caso de prueba al tener más información por la que clasificar. La generación empleada en la propuesta fue un tanto similar, se confeccionaron se forma manual algunas “semillas” (dígase casos representativos o canónicos) y a partir de estas, surgieron combinaciones de casos utilizando algunos operadores aleatorios en combinación con palabras extraídas de documentos en inglés. Por ejemplo, en Anagramas se tuvo un parámetro de “cantidad de letras consecutivas repetidas” para garantizar la variedad que es difícil de encontrar en palabras comunes que no tienen más de 2 letras iguales seguidas. Esto le proporcionó un mejor rendimiento al algoritmo.

Pregunta 3

Los experimentos realizados fueron sobre el problema ”Suma de polinomios”, donde comprobar si la solución es correcta para un caso se realiza en tiempo polinomial. ¿Es factible la utilización de esta solución en problemas donde comprobar la correctitud de una solución tenga costo exponencial?

Respuesta 3

El comprobar si una solución es correcta se realiza un número constante de veces, por lo cual no se ve afectada la ejecución de la solución propuesta, en cambio compromete las procesamientos previos y posteriores.

Una vez recibido el conjunto de casos de prueba generado, es necesario el proceso de construcción de los vectores de información, para ello se visita solo una vez cada caso por cada implementación provista (procesamiento previo). Durante la ejecución del algoritmo de reducción, no se vuelve a comprobar la correctitud de una solución. Sería luego de seleccionado el conjunto final de casos, que con las soluciones de los estudiantes se vuelve a hacer necesario el saber si la salida de dichas soluciones es correcta, pero en este caso sería una vez por cada caso y solución.

Pregunta 4

En los resultados obtenidos por GRASP se observa que el porcentaje deseado para la nota de 3 es de 30 %, en la práctica no es un valor real deseado. ¿Qué sucede si se espera un valor en el rango [60,70]?

Respuesta 4

Este ajuste fue experimental por las características del problema multiplicación de polinomios. La siguiente Tabla 1 muestra cuáles fueron los porcientos iniciales y finales en el problema multiplicación de polinomios.

Porcentaje inicial/esperado del 3	19.60/30
Porcentaje final/esperado del 3	30.06/30

Cuadro 1: Tabla de los ajustes realizados

Es preciso conocer cuáles fueron las soluciones usadas como nota de 3 y como nota de 4.

- Nota de 4. Solo falla en un tipo de casos: en el que uno de los dos sea el polinomio 0 y el otro un polinomio de grado mayor que 1.
- Nota de 3. Esta solución obvia lo anteriormente acumulado del mismo grado y lo reescribe, por lo cual falla en casi todos los casos, excepto en aquellos polinomios en que sean excluyentes los grados entre sí.

Entonces, dado que los casos que acierta el 3 son muy pocos si se recibe una muestra representativa de entrada, es necesario definir un bajo porciento de acierto para que los algoritmos tengan un bajo margen de error. Lo que pasaría si se define un rango entre [60, 70] para la nota de 3, está en

dependencia de la entrada. Si es lo suficientemente grande se espera que los algoritmos logren reducir de manera aceptable la función objetivo, de lo contrario lanzarán márgenes de error muy grandes como sucedió en las experimentaciones iniciales donde se definían porcientos similares.

Estos porcentajes actualmente se reciben como parámetros de entrada y no son valores fijos. Sin embargo, pueden ser aprendidos, como en el anterior ejemplo.

Preguntas del Tribunal

Pregunta 1

Como se muestra en los recuadros rojos en las imágenes mostradas a continuación de este párrafo, para el algoritmo genético existen momentos de una mejora rápida y pronunciada en la evaluación de la función objetivo. Muestre el contexto de la ejecución del algoritmo en esos momentos y haga un análisis de qué rasgos del algoritmo genético utilizado están detrás de esas rápidas mejoras.

Respuesta 1

Las particularidades del contexto y los operadores genéticos aplicados son claves en el comportamiento que describe el algoritmo.

- operador $op_1(i_1, i_2)$: mezcla la primera mitad de i_1 con la segunda mitad de i_2.
- operador $op_2(i_1, i_2)$: mezcla de forma alternada i_1 e i_2.
- operador $op_3(i_1, i_2)$: unión de miembros aleatorios de i_1 e i_2.
- operador $op_4(i_1, i_2, p)$: mezcla el $p\,\%$ de i_1 con el $(1-p)\,\%$ de i_2.

Es notable una alta variabilidad en los operadores genéticos, tal indicador favorece la exploración del espacio de búsqueda.

- Se seleccionan los 2 mejores individuos
- Se seleccionan 2 individuos al azar

Esta estrategia de selección puede decirse que permite la exploración más allá de la brindada a partir de los 2 mejores individuos, lo cual surgen nuevos óptimos. Por último, los 4 operadores genéticos se les aplican a ambas parejas de candidatos, lo cual potencia aún más la exploración, dando como resultado 8 nuevos individuos. Por otro lado, al ser problemas relativamente sencillos los aplicados de examen 1 de las pruebas de programación, el espacio

de búsqueda de estos resulta pequeño en cuanto a tipos de casos. El de multiplicación de polinomios es aún más reducido que el de anagramas, por eso (y dados los altos índices de exploración del algoritmo genético) es que observa en la Figura (1) un brusco descenso porque se explora rápidamente el espacio, mientras que en la Figura (2) se aprecian también pero más espaciados y menos bruscos. En el lado de GRASP, es casi determinista si consideramos la evaluación de la f.o. y la cantidad de casos de cada tipo en cada ejecución. Elimina de manera constante un caso desde el inicio hasta que se estanca y no mejora su solución. El proceso varía de una ejecución a otra y la información relacionada con el empleo de los operadores no se guarda como resultado de la ejecución y, por tanto, no es posible determinar explícitamente cuáles operadores o individuos provocaron la mejora. Sin embargo, los saltos son comunes en todas las ejecuciones y son producto del uso de operadores.

Notas Añadidas

Nota 1

En el título del trabajo aparece el término aprendizaje, pero no se evidencia una etapa de entrenamiento o que aproveche las experiencias pasadas. ¿Acaso se entiende por aprendizaje a las decisiones que toman las heurísticas al recorrer el espacio o se refiere a que se utilizaron datos reales como base para generar los casos de prueba?

Respuesta 1

Según la Inteligencia Artificial, un algoritmo de aprendizaje es un fragmento de código que permite explorar y analizar conjuntos de datos complejos y a buscar significado en ellos. Usan parámetros basados en los datos de entrenamiento, un subconjunto de datos que representa el conjunto más grande. A medida que aumentan los datos de entrenamiento para representar el mundo de una forma más realista, el algoritmo calcula resultados más precisos [5]

El término aprendizaje hace referencia al proceso que tiene que ver con la generación de casos de prueba que es dependiente del problema. Un ejemplo de ello fue el ajuste de 30 % esperado por la nota 3 para un mejor funcionamiento de las metaheurísticas. También alude al uso de casos reales, como los usados en Anagramas, para una mejor generación. Se prueba con un conjunto inicial de casos generados, el cual se va modificando hasta estar bien representado el universo de los mismos.

Nota 2

¿Por qué la necesidad de explorar el espacio de esta forma? ¿Se puede resolver el problema de manera determinista mediante reducciones del conjunto inicial intentando ajustar el porcentaje deseado? ¿Sucedería igual si se tiene más de una implementación correcta por calificación?

Respuesta 2

GRASP resuelve de manera determinista el problema mediante reducciones del conjunto inicial intentando ajustar el porcentaje deseado. Determinista en cuanto a valor final de la función objetivo y "tipos" de casos de prueba resultantes en cada conjunto solución. Esto fue apreciado en varias corridas sobre el mismo conjunto de datos. Dicho comportamiento se debe a las características de los casos de prueba (poca variedad) y al pequeño tamaño de los vectores de información.

La necesidad de explorar el espacio de manera diferente surge para lograr mejores resultados, es ahí donde entra el algoritmo genético con un alto índice de exploración. El método de exploración ha de ser proporcionada con heurísticas, pues el espacio de búsqueda es infinito.

Al tener más de una implementación por nota, se espera (con la debida contemplación en el código) obtener mejores resultados, pues se tendría más información de cada caso de prueba. Solo se cuenta con si acertó o no. De esta forma no solo se tendría en cuenta la cantidad de casos de cada tipo, además se tendría una ponderación dentro de cada categoría. En lugar de un vector binario de longitud 2, se estaría trabajando con un valor continuo en el intervalo [0, 1] para cada implementación. Esto permite una mejor diferenciación entre casos y se podrían lograr mejores conjuntos finales, incluso en menos iteraciones.

Referencias

[1] J. A. Ambuludi Olmedo, *Aplicación de Algoritmos Genéticos para la optimización de problemas combinatorios.* PhD thesis, 2018.

[2] L. B. Cicerchia, L. M. Esnaola, J. P. Tessore, H. D. Ramón, C. C. Russo, and C. A. Martínez, "Problemas de optimización combinatoria: una propuesta que combina algoritmos genéticos y metaheurísticas," 2017.

[3] J. Rodriguez and G. Rodriguez, "Generación automática de casos de prueba para test de una gui usando colonia de hormigas y metaheurística golosa," 2016.

[4] A. Macías-Rojas, M. D. Delgado-Dapena, J. Fajardo-Calderín, and D. Larrosa-Uribazo, "Generador de valores de casos de prueba funcionales," *Lámpsakos*, no. 15, pp. 51–58, 2016.

[5] “Algoritmos de aprendizaje automático.” urlhttps://azure.microsoft.com/es-es/overview/machine-learning-algorithms.