Prueba de Evaluación Continua 1

Juan José Rodríguez Aldavero
Optimización Metaheurística

Resumen

Este documento, correspondiente a la primera Prueba de Evaluación Continua de la asignatura Optimización Metaheurística, detallará los algoritmos

1. Introducción

La **optimización metaheurística** es un caso particular dentro del campo de la optimización estocástica, la cual es una rama de las matemáticas que se ocupa de resolver problemas de optimización por medio de algoritmos fundamentados en la aleatoriedad. En particular, la optimización metaheurística permite encontrar soluciones *lo suficientemente buenas* para una colección extremadamente amplia de problemas, para los cuales no se conocen soluciones óptimas ni algoritmos para encontrarlas en un número finito de pasos.

Un problema puede ser tratado mediante algoritmos de optimización metaheurística siempre y cuando se disponga de una **función de calidad (fitness)** que permita evaluar posibles soluciones test. No es necesario conocer los detalles de esta función, y la mayor parte de las veces se desconocerá su forma.

2. Gradient-based Optimization

La **optimización basada en el gradiente** consiste en mejorar de forma iterativa soluciones test mediante su desplazamiento en el espacio de soluciones en la dirección de su derivada / gradiente. El problema principal de este algoritmo es que puede converger en mínimos locales y no en el mínimo global.

Podemos ilustrar este algoritmo en su forma general para un espacio de soluciones n-dimensional y con múltiples comienzos (para así tener más posibilidades de encontrar el mínimo global) con el siguiente algoritmo:

Algorithm 3 *Gradient Ascent with Restarts*

$\vec{x} \leftarrow$ random initial value
$\vec{x}^* \leftarrow \vec{x}$ ▷ $\vec{x}^*$ will hold our best discovery so far
repeat
  repeat
    $\vec{x} \leftarrow \vec{x} + \alpha \nabla f(\vec{x})$ ▷ In one dimension: $x \leftarrow x + \alpha f'(x)$
  until $||\nabla f(\vec{x})|| = 0$ ▷ In one dimension: **until** $f'(x) = 0$
  if $f(\vec{x}) > f(\vec{x}^*)$ **then** ▷ Found a new best result!
    $\vec{x}^* \leftarrow \vec{x}$
  $\vec{x} \leftarrow$ random value
until we have run out of time
return $\vec{x}^*$

Este algoritmo se emplea frecuentemente en problemas de Machine Learning para, por ejemplo, optimizar la función de error de una red neuronal (mediante el mecanismo conocico como *backpropagation*). Vemos como para poder aplicar este algoritmo es necesario conocer la forma de la función de fitness $f(\vec{x})$ lo cual la mayor parte del tiempo no es posible.

3. Single-State Methods

La mayor parte de las veces que recurrimos a algoritmos de metaheurística no es posible calcular el gradiente de la función de calidad $f(\vec{x})$. Es más, la mayor parte del tiempo no se sabe cual es $f(\vec{x})$ ni la forma que tiene. Solo se puede emplear para evaluar soluciones test $\vec{x}_i$, funciona como una caja negra. Por este motivo, el procedimiento general para un algoritmo metaheurístico es:

- Crear un conjunto de soluciones candidatas $\vec{x}_i$.
- Evaluar su calidad mediante $f()$.
- Copiar las soluciones candidatas y hacer una pequeña modificación (*tweak*) para crear soluciones candidatas ligeramente diferentes.

3.1. Hill-Climbing

En este algoritmo, empleamos el procedimiento anterior y, simplemente, nos quedamos con las funciones que mejoren en calidad hasta alcanzar el máximo local más cercano:

Algorithm 4 *Hill-Climbing*

```
1: S ← some initial candidate solution                    ▷ The Initialization Procedure
2: repeat
3:     R ← Tweak(Copy(S))                                 ▷ The Modification Procedure
4:     if Quality(R) > Quality(S) then                    ▷ The Assessment and Selection Procedures
5:         S ← R
6: until S is the ideal solution or we have run out of time
7: return S
```

Observar como es una **aproximación estocástica** al algoritmo basado en el gradiente, en el sentido de que la operación *tweak* es aleatoria y en ocasiones producirá soluciones mejores o peores.

El significado de la operación *tweak* depende de la representación del espacio de soluciones que estemos utilizando. Por ejemplo, si las soluciones $\vec{x}_i$ son vectores de variable real, podemos definir *tweak* mediante una convolución uniforme (añadir ruido a cada una de las componentes del vector en función de una probabilidad).

Normalmente podremos ajustar el nivel de **exploración frente a explotación** del algoritmo, en función de unos parámetros α_i.

3.2. Random search

Un ejemplo de algoritmo con un grado extremo de exploración frente a explotación es el **algoritmo de búsqueda aleatoria** que, como indica el nombre, busca soluciones de forma aleatoria en el espacio de soluciones y se queda con la mejor.

Podemos combinar las ideas del algoritmo random search con el resto de algoritmos incorporando n comienzos aleatorios. Esto aumentará el grado de exploración.

3.3. Simmulated Annealing

Los algoritmos de Simmulated Annealing pretenden emular conceptos de física estadística introduciendo una distribución de probabilidad:

$$P(t, R, S) = e^{\frac{Quality(R) - Quality(S)}{t}}$$

donde t es un parámetro llamado temperatura, S es la solución original y R es la solución modificada. El algoritmo es similar al de Hill-Climbing anterior, con la diferencia de que permite que soluciones R peores que la anterior se acepten con probabilidad P. Normalmente la temperatura es un parámetro que controla el nivel de explotación y se va disminuyendo con el tiempo. Cuanto mayor es t, mayor es la probabilidad de aceptar malas soluciones, y viceversa. En el límite $t \to \infty$ el algoritmo converge a random search, mientras que en el límite $t \to 0$ converge a hill-climbing.

4. Population Methods

Mientra que los algoritmos single-state mejoran de forma iterativa soluciones $f(\vec{x})$ de manera individual, los algoritmos de población iteran sobre un conjunto de soluciones que a su vez interactúan entre sí. El ejemplo más importante de estos algoritmos viene dado por las **estrategias evolutivas**, las cuales se inspiran en la naturaleza; y los algoritmos genéticos, un subtipo de estrategia evolutiva que incorpora ideas como mutación y recombinación de información.

4.1. Estrategias evolutivas

Las estrategias evolutivas son una primera aproximación a los algoritmos genéticos en las cuales, dentro de una población de ν individuos (soluciones test), sobreviven los μ más adaptados al medio (minimizan la función de calidad dentro del espacio de soluciones). Después, estos $\nu - \mu$ individuos darán lugar a una descendencia de ν nuevos individuos por medio de una mutación (antiguo *tweak*). Normalmente cada uno de los μ progenitores tendrá $\frac{\nu}{\mu}$ descendientes.

Existen dos formas de plantear un algoritmo evolutivo:

- Algoritmo (μ, ν): $\nu - \mu$ padres dan lugar a ν hijos mediante mutación. Cada nueva generación estará compuesta de nuevos individuos, lo que maximiza el grado de exploración pero descarta antiguos individuos que posiblemente estén mejor adaptados al medio que su descendencia.
- Algoritmo $(\mu+\nu)$: $\nu-\mu$ padres dan lugar a μ hijos. Cada nueva generación está compuesta por una mezcla de los individuos más adaptados y su descendencia, lo que aumenta el grado de explotación. Existe el riesgo de que un individuo esté demasiado adaptado al medio y no tenga competencia, lo que da lugar a que el algorimto converga rápidamente a una solución subóptima.

4.2. Algoritmos genéticos

Los algoritmos genéticos son estrategias evolutivas del tipo (μ, ν) a las que se les incorporá el concepto de **recombinación**. Esto permite que la descendencia pueda heredar buenos (y malos) atributos de los padres que pueden ser muy diferentes entre sí, aumentando el grado de exploración. El proceso toma dos individuos de la población, que pueden ser

seleccionados de diferentes formas, y crea ν descendientes mediante una recombinación aleatoria de su genotipo combinado con eventuales mutaciones. El algoritmo general es:

Algorithm 20 *The Genetic Algorithm (GA)*

```
popsize ← desired population size                    ▷ This is basically λ. Make it even.
P ← {}
for popsize times do
    P ← P ∪ {new random individual}
Best ← □
repeat
    for each individual P_i ∈ P do
        AssessFitness(P_i)
        if Best = □ or Fitness(P_i) > Fitness(Best) then
            Best ← P_i
    Q ← {}                                           ▷ Here's where we begin to deviate from (μ, λ)
    for popsize/2 times do
        Parent P_a ← SelectWithReplacement(P)
        Parent P_b ← SelectWithReplacement(P)
        Children C_a, C_b ← Crossover(Copy(P_a), Copy(P_b))
        Q ← Q ∪ {Mutate(C_a), Mutate(C_b)}
    P ← Q                                            ▷ End of deviation
until Best is the ideal solution or we have run out of time
return Best
```

El concepto de recombinación depende mucho de la estructura de las soluciones. Por ejemplo, si estas toman la forma de vectores booleanos, la recombinación puede venir dada por la sustitución probabilística de n genes aleatorios (**Uniform Recombination**). Si las soluciones toman la forma de vectores reales, existen diferentes tipos de recombinación, como por ejemplo tomar valores intermedios dentro de la línea que conecta ambos vectores (**Line Recombination**).

Para aumentar el grado de exploración frente a explotación, es conveniente elegir los individuos supervivientes de manera estocástica. Un ejemplo es seleccionar con una probabilidad proporcional al nivel de adecuación de cada individuo (**Roulette Selection**). Otra posibilidad es escoger un conjunto aleatorio de t individuos de la población y seleccionar al mejor (**Tournament Selection**).

Una forma de incrementar el nivel de explotación sin comprometer el algoritmo es con técnicas de **elitismo**, en las cuales pasamos los n mejores individuos de una generación a la siguiente.

Todos estos algoritmos de población pueden incorporar ideas de los de estado único para dar lugar a los **algoritmos híbridos**, como por ejemplo el algoritmo de Evolución Diferencial.

5. Rich Vehicle Routing Problem

El Vehicle Routing Problem (VRP) es un problema computacional que pretende encontrar la distribución idónea de las rutas de un conjunto de vehículos tal que son capaces de satisfacer las demandas de consumidores sin violar un conjunto dado de restricciones. Computacionalmente, se trata en general de un problema NP-hard del que no se conocen algoritmos para encontrar soluciones óptimas. Ejemplos de VRP clásicos son los llamados Constrained VRP (CVRP) o Heterogeneous VRP (HVRP).

Existe un subconjunto de VRPs denominados Rich VRPs los cuales pretenden representar condiciones más realistas como por ejemplo comportamientos estocásticos sobre las restricciones del problema, integración con otras actividades (ej. Gestión de inventarios) u optimizaciones multiobjetivo, lo que hace que en general supongan una buena aproximación a la realidad.

Se ha pretendido resolver VRPs mediante un enfoque clásico buscando algoritmos de optimización que encuentren soluciones cercanas a las óptimas para tamaños pequeños. Un ejemplo de estos algoritmos son los de programación lineal o matemática. Sin embargo, una línea de investigación más exitosa ha sido la del desarrollo de algoritmos de optimización heurística y metaheurística, ya que pretenden encontrar soluciones buenas en un espacio corto de tiempo y para tamaños grandes. En general, un algoritmo heurístico se entiende como aquel especialmente diseñado para el problema, mientras que un algoritmo metaheurístico es uno más general, aplicable a un conjunto mayor que puede servir como guía inicial para elaborar heurísticas. Ejemplos de estos algoritmos aplicados a RVRPs son los algoritmos genéticos, Ant Colony o GRASP.

Como breve ejemplo de la aplicación de metaheurísticas a RVRP, tenemos el trabajo de Vidal et al, que ha estudiado la aplicación de más de 64 algoritmos metaheurísticos a RVRP comparándolos con las soluciones de 15 algoritmos clásicos de VRP. Este autor concluye que la metodología idónea consiste el uso de algoritmos híbridos, los cuales toman ideas de estrategias clásicas junto con metaheurísticas (matheuristics) o técnicas de simulación (simheuristics) así como con estrategias de paralelización. El propio autor remarca la importancia de mantener un balance apropiado entre la exploración y la explotación de la metaheurística en el espacio de soluciones. Estos algoritmos suponen una mejoría significativa a la hora de tratar RVRPs y suponen una línea activa de investigación académica.