Computación Concurrente

Optimización por enjambre de partículas

Chaparro Sicardo Tanibeth

Lerín Hernández Natalia

Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas
UNAM
Ciudad de México

El algoritmo de optimización por enjambre de partículas (PSO) sirve para estudiar situaciones en las que se requiere conocer soluciones óptimas bajo condiciones o criterios particulares. Se distribuye la población de partículas en un espacio de búsqueda definido de modo que los operadores propios del algoritmo impiden soluciones imprecisas, pues no se estancan óptimos locales.

El algoritmo considera una población de partículas que operan dentro de un espacio de búsqueda dado, acotado. Se repite un mismo procedimiento iterativamente una cantidad de veces finita. En cada una de las iteraciones se producen nuevas posiciones para las partículas, a partir de una velocidad dada a partir de la posición global más óptima y la mejor posición actual de cada partícula. Se toma también una función objetiva para valorar la calidad de la población por medio de la evaluación individual de cada partícula en la nueva posición. También en cada iteración hay un intercambio de información entre los elementos de la población para alcanzar más rápidamente el objetivo común.

Este trabajo se realizó siguiendo el Capítulo 4 del libro de Erik Cuevas (2016).

1. Modelo matemático

Las principales fases o etapas del algoritmo PSO estándar son las siguientes:

- Inicializar la población de partículas
- Evaluar la población en la función objetivo escogida y elegir la partícula con el mejor desempeño
- Crear un ciclo siempre que se cumpla una condición de paro
- Crear un ciclo para todas las partículas en todas las dimensiones
- Para cada partícula, otorgar una nueva velocidad
- Calcular la nueva posición dada la velocidad y posición anterior
- Nuevamente evaluar en la función objetivo y considerar la mejor partícula de entre la nueva y la anterior

- Terminar el ciclo de todas las partículas
- Actualizar la población con las mejores partículas
- Dada la condición de paro, terminar su ciclo

Se verán a continuación las distintas etapas para completar el algoritmo:

1. Inicialización
2. Velocidad de las partículas
3. Movimiento de las partículas

1.1 Inicialización

Como se observó anteriormente, el algoritmo comienza con una solución dada y ésta se va modificando (con los operadores de PSO) tomando aquellos valores que optimicen la solución y así nos acerquen al resultado deseado. Podría pensarse de esta parte como un algoritmo voraz. La posición inicial de las partículas se va a tomar considerando la siguiente definción

$$x_k^{i,t} = l_k + rand(u_k - l_k) \ , \ x_k^{i,t} \in \mathbf{x}^t \tag{1}$$

Donde $x_k^{i,t}$ es la i-ésima partícula de la población x^t; i es el número de partícula y se encuentra en $\{1, 2, ...Par_N\}$. La dimensión del problema se enceuntra definida en k y el número de iteraciones es t. Además u_k y l_k son los límites inferior y superior respectivamente para las dimensiones del espacio de búsqueda. Se utiliza *rand* para indicar un número aleatorio uniformemente distribuido en el (0,1).

1.2 Velocidad de las partículas

Ya que queremos optimizar al seleccionar la posición óptima modificando primero las posiciones, definiremos la forma en que se obtiene dicha nueva posición a comparar. Para ello primero se define una velocidad con la que cambia de posición la partícula. Cabe mencionar que la primera posición tiene velocidad previa 0.

$$\mathbf{v}^{t+1} = \mathbf{v}^t + rand_1 \times (\mathbf{P} - \mathbf{X}^t) + rand_2 \times (\mathbf{G} - \mathbf{X}^t) \tag{2}$$

Por lo mencionado anteriormente, sabemos que la anterior es una definición recursiva para la velocidad en el $t + 1$. Además, la ecuación (2) incluye $\mathbf{P}$ las mejores posiciones hasta el momento, asociadas a la vecindad de cada partícula, y $\mathbf{G}$ la posición de la mejor partícula actual a nivel global. Notemos que la anterior definición es una operación vectorial. Algo importante que debe notarse es que se está realizando una operación de velocidad dadas posiciones, por lo que hablando estrictamente de la definición de velocidad, se está considerando que cada iteración tiene un tiempo de realización constante que para conveniencia del algoritmo y las expresiones calculadas, se toma como 1. Los números aleatorios uniformes escogidos en esta ocasión tienen la finalidad de permitir que la nueva posición sea aleatoria pero siguiendo la dirección del óptimo local y global.

1.2.1 Movimiento de las partículas

Una vez que se tiene la definición de la velocidad con la que cambia la posición de las partículas, solo hará falta utilizar la definición de velocidad considerando tiempos unitarios de cambio entre y cambio entre posiciones dados por la posición en cada iteración, es decir:

$$\mathbf{x}^{t+1} = \mathbf{x}^t + \mathbf{v}^{t+1} \tag{3}$$

2. Implementación

2.1 Pseudocódigo

A continuación se muestra un pseudocódigo de la implementación secuencial del PSO.

Algoritmo 1 Optimización por enjambre de partículas, secuencial

function PSO($d, part\_n, l, u, max\_iter, funcion$)
  $\mathbf{x} = l^T + rand(Unif(0,1), (part\_n, d)) * (u-l)^T$ ▷ Iniciamos posiciones
  $obj\_func \leftarrow funcion(\mathbf{x}, d)$ ▷ Se evalúa la función
  $evol\_func\_obj \leftarrow$ vector de ceros, tamaño(1, max_iter)
  $ind \leftarrow min(OF)$ ▷ Índice del mínimo global
  $new_obj_func \leftarrow$vector de ceros, tamaño(1, part_n)
  $mejor_pos \leftarrow$posición del mínimo global
  $g_opt \leftarrow$vector de unos, tamaño(part_n,d)
  for $i \in [1, d]$ **do** ▷ Iniciamos valores óptimos
    $g\_opt(todas, i) = x(ind, i)$
  end for
  $\mathbf{v} \leftarrow$ vector de ceros, tamaño(part_n, d) ▷ Iniciamos velocidades
  t = 1 ▷ Iniciamos optimización PSO
  for $t \in [1, max\_iter]$ **do**
    $v \leftarrow$nueva velocidad
    $x \leftarrow$nueva posición
    for $i \in [1, part\_n]$ **do** ▷ Verifica partículas entre límites
      Si se salen de los límites, otorga el valor del límite.
      $new\_obj\_func(i) \leftarrow funcion(x(i, todas), d)$
      Si es menor al valor de la función objetivo, actualiza óptimo local y función objetivo
    end for
  end for
  $nvo\_glob\_opt \leftarrow min(obj\_func)$
  $ind \leftarrow$índice del nuevo mínimo
  Verifica si se actualiza el óptimo global
  $t = t + 1$ ▷ Se incrementa el contador
  return mejor_pos
end function

2.2 Secuencial

2.2.1 Rosenbrock

Al implementar PSO con 6500 partículas y 1000 iteraciones para la función escalar multivariada de evaluación de Rosenbrock en dos variables dada por

$$f(\mathbf{x}) = \sum_{i=1}^{d-1} [100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2] \tag{4}$$

se obtuvo la solución óptima:

$$[1.0006(0.0066)\,, 1.0011(0.0135)]$$

Notemos que la solución óptima *real* es $[1, 1]$ por lo que se tiene un error porcentual del -0.08% en la norma.
Esta implementación en promedio tardó 14275.33 (315.12) ms.

2.2.2 Parábola bidimensional

Como segunda prueba se consideró la ecuación de una parábola en dos dimensiones, centrada en el orígen:

$$f(\mathbf{x}) = x^2 + y^2 \tag{5}$$

Dicha función tiene un mínimo en el $[0, 0]$, y con la implementación de PSO con 6500 partículas y 1000 iteraciones se obtuvo la solución óptima:

$$[-0.00005(0.00121)\,, -0.0005(0.0004)]$$

El error absoluto presentado en la norma del resultado es de 0.0005.
Esta implementación en promedio tardó 14377.0 (437.1) ms.

2.3 Paralela

Se realizó el algoritmo en paralelo mediante la división del dominio completo. Se utilizó un método del tipo *Maestro-Esclavos*, en el cual cada hilo realiza la misma tarea de implementar el algoritmo secuencial y el maestro toma aquel valor que fue el mínimo global dentro de todos los obtenidos. De esta forma

Se implementó para las mismas funciones de prueba que en secuencial. Se trabajó con 10 nodos en cada prueba.

2.3.1 Rosenbrock

Se consideraron 6500 partículas y 1000 iteraciones en la implementación. se obtuvo la solución óptima:

$$[1.0034(0.0060)\,, 1.0067(0.0.0122)]$$

Nuevamente sabemos que la solución óptima *real* es $[1, 1]$ de modo que se se tiene un error porcentual del -0.5052% en la norma.
Esta implementación en promedio tardó 5090.4 (37.9) ms.

2.3.2 Parábola bidimensional

Esta función tiene el mínimo en el $[0, 0]$, y con la implementación de PSO con 6500 partículas y 1000 iteraciones se obtuvo la solución óptima:

$$[-0.0003(0.0009)\,, 0.0004(0.0014)]$$

El error absoluto presentado en la norma del resultado es de 0.0005.
Esta implementación en promedio tardó 5119.8 (38.05) ms.

Resumen de resultados

Rosenbrock

	Error porcentual	Tiempo de ejecución (ms)
Secuencial	-0.08%	14275.33 (315.12)
Paralelo	-0.5052%	5090.4 (37.9)

Table 1: Resumen de resultados de implementación de PSO para la función objetivo de Rosenbrock

Parábola bidimensional

	Error absoluto	Tiempo de ejecución (ms)
Secuencial	0.0005	14377.0 (437.1)
Paralelo	0.0005	5119.8 (38.05)

Table 2: Resumen de resultados de implementación de PSO para la función objetivo de una parábola bidimensional

2.4 Análisis de aceleración y escalabilidad

Sea t_{seq} el tiempo tomado por un *programa secuencial* y t_P el tiempo invertido por un *programa paralelo* equivalente ejecutado en P procesadores, se pueden definir las siguientes cantidades como Nielsen (2016):

- *speedup*: speedup(P)= $\frac{t_{seq}}{t_P}$
- *escalabilidad*: escalabilidad(O,P)= $\frac{t_O}{t_P}$

Con las definiciones anteriores podemos calcular el speed up y escalabilidad de nuestras implementaciones. En el caso de speed up tomaremos los promedios de tiempo requerido para el caso secuencial y el paralelo con 10 procesadores. Para la escalabilidad se consideró el promedio del tiempo de ejecución de las implementaciones en paralelo con 5 procesadores, de forma que $O = 5\,, P = 10\,, O < P$.

	Speed Up	Escalabilidad
Rosenbrock	2.80	1.9059
Parábola bidimensional	2.80	1.8062

Table 3: Calculo del speed up y la escalabilidad de las implementaciones.

3. Conclusiones

Los resultados obtenidos muestran una mejora evidente en el tiempo de ejecución del algoritmo estudiado luego de la paralelización. En el caso actual y con los parámetros utilizados podemos dar un vistazo al poder que se encuentra detrás del computo de alto rendimiento. Para casos reales con millones de datos o funciones de muy grandes dimensiones, aún con algoritmos eficientes como el PSO, las ejecuciones en paralelo permiten aumentar la eficiencia de los programas realizados. La implementación paralela ofrece una optimización considerable con respecto al tiempo cuando tenemos un número suficientemente grande de partículas, ya que es entonces que se aprovecha el paso de mensajes entre procesadores e hilos y la repartición de las tareas.

Referencias

Margarita Díaz Erik Cuevas, Valentín Osuna. *Optimización: Algoritmos Programados con MATLAB*. Alfaomega, México, 2016.

Frank Nielsen. *Introduction to HPC with MPI for Data Science*. Springer, Switzerland, 2016.