Práctica Alternativa al Examen: Búsqueda Ramificada con Momentos

David Cabezas Berrido

Índice

Contenido

Inspiración

Diseñaremos una metahurística inspirándonos en un fenómeno físico, la conservación del momento lineal o del movimiento.

Cuando un objeto se desprende de otro que se está moviendo, conserva su momento. Por ejemplo, esto se manifiesta cuando un paracaidista salta de un avión o se desprenden fragmentos de un meteorito.

Idea general

- Nuestra metaheurística, **Búsqueda Ramificada con Momentos**, pretende "lanzar" soluciones que se muevan por el espacio.
- Cada solución tendrá asociado un vector velocidad (momento), que determina la dirección en la que se mueve.
- Las soluciones se ramifican o dividen en otras de forma que las soluciones resultantes conserven la inercia de la solución de la que partieron.
- Necesitamos definir un algoritmo de trayectoria para las soluciones que le dé sentido al momento.

Contenido

Resumen

Cada solución S cuenta con un impulso $\lambda \in \mathbb{R}^+$, un vector momento μ y un número de evaluaciones restantes *evals*. En cada iteración hace una de las siguientes acciones.

- La solución desaparece, es truncada.
- La solución se divide en dos, que "saldrán disparadas" en direcciones opuestas. Las soluciones generadas conservan la inercia de la original.
- La solución se "desvía" hacia un vecino con mejor fitness y luego se actualiza usando su momento. La regla es: $S \leftarrow S + \lambda\mu$. La solución pierde cierta cantidad de impulso.

Acción a realizar

- Queremos truncar soluciones con escasas opciones de superar a la mejor encontrada hasta el momento. Truncaremos cuando la diferencia de fitness sea alta y el impulso bajo o queden pocas evaluaciones por hacer.

- Queremos ramificar soluciones prometedoras, pero queremos que exista variedad entre las soluciones. Dividiremos soluciones con diferencia de fitness baja y número de evaluaciones suficiente para que las soluciones generadas puedan prosperar. Conviene dejar que las soluciones resultantes de una ramificación se separen antes de volver a ramificarlas, no queremos ramificar ni muy pronto (poca variedad) ni muy tarde (pocas evaluaciones restantes).

- En la mayoría de ocasiones, la solución se limita a avanzar.

No hay una mejor forma de concretar estas ideas. A continuación ofrecemos nuestra implementación.

Algorithm 1: BRANCH: Búsqueda Ramificada: Bucle Principal.

Input: Una solución: vector de flotantes S
Input: Su momento: vector de flotantes μ
Input: Su impulso: escalar positivo λ
Input: Evaluaciones a realizar: entero positivo $evals$
$fitness \leftarrow \text{eval}(S)$ `// Supondremos la función de evaluación siempre disponible`
$evals \leftarrow evals - 1$
$best \leftarrow \min(best, fitness)$
while $evals > 0$ **do**
  $D \leftarrow fitness - best$ `// Suponemos accesible la mejor fitness obtenida hasta el momento`
  $\hat{D} \leftarrow \frac{D}{1+D}$ `// Diferencia normalizada en` $[0, 1[$
  $p \leftarrow U[0, 1]$ `// Flotante aleatorio en` $[0, 1]$`, elegido según la distribución uniforme`
  if $(p < P_{vanish} \frac{\hat{D}}{\lambda}$ ***and*** $evals \leq MaxEvalsTruncate)$ ***or*** $\lambda < MinImpulse$ **then**
    La solución es truncada.
  else if $p > P_{split} \frac{\hat{D}}{\lambda}$ ***and*** $MinImpulseSplit \leq \lambda \leq MaxImpulseSplit$ ***and*** $evals \geq MinEvalsSplit$ **then**
    La solución se divide en dos.
  else
    La solución avanza.

Truncamiento

Truncamos la solución con un simple **break** para el bucle.

Acumulamos las evaluaciones restantes para que no se desperdicien:

$$spareEvals \leftarrow spareEvals + evals$$

Ramificación

- Se generan dos soluciones con empujes opuestos desde S y se acumulan los momentos: $\mu_1 = \mu + M$, $\mu_2 = \mu - M$, donde M es el empuje.
- Los impulsos se recargan en cierta proporción del impulso perdido desde el inicial (λ_0): $\lambda_1 = \lambda_2 = \lambda + SplitImpulse \cdot (\lambda_0 - \lambda)$.
- En la mayoría de ocasiones, la solución se limita a avanzar.
- Las evaluaciones pendientes se reparten equitativamente entre las dos soluciones generadas. Se rescata la evaluación sobrante en caso de que el número de evaluaciones sea impar.
- La solución es destruida, ya que se queda sin evaluaciones.

Algorithm 2: SPLIT: Ramificación de una solución S como la de entrada de BRANCH.

$modification \leftarrow$ vector aleatorio, componentes según una $U[0,1]$
$\mu_1 \leftarrow \mu + modification$ `// Las soluciones salen en direcciones opuestas`
$\mu_2 \leftarrow \mu - modification$
$\lambda_1 \leftarrow \lambda + SplitImpulse \cdot (\lambda_0 - \lambda)$ `// Se recarga parcialmente el impulso`
$\lambda_2 \leftarrow \lambda + SplitImpulse \cdot (\lambda_0 - \lambda)$
$S_1 \leftarrow S + \lambda_1 \mu_1$
$S_2 \leftarrow S + \lambda_2 \mu_2$
clip(S_1) `// Si alguna componente se sale del rango, se fija en el borde`
clip(S_2)
if $evals \,\%2 = 1$ **then**
  $spareEvals \leftarrow spareEvals + 1$ `// Para evitar perder evaluaciones`
branch($S_1, \mu_1, \lambda_1, evals/2$)
branch($S_2, \mu_2, \lambda_2, evals/2$)
break `// Esta solución desaparece`

Avance

- Se buscan vecinos aleatorios en un entorno de la solución. En nuestro caso hemos elegido radio $\sqrt{\lambda}$.
- Si se encuentra un vecino mejor en un determinado número de intentos, éxito. De lo contrario, fracaso.

En caso de éxito:

- Se modifica el momento de la solución desviándolo (más o menos, según la mejora) hacia la dirección del vecino.
- Se desplaza la solución usando el momento y el impulso: $S \leftarrow S + \lambda\mu$.
- Se reduce ligeramente el impulso.

En caso de fracaso (posible máximo local):

- Se reduce moderadamente el impulso.

Algorithm 3: ADVANCE: Avance de la solución S.

while *no se encuentre un vecino mejor* ***and*** *no se excedan* $ImproveLimit$ *intentos* ***and*** $evals > 0$ **do**

- $modification \leftarrow$ vector aleatorio, componentes según una $U[-\sqrt{\lambda}, \sqrt{\lambda}]$
- $S' \leftarrow S + modification$ `// Vecino aleatorio`
- $\text{clip}(S')$
- $neighbor\_fitness \leftarrow \text{eval}(S')$
- $evals \leftarrow evals - 1$

if *el vecino* S' *mejora la fitness de* S **then**

- $best \leftarrow \min(best, neighbor\_fitness)$
- $D \leftarrow fitness - neighbor\_fitness$
- $\hat{D} \leftarrow \frac{D}{1+D}$ `// Mejora normalizada en` $[0, 1[$
- $neighbor\_weight \leftarrow BaseWeight + (1 - BaseWeight) \cdot \hat{D}$
- $\mu \leftarrow (1 - neighbor\_weight)\mu + neighbor\_weight \cdot modification$ `// Se desvía la inercia de la solución hacia el vecino mejor`
- $S \leftarrow S + \lambda\mu$ `// La solución se desplaza en su dirección`
- $\text{clip}(S)$
- $fitness \leftarrow \text{eval}(S)$
- $evals \leftarrow evals - 1$
- $best \leftarrow \min(best, fitness)$
- $\lambda \leftarrow DecreaseSuccess \cdot \lambda$ `// Éxito: se reduce ligeramente el impulso`

else

- $\lambda \leftarrow DecreaseFail \cdot \lambda$ `// Fracaso: se reduce moderadamente el impulso`

Invocación del algoritmo

Aplicamos el algoritmo como una búsqueda de trayectoria múltiple.

Algorithm 4: MAIN: Llamadas sucesivas al algoritmo de búsqueda Ramificada con Momentos hasta consumir todas las evaluaciones disponibles.

$best \leftarrow$ un valor mayor que cualquier posible evaluación
$spareEvals \leftarrow 10000 \cdot$ dimensión del espacio de soluciones
while $spareEvals > 0$ **do**
  $evals \leftarrow spareEvals$
  $spareEvals \leftarrow 0$
  $S \leftarrow$ vector aleatorio, componentes según una $U[-100, 100]$
  $\mu \leftarrow$ vector aleatorio, componentes según una $U[-1, 1]$
  branch$(S, \mu, \lambda_0, evals)$ `//` $spareEvals$ `puede verse incrementada durante la ejecución`

Contenido

Valor de los parámetros en nuestra experimentación

Parámetro	**Tipo / Rango**	**Valor**
λ_0	Real	1
$MaxEvalsTruncate$	Entero menor que $10000 \cdot \dim$	$1200 \cdot \dim$
$MinImpulse$	Real menor que λ_0	$0{,}01 \cdot \lambda_0$
$MinImpulseSplit$	Real menor que λ_0	$0{,}1 \cdot \lambda_0$
$MaxImpulseSplit$	Real en $]MinImpulseSplit, \lambda_0[$	$0{,}7 \cdot \lambda_0$
$MinEvalsSplit$	Entero menor que $10000 \cdot \dim$	$400 \cdot \dim$
$SplitImpulse$	Proporción (real en [0,1])	0.5
$ImproveLimit$	Entero menor que $10000 \cdot \dim$	$10 \cdot \dim$
$BaseWeight$	Proporción en $[0, 1[$	0.2
$DecreaseSuccess$	Proporción en $]0, 1[$	0.99
$DecreaseFail$	Proporción en $]0, DecreaseSuccess[$	0.9