Benchmarking the Hooke-Jeeves Method, MTS-LS1, and BSrr on the Large-scale BBOB Function Set

The BBOB-2022 workshop at Boston

Ryoji Tanabe

Yokohama National University
Yokohama, Japan
Separability in black-box numerical optimization

A $D$-dim. separable function $f$ can be $D$ 1-dim. functions

$$\arg\min_{\mathbf{x}} f(\mathbf{x}) = \left( \arg\min_{x_1} f(x_1, \ldots), \ldots, \arg\min_{x_D} f(\ldots, x_D) \right)$$

- Separable functions are easier to solve than nonseparable ones
  - If an optimizer can exploit the separability
  - E.g., Coordinate-wise optimizers

**IMHO, a separable real-world problem is very rare**

- Some decision variables are likely to depend on each other
- The motivation to study optimizers for separable functions is weak
- Just in case, it is better for an algorithm portfolio to contain an optimizer that can exploit the separability
  - An efficient algorithm selection system is available [Tanabe 22]

---

Benchmarks three optimizers for separable functions on **bbob-largescale**

1. **The Hooke-Jeeves method (HJ) [Hooke 61]**
   - One of the most classical black-box optimizers

2. **Multiple trajectory search local search 1 (MTS-LS1) [Tseng 08]**
   - Designed for the CEC LSGO competition 2008
   - Some winners of the CEC (LSGO) competitions used MTS-LS1
   - Very similar to the Hooke-Jeeves method, but it has been overlooked

3. **Brent-STEP in a round-robin manner (BSrr) [Baudis 15]**
   - State-of-the-art for the five separable **bbob** functions \((f_1, \ldots, f_5)\)
   - BSrr is a member of a portfolio in recent algorithm selection systems

---


Lin-Yu Tseng, Chun Chen: Multiple trajectory search for Large Scale Global Optimization. IEEE Congress on Evolutionary Computation 2008: 3052-3059

Petr Baudis, Petr Posík: Global Line Search Algorithm Hybridized with Quadratic Interpolation and Its Extension to Separable Functions. GECCO 2015: 257-264
The Hooke-Jeeves method: a pattern move (variable-wise operation)

- HJ iteratively improves a search point \( x \in \mathbb{R}^D \) by two moves:
  1. a pattern move (variable-wise operation)
  2. an exploratory move (vector-wise operation)
- In the pattern move, HJ generates a new point \( x^{\text{new}} \) by perturbing only one variable \( x_i \in x \) (from \( i = 1 \) to \( D \))

\[
\begin{align*}
  x^{\text{new}}_i &\leftarrow x_i + \sigma(x^{\text{up}}_i - x^{\text{low}}_i) \quad \text{or} \quad x^{\text{new}}_i \leftarrow x_i - \sigma(x^{\text{up}}_i - x^{\text{low}}_i) \\
  \sigma &\quad \text{step-size (the initial } \sigma^{\text{init}} = 0.4 \text{)} \\
  x^{\text{up}}_i \text{ and } x^{\text{low}}_i &\quad \text{the upper and lower bounds for the } i\text{-th variable}
\end{align*}
\]

- When all trials for all variables were unsuccessful, \( \sigma \leftarrow c \times \sigma \)
  - \( c \): learning rate (typically, \( c = 0.5 \))
The Hooke-Jeeves method: an exploratory move (vector-wise operation)

- If the pattern move was successful for at least one variable, HJ performs a bonus operation.
- HJ generates a new point $\mathbf{x}^{\text{new}}$ by taking the difference from the previous one $\mathbf{x}^{\text{prev}}$ to the current one $\mathbf{x}$.
  - $\mathbf{x}^{\text{new}} \leftarrow \mathbf{x} + (\mathbf{x} - \mathbf{x}^{\text{prev}})$
The overall procedure of the Hooke-Jeeves method

1. Initialize $x, \sigma \leftarrow \sigma^{\text{init}}$;
2. while not happy do
   \[ x^{\text{prev}} \leftarrow x; \]
   /* The pattern move (variable-wise operation) */
   for $i \in \{1, \ldots, D\}$ do
   \[ x^{\text{new}} \leftarrow x; \]
   \[ x_i^{\text{new}} \leftarrow x_i + \sigma(x_i^{\text{up}} - x_i^{\text{low}}); \]
   if $f(x^{\text{new}}) < f(x)$ then $x \leftarrow x^{\text{new}}$;
   else
   \[ x^{\text{new}} \leftarrow x; \]
   \[ x_i^{\text{new}} \leftarrow x_i - \sigma(x_i^{\text{up}} - x_i^{\text{low}}); \]
   if $f(x^{\text{new}}) < f(x)$ then $x \leftarrow x^{\text{new}}$;
   /* The exploratory move (vector-wise operation) */
   if $f(x) < f(x^{\text{prev}})$ then
   \[ x^{\text{new}} \leftarrow x + (x - x^{\text{prev}}); \]
   if $f(x^{\text{new}}) < f(x)$ then $x \leftarrow x^{\text{new}}$;
   else $\sigma \leftarrow c \times \sigma$;
Two main differences between MTS-LS1 and the Hooke-Jeeves method

1. MTS-LS1 does not adopt the exploratory move (vector-wise operat.)
2. MTS-LS1 reinitializes the step-size $\sigma$ when $\sigma$ is too small
The Hooke-Jeeves method vs. MTS-LS1

The Hooke-Jeeves method

1. Initialize \( x, \sigma \leftarrow \sigma^{\text{init}} \);  
2. while not happy do  
   3. \( x^{\text{prev}} \leftarrow x \);  
   4. for \( i \in \{1, \ldots, D\} \) do  
      5. \( x^{\text{new}} \leftarrow x \);  
      6. \( x^{\text{new}}_i \leftarrow x_i + \sigma(x^{\text{up}}_i - x^{\text{low}}_i) \);  
      7. if \( f(x^{\text{new}}) < f(x) \) then  
         x \leftarrow x^{\text{new}} \);  
   else  
      8. \( x^{\text{new}} \leftarrow x \);  
      9. \( x^{\text{new}}_i \leftarrow x_i - \sigma(x^{\text{up}}_i - x^{\text{low}}_i) \);  
      10. if \( f(x^{\text{new}}) < f(x) \) then  
          x \leftarrow x^{\text{new}} \);  
9. if \( f(x) < f(x^{\text{prev}}) \) then  
   10. \( x^{\text{new}} \leftarrow x + (x - x^{\text{prev}}) \);  
   11. if \( f(x^{\text{new}}) < f(x) \) then  
      x \leftarrow x^{\text{new}} \);  
12. else \( \sigma \leftarrow c \times \sigma \);  

MTS-LS1

1. Initialize \( x, \sigma \leftarrow \sigma^{\text{init}} \);  
2. while not happy do  
   3. \( x^{\text{prev}} \leftarrow x \);  
   4. for \( i \in \{1, \ldots, D\} \) do  
      5. \( x^{\text{new}} \leftarrow x \);  
      6. \( x^{\text{new}}_i \leftarrow x_i - \sigma(x^{\text{up}}_i - x^{\text{low}}_i) \);  
      7. if \( f(x^{\text{new}}) < f(x) \) then  
         x \leftarrow x^{\text{new}} \);  
   else  
      8. \( x^{\text{new}} \leftarrow x \);  
      9. \( x^{\text{new}}_i \leftarrow x_i + 0.5\sigma(x^{\text{up}}_i - x^{\text{low}}_i) \);  
      10. if \( f(x^{\text{new}}) < f(x) \) then  
          x \leftarrow x^{\text{new}} \);  
      11. if \( f(x) = f(x^{\text{prev}}) \) then  
          \( \sigma \leftarrow c \times \sigma \);  
      12. if \( \sigma(x^{\text{up}}_1 - x^{\text{low}}_1) < 10^{-15} \) then  
          \( \sigma \leftarrow \sigma^{\text{init}} \)  
13. else \( \sigma \leftarrow c \times \sigma \);
The Brent-STEP method for 1-dimensional optimization

The Brent method (e.g., fminbnd in Matlab)
- It simultaneously performs the bisection and the secant methods
- **Pros**: It performs very well on unimodal functions
- **Cons**: It performs poorly on multimodal functions

Select The Easiest Point (STEP) [Langerman 94]
- It sequentially selects an interval with the smallest difficulty
- **Pros**: It performs well on multimodal functions
- **Cons**: It generally converges slow

The Brent-STEP method aims to take their **pros**
- First, it runs the Brent method
- If the search fails (i.e., on multimodal functions), it then runs STEP

BSrr: An extension of the Brent-STEP method to $D$-dimensional opt.

- BSrr applies Brent-STEP to each variable in a round-robin manner
- It is competitive with more sophisticated ones [Posík 15]

1. Initialize $x$;
2. while not happy do
   3. for $i \in \{1, \ldots, D\}$ do
      4. $x^{\text{new}} \leftarrow x$;
      5. $x_{i}^{\text{new}} \leftarrow$ Apply a single iteration of $\text{brent\_step}$ to $x_i$;
      6. if $f(x^{\text{new}}) < f(x)$ then
         7. $x \leftarrow x^{\text{new}}$;
         8. Update internal parameters of $D$ $\text{brent\_step}$;

---

Petr Posík, Petr Baudis: Dimension Selection in Axis-Parallel Brent-STEP Method for Black-Box Optimization of Separable Continuous Functions. GECCO (Companion) 2015: 1151-1158
The three optimizers are sensitive to the order of variables

Results of MTS-LS1 on Schwefel 1.2

\[ f(x) = \sum_{i=1}^{D} (\sum_{j=1}^{i} x_j)^2 \]

- Similar to LeadingOnes, the first \( i \) variables are dependent
  - lexical: \( x_1, x_2, x_3, x_4, \ldots \)
  - random: \( x_9, x_1, x_8, x_3, \ldots \)
- Max. fevals = \( 10^5 \times D \)
- N. runs = 31

- MTS-LS1 perturbs variables in a lexical order (from \( x_1 \) to \( x_D \))
  - It can unintentionally exploit the order of variables
- Their operators are not permutation-invariant [Lehre 12]
- This issue can be very very easily addressed
  - by randomly shuffling the order of perturbations

---

Experimental setup

- The 24 bobb-largescale functions [Varelas 20]
  - Dimension $D \in \{20, 40, 80, 160, 320, 640\}$
  - The results of L-BFGS were taken from [Varelas 19] as a base line
- The Hooke-Jeeves method and MTS-LS1
  - We implemented them in C (https://github.com/ryojitanabe/largebbob2022)
  - The maximum number of function evaluations: $10^4 \times D$
  - The initial step size $\sigma^{\text{init}} = 0.4$ (is this best for HJ?)
  - The learning rate $c = 0.5$ and 0.9
    - “HJ-5” and “MTS-LS1-5” are HJ and MTS-LS1 with $c = 0.5$
    - “HJ-9” and “MTS-LS1-9” are HJ and MTS-LS1 with $c = 0.9$
- BSrr
  - We used the Python implementation of BSrr (https://github.com/pasky/step)
  - Default setting
  - The maximum number of function evaluations: $10^3 \times D$
Aggregated results on the separable function group \((f_1, \ldots, f_5)\) and the moderate conditioning function group \((f_6, \ldots, f_9)\) for \(D = 320\)

BSrr, HJ-9, and MTSLS1-9 outperform L-BFGS on \(f_1, \ldots, f_5\)

- BSrr performs the best on \(f_1, \ldots, f_5\) for all \(D\)
- HJ-5 and MTSLS1-5 (with the learning rate \(c = 0.5\)) do not work
  - \(c = 0.5\) is recommended for CEC functions, but unsuitable for BBOB?
- They are outperformed by L-BFGS on \(f_6, \ldots, f_9\)
Performance deterioration of BSrr on $f_2$ and $f_4$ for $D \geq 320$

BSrr could not reach $x^*$ on $f_2$ and $f_4$ for $D \geq 320$

- But, BSrr still performs better than the other optimizers
- The small max. f-evals ($10^3 \times D$) may be the reason
MTS-LS1 works well for $f_3$, but does not work for $f_4$

- MTS-LS1 uses (almost) the symmetric operation
- MTS-LS1 can perform poorly on a function with an asymmetric landscape structure, e.g., $f_4$
Comparison of HJ and MTS-LS1 on $f_2$ and $f_3$ for $D = 320$

HJ can outperform MTS-LS1 on unimodal functions, e.g., $f_2$
- HJ adopts the exploratory move (vector-wise operat.)

MTS-LS1 can outperform HJ on multimodal functions, e.g., $f_3$
- MTS-LS1 adopts the reinitialization strategy for the step-size $\sigma$
- HJ can be improved by a restart strategy or the reinitialization for $\sigma$
Benchmarking HJ, MTS-LS1, and BSrr on bbob-largescale

- BSrr generally performs the best on $f_1, ..., f_5$
  - BSrr can complement L-BFGS and CMA-ES variants 😊
  - Its performance deterioration was observed on $f_2$ and $f_4$

- MTS-LS1 cannot handle the asymmetricity in $f_4$
  - Due to the symmetric operation
  - The same is true for HJ

- HJ performs better than MTS-LS1 on unimodal functions
  - But, HJ is outperformed by MTS-LS1 on multimodal functions
  - A restart strategy or the reinitialization for $\sigma$ is needed

Future work

- Benchmarking the winners of the CEC LSGO competitions
  - E.g., MOS, SHADE-ILS, and CC-RDG3
  - Especially, variable-decomposition-based approaches
The C code is much faster than the Python code

<table>
<thead>
<tr>
<th>Optimizers</th>
<th>Languages</th>
<th>20-D</th>
<th>40-D</th>
<th>80-D</th>
<th>160-D</th>
<th>320-D</th>
<th>640-D</th>
</tr>
</thead>
<tbody>
<tr>
<td>HJ</td>
<td>C</td>
<td>Na</td>
<td>4.2</td>
<td>5.9</td>
<td>11</td>
<td>21</td>
<td>41</td>
</tr>
<tr>
<td>MTS-LS1</td>
<td>C</td>
<td>4.1</td>
<td>2.0</td>
<td>5.8</td>
<td>11</td>
<td>21</td>
<td>42</td>
</tr>
<tr>
<td>BSrr</td>
<td>Python</td>
<td>13</td>
<td>20</td>
<td>33</td>
<td>62</td>
<td>120</td>
<td>270</td>
</tr>
</tbody>
</table>

- CPU time to run the three optimizers on the 24 bboblargescale functions for 2D function evaluations
- Computation environment
  - Ubuntu 18.04
  - Intel(R) 52-Core Xeon Platinum 8270 (26-Core×2) 2.7GHz
  - Compile options -O2
- \( f_{21} \) for \( D = 640 \) may be particularly time-consuming
  - \( f_{21} \): the Gallagher’s Gaussian 101-me Peaks function
Unexpected results on $f_{19}$ for any $D$ pointed out by a reviewer (Thanks!)

The initialization method significantly influences the results

- The initial point in HJ-5, HJ-9, MTSLS1-5, and MTSLS1-9
  - The center of the search space $(0, ..., 0)$
- The initial point in L-BFGS and BSrr
  - randomly generated in the search space
- The solution at $(0, ..., 0)$ may have a good objective value
- Known issue? (https://github.com/numbbo/coco/issues/1851)