Iterated Local Search and Particle Swarm Optimization for Floating-point Error Estimation

Preliminary:

Both Iterated Local Search (ILS) and Particle Swarm Optimization (PSO) divide each value range into $N_{gra}$ mutual exclusive divisions. ($N_{gra}$ is a parameter that is decided by users.) Consequently, for an initial configuration $C_{init}$, ILS/PSO creates a (fixed) set of tighter configurations $C_{univ}$.

$$C_{init} : \{ \{ I_1 \mapsto R_1 \} \ldots \{ I_n \mapsto R_n \} \}$$

$$C_{univ} : \{ \{ I_1 \mapsto R_1 p_1/N_{gra} \} \ldots \{ I_n \mapsto R_n p_n/N_{gra} \} \}$$

$$0 \leq p_1, \ldots p_n < N_{gra}$$

Two configurations in $C_{univ}$, $c_x$ and $c_y$, are “neighbors” if $c_x = c_y$ or they satisfy all the following constraints: 1) $c_x$ and $c_y$ have the same domain, 2) there exists only one variable $I_i$ such that $c_x(I_i)$ and $c_y(I_i)$ are adjacent ranges, and 3) for any variable $I_j$ which is not equal to $I_i$, $c_x(I_j) = c_y(I_j)$. A configuration, $c_x$, can “move” to another configuration, $c_y$, by one step if $c_x$ and $c_y$ are neighbors. A configuration, $c_x$, can “move” $t$ steps to $c_y$ if there exists a sequence of configurations, $c_x, c_y^1, \ldots, c_y^t$, such that $c_x$ and $c_y^1$ are neighbors and $c_y^i$ and $c_y^{i+1}$ are neighbors ($1 \leq i < t - 1$). ILS/PSO searches among $C_{univ}$ and tries to detect high floating-point errors.

High-level Idea of ILS:

ILS starts from a single configuration in $C_{univ}$, randomly moves to another configuration, evaluate the configuration with all its neighbors, and chooses the “best” configuration to start the next iteration.

High-level Idea of PSO:

PSO holds a group of configurations called a “swarm.” It evaluates every configuration in the swarm and ranks them by their evaluation results (measured floating-point errors). To find a new swarm, PSO moves every configurations in the swarm according to their ranks: the higher measured floating-point error, the fewer step for the configuration to move.

Helper Functions of ILS and PSO:

Algo. 1 shows the shared helper functions used by ILS and PSO. Function $RandConf$ randomly returns a configuration in $C_{univ}$. Function $NeighborConfs$ takes a configuration and returns all its neighbor configurations. Function call $MoveConf(c, t)$ moves a given configuration $(c)$ $t$ steps.

ILS Algorithm

Before introducing ILS algorithm (in Algo. 2), we introduce an important ILS’s subroutine “IFI.” $IFI$ takes a configuration, enumerates all its neighbors, and evaluates all the enumerated configurations. After evaluations, $IFI$ returns the configuration that resulted in the highest floating-point error.

ILS algorithm (Algo. 2) is compromised by three phases:
Algorithm 1 Helper Functions for ILS and PSO

1: global: \( C_{\text{init}}, N_{\text{gra}} \)
2: procedure R\( \text{ANDCONF} \)
3: \hspace{1em} return: random \( C_{\text{univ}} \)
4: end procedure
5:
6: procedure NeighborConfs\((conf)\)
7: \hspace{1em} confs \(\leftarrow\) \{conf\}
8: \hspace{1.1em} for each \((I_i \mapsto R_i^{p/N_{\text{gra}}}) \in conf\) do
9: \hspace{2.2em} conf’ \(\leftarrow\) conf \(\setminus\) \{(I_i \mapsto R_i^{p/N_{\text{gra}}})\}
10: \hspace{2.2em} \(p^+ \leftarrow\) max\((p + 1, N_{\text{gra}} - 1)\)
11: \hspace{2.2em} \(p^- \leftarrow\) min\((p - 1, 0)\)
12: \hspace{2.2em} conf'\(+\) \(\leftarrow\) conf' \(\cup\) \{(I_i \mapsto R_i^{p^+/N_{\text{gra}}})\}
13: \hspace{2.2em} conf'\(-\) \(\leftarrow\) conf' \(\cup\) \{(I_i \mapsto R_i^{p^-/N_{\text{gra}}})\}
14: \hspace{1em} confs \(\leftarrow\) confs \(\cup\) conf'\(+\) \(\cup\) conf'\(-\)
15: end for
16: return: confs
17: end procedure
18:
19: procedure MoveConf\((conf, n_{\text{move}})\)
20: \hspace{1em} c \(\leftarrow\) conf
21: \hspace{2.1em} for \(i = 1\) to \(n_{\text{move}}\) do
22: \hspace{3.2em} c \(\leftarrow\) random NeighborConfs\((c)\)
23: \hspace{2.5em} end for
24: return: c
25: end procedure

- Phase 1 (line 22 to 29) randomly chooses \( N_{p1} \) configurations, evaluates them, and preserves the one (\( CurrCon{f} \)) which is evaluated to the highest error.
- Phase 2 (line 31) explores all \( CurrCon{f} \)'s neighbors by invoking \( IFI \).
- Phase 3 (line 33 to 40) randomly moves the \( CurrCon{f} \) by \( N_{p2} \) steps, (by invoking \( MoveCon{f} \)) and evaluates all neighbors after moving.
- Repeat phase 3 until running out of “resource” (could be time or the total number of shadow value executions).
- Both \( N_{p1} \) and \( N_{p2} \) are user-decided parameters.
- Before moving the configuration in phase 3, we have an opportunity to choose a random starting point (line 37 to 39).

Explanations of PSO (Algo. 3)

Before introducing PSO algorithm, we introduce some of PSO’s terminologies first. \([item_0 \ldots item_n]\) denotes an ordered list. Items, \( item_0 \ldots item_n \), are stored in order. For an ordered list \( l = \ldots \)
[[item_0 \ldots item_n]]$, we use $l[p]$ to denote the $p$'th item of $l$: $l[p] = item_p$. A “swarm” is an ordered list of configurations. Function $RandSwarm$ in Algo. 3 randomly generates a swarm.

PSO starts from a random swarm, and repeatedly evaluates the swarm until running out of resource. Evaluation of a swarm ($EvaSwarm$ in Algo. 3) is composed by three phases:

- Phase 1 (line 15 to 20) evaluates all configurations in a swarm.
- Phase 2 (line 21) sorts all configurations by their. The configuration with the highest error will be given the highest rank.
- Phase 3 (line 22 to 24) moves configurations with different steps according to their ranks: the higher the rank, the fewer the steps to move.
- Note that $EvaSwarm$ has side-effect on $WorstErr$. 


Algorithm 2 Iterated Local Search for Floating-point Error Estimation

1: Input: \( P, M_{\text{init}}, N_{p1}, N_{p2}, N_{\text{gra}}, N_{\text{eva}} \)
2: Output: The Highest Floating-point Error.
3: \( \text{CurrConf} \leftarrow \text{RandConf} \)
4: \( \text{global} \) \( \text{WorstErr} \leftarrow \text{Eva}(P, \text{CurrConf}, N_{\text{eva}}) \)
5:
6: procedure \( \text{IFI}(\text{conf}) \)
7: \( \text{LocalErr} \leftarrow 0 \)
8: \( \text{confs} \leftarrow \text{NeighborConfs}(\text{conf}) \)
9: for all \( c \in \text{confs} \) do
10: \( \text{err} \leftarrow \text{Eva}(P, c, N_{\text{eva}}) \)
11: if \( \text{err} > \text{LocalErr} \) then
12: \( \text{LocalErr} \leftarrow \text{err} \)
13: \( \text{LocalConf} \leftarrow c \)
14: end if
15: end for
16: if \( \text{LocalErr} > \text{WorstErr} \) then
17: \( \text{WorstErr} \leftarrow \text{LocalErr} \)
18: end if
19: return: \( \text{LocalConf} \)
20: end procedure
21:
22: for \( i = 1 \) to \( N_{p1} \) do
23: \( c \leftarrow \text{RandConf} \)
24: \( \text{err} \leftarrow \text{Eva}(P, c, N_{\text{eva}}) \)
25: if \( \text{err} > \text{WorstErr} \) then
26: \( \text{WorstErr} \leftarrow \text{err} \)
27: \( \text{CurrConf} \leftarrow c \)
28: end if
29: end for
30:
31: \( \text{CurrConf} \leftarrow \text{IFI}(\text{CurrConf}) \)
32:
33: while has resource do
34: \( c \leftarrow \text{CurrConf} \)
35: \( c = \text{MoveConf}(c, N_{p2}) \)
36: \( \text{CurrConf} \leftarrow \text{IFI}(c) \)
37: if restart then
38: \( \text{CurrConf} \leftarrow \text{RandConf} \)
39: end if
40: end while
41: return: \( \text{WorstErr} \)
Algorithm 3 Particle Swarm Optimization for Floating-point Error Estimation

1: Input: $P, C_{init}, N_{swarm}, N_{vel}, N_{gra}, N_{eva}$
2: Output: The Highest Floating-point Error.
3: global $Swarm \leftarrow \text{RandSwarm}$
4: global $WorstErr \leftarrow 0$
5:
6: procedure $\text{RandSwarm}$
7: parts $\leftarrow []$
8: for $i = 1$ to $N_{swarm}$ do
9: parts $\leftarrow$ parts $\cup \left( \text{RandConf} \mapsto 0 \right)$
10: end for
11: return: parts
12: end procedure
13:
14: procedure $\text{EvaSwarm}$
15: for all $(c \mapsto e) \in Swarm$ do
16: err $\leftarrow \text{Eva}(P, c, N_{eva})$
17: if err $> WorstErr$ then
18: WorstErr $\leftarrow$ err
19: end if
20: end for
21: Swarm $\leftarrow \text{sort-by-err: Swarm}$
22: for all $Swarm[i], 0 \leq i < N_{swarm}, Swarm[i] = c_i \mapsto err_i$ do
23: $c_i = \text{MoveConf}(c_i, N_{vel} \ast i)$
24: end for
25: end procedure
26:
27: while has resource do
28: $\text{EvaSwarm}$
29: end while
30:
31: return: WorstErr