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 $\mathbb{C}_{univ}$.

$$C_{init} : \left\{ \begin{array}{l} I_1 \mapsto R_1 \\ \dots \\ I_n \mapsto R_n \end{array} \right\} \quad \mathbb{C}_{univ} : \left\{ \left\{ \begin{array}{l} I_1 \mapsto {R_1}^{p_1/N_{gra}} \\ \dots \\ I_n \mapsto {R_n}^{p_n/N_{gra}} \end{array} \right\} \right\}$$

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

Two configurations in $\mathbb{C}_{univ}$, 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^t if there exists a sequence of configurations, $c_x, c_y^1, \dots, c_y^t$, 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 $\mathbb{C}_{univ}$ and tries to detect high floating-point errors.

High-level Idea of ILS:

ILS starts from a single configuration in $\mathbb{C}_{univ}$, 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 $\mathbb{C}_{univ}$. 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

global: C_{init}, N_{gra}
procedure RANDCONF
  return: random $\mathbb{C}_{univ}$
end procedure

procedure NEIGHBORCONFS($conf$)
  $confs \leftarrow \{conf\}$
  for each $(I_i \mapsto R_i{}^{p/N_{gra}}) \in conf$ **do**
    $conf' \leftarrow conf \setminus \{(I_i \mapsto R_i{}^{p/N_{gra}})\}$
    $p^+ \leftarrow max(p+1,\ N_{gra}-1)$
    $p^- \leftarrow min(p-1, 0)$
    $conf^+ \leftarrow conf' \cup \{(I_i \mapsto R_i{}^{p^+/N_{gra}})\}$
    $conf^- \leftarrow conf' \cup \{(I_i \mapsto R_i{}^{p^-/N_{gra}})\}$
    $confs \leftarrow confs \uplus conf^+ \uplus conf^-$
  end for
  return: $confs$
end procedure

procedure MOVECONF($conf$, n_{move})
  $c \leftarrow conf$
  for $i = 1$ *to* n_{move} **do**
    $c \leftarrow$ **random** $NeighborConfs(c)$
  end for
  return: c
end procedure

- Phase 1 (line 22 to 29) randomly chooses N_{p1} configurations, evaluates them, and preserves the one ($CurrConf$) which is evaluated to the highest error.
- Phase 2 (line 31) explores all $CurrConf$'s neighbors by invoking IFI.
- Phase 3 (line 33 to 40) randomly moves the $CurrConf$ by N_{p2} steps, (by invoking $MoveConf$) 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 $l =$

$[\![item_0 \dots item_n]\!]$, 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

```
Input: P, M_init, N_p1, N_p2, N_gra, N_eva
Output: The Highest Floating-point Error.
CurrConf ← RandConf
global WorstErr ← Eva(P, CurrConf, N_eva)

procedure IFI(conf)
    LocalErr ← 0
    confs ← NeighborConfs(conf)
    for all c ∈ confs do
        err ← Eva(P, c, N_eva)
        if err > LocalErr then
            LocalErr ← err
            LocalConf ← c
        end if
    end for
    if LocalErr > WorstErr then
        WorstErr ← LocalErr
    end if
    return: LocalConf
end procedure

for i = 1 to N_p1 do
    c ← RandConf
    err ← Eva(P, c, N_eva)
    if err > WorstErr then
        WorstErr ← err
        CurrConf ← c
    end if
end for

CurrConf ← IFI(CurrConf)

while has resource do
    c ← CurrConf
    c = MoveConf(c, N_p2)
    CurrConf ← IFI(c)
    if restart then
        CurrConf ← RandConf
    end if
end while
return: WorstErr
```

Algorithm 3 Particle Swarm Optimization for Floating-point Error Estimation

Input: P, C_{init}, N_{swarm}, N_{vel}, N_{gra}, N_{eva}
Output: The Highest Floating-point Error.
global $Swarm \leftarrow RandSwarm$
global $WorstErr \leftarrow 0$

procedure RANDSWARM
    $parts \leftarrow [\![\,]\!]$
    for $i = 1$ *to* N_{swarm} **do**
        $parts \leftarrow parts\ :\ (RandConf \mapsto 0)$
    end for
    return: $parts$
end procedure

procedure EVASWARM
    for all $(c \mapsto e) \in Swarm$ **do**
        $err \leftarrow Eva(P,\ c,\ N_{eva})$
        if $err > WorstErr$ **then**
            $WorstErr \leftarrow err$
        end if
    end for
    $Swarm \leftarrow$ **sort-by-err:** $Swarm$
    for all $Swarm[i].\ 0 \le i < N_{swarm}.\ Swarm[i] = c_i \mapsto err_i$ **do**
        $c_i = MoveConf(c_i,\ N_{vel} * i)$
    end for
end procedure

while has resource **do**
    $EvaSwarm$
end while

return: $WorstErr$