Simulated Annealing (SA) is used as a baseline for comparison by both Nature and SB. In our 2023 work [2], we implemented SA following the description in SB. We used 320 independent workers (i.e., they do not interact among themselves), stopped the SA runs at the 12-hour mark, and reported the best solution found up to that point. However, running this experiment requires multiple servers to distribute the workers, making it difficult for a regular user to integrate SA into their workflow. To improve our SA baseline, we wrapped SA with the Go-With-the-Winners (GWTW) [1] metaheuristic. In our updated implementation, we use action sampling probabilities or move ratios for swap, shift, move and shuffle to 0.24 and for flip to 0.04 and set the number of iterations for each design such that overall runtime is less than 12 hours. Algorithm 1 presents our GWTW approach and the details are as follows:

- **Lines 1-2:** Initializes all the SA workers using $initWorker$, where the first argument is the worker ID and the second argument is the seed. Based on the seed, we randomly shuffle macros of the same size and initialize the placement using spiral initialization for odd worker IDs and greedy packing for even worker IDs.
- **Lines 3-9:** Each SA worker is run for $sync\_iter$ iterations in parallel. $sync\_iter$ is set based on $sync\_freq$. We use $sync\_freq = 0.1$, meaning there are 9 synchronizations among the workers.
- **Lines 10-11:** Enables fast synchronization. When is_async is one, if any worker finishes $sync\_iter$ iterations, the rest complete their current iteration and then stop. This reduces the wait time for synchronization but introduces nondeterminism. In our SA runs, we set is_async to zero.
- **Lines 13-15:** The algorithm stops when $Iter$ iterations are performed; otherwise, $syncWorkers$ selects the top k workers based on proxy cost and replicates their macro locations and orientations to the remaining workers based on the replication ratio r.
- **Line 16:** writes out the best macro placement solution in terms of proxy cost for each worker.

We first place the soft-macros and report the proxy cost on the best macro placement solutions for each worker using the plc_client, then choose the best solution in terms of proxy cost for P&R evaluation. This GWTW-based SA achieves similar or better results compared to the ISPD23 SA using only one quarter of CPU resources. To ensure reproducibility across machines, (i) we use lookup tables for exponent computation and provide a binarized version of the lookup table in our repository; (ii) The repository includes scripts to generate Docker and Singularity images to create the same environment. We tested our implementation on Intel(R) Xeon(R) Gold and AMD EPYC 7742 CPUs and obtained identical SA solutions using the same Docker or Singularity image.

References

[1] D. Aldous and U. Vazirani, "Go With the Winners Algorithms", *Proc. FOCS*, 1994, pp. 492-501.

Algorithm 1: Go-With-The-Winners SA

Input: Move ratios: (swap, shift, flip, move, shuffle) MR,
Random seeds: $seed = 1$,
Number of iterations: $Iters$,
$N \times \#macro$ moves per iteration, with $N = 20$,
Initial temperature: $T_0 = 0.005$,
Minimum temperature: $T_{min} = 1e-8$,
Cooling rate: $\alpha = \exp\left(\frac{\ln(T_{min}/T_0)}{Iters}\right)$,
Density weight: $\gamma = 0.5$,
Congestion weight: $\lambda = 0.5$,
Number of workers: $W = 80$,
Replicated top $k = 8$ workers,
Replication ratio: $r = [10, 10, \ldots, 10]$,
Synchronization frequency: $sync\_freq = 0.1$,
Enable fast synchronization: $is\_async = 0$

Output: Macro placement solutions.

```
// Run in Parallel
for i ← 0 to W − 1 do
    workers.append(initWorker(i, seed + i, MR, N, T_0, α))
iter_count ← 0, sync_iter ← Iter × sync_freq;
while True do
    end_iter ← min(Iters, iter_count + sync_iter);
    global stop_worker ← 0;
    // Run in Parallel
    for i ← 0 to W − 1 do
        workers[i].runSA(iter_count, end_iter,
        stop_worker);
        // Thread Critical Section
        if is_async and stop_worker == 0 then
            stop_worker ← 1;
    iter_count ← end_iter;
    if iter_count == Iters then
        break
    syncWorkers(workers, k, r);
```

Write out the best solution of each worker.

[2] C.-K. Cheng, A. B. Kahng, S. Kundu, Y. Wang and Z. Wang, "Assessment of Reinforcement Learning for Macro Placement", *Proc. ISPD*, 2023, pp. 158-166.