Enhanced Whale Optimization Algorithm for Dependent Tasks Offloading Problem in Multi-edge Cloud Computing

Atluri Charitha Sri-B220213CS
Bandapalli Venkata Durga Sahithi-B220753CS
K Triveni-B220935CS
B.Tech, Computer Science and Engineering
National Institute of Technology, Calicut

December 2025

1 Code Description

Function Name: `compute_channel_gain(distance)`

Description:
This function calculates the wireless channel gain based on distance using an inverse fourth-power path loss. If the distance is greater than zero, it returns $1/\text{distance}^4$; if not, it returns 1.0.

Function Name: `shannon_rate(W, TP, Gk, σ)`

Description:
This function calculates the data rate using Shannon's capacity formula. It takes bandwidth W, transmit power TP, channel gain Gk, and noise power σ^2. It calculates the argument $1 + \frac{TP \cdot Gk}{\sigma^2}$.

Function Name: `compute_metrics_for_agent(agent, successors, Qin_MB, Qout_MB, CC_cycles, sys_params, M)`

Description:
This function calculates all the performance metrics of the solution represented by a given agent in a task-offloading optimization problem. The agent represents both the task execution order and each task's offloading location. It calculates execution time, communication time, energy consumption, and cost of each task, depending on whether a task runs locally, on a specific MEC server, or on the cloud. The algorithm calculates the total latency (L_total), total energy consumption (E_total), and total user cost (UC_total) after processing all tasks. A final objective value Z is computed through weighted coefficients combining latency, energy, and cost.

Function Name: `TLRS_normalize(task_cont_values, successors)`

Description:
This function first determines which tasks must be completed before moving on to other tasks (in-degree).Importantly, each task has a "priority score" assigned to it.After all prerequisites have been completed, the tool selects the tasks that are ready to begin, starting with the one with the highest priority score.It keeps doing this, ensuring that the final output list is a legitimate sequence in which you never begin a task before its dependencies are finished, but you also eliminate the most crucial tasks that are ready as quickly as you can.

Function Name: `linear_scale_locations(vec, PL_total)`

Description:
This function linear_scale_locations serves as a data mapper. It converts a list of numerical values (vec) into matching indices within an integer range (PL_total). In order to do this, it first determines the input data's maximum and minimum. Next, it uses linear scaling to transform the original numbers into proportionate integer locations that are uniformly distributed throughout the available PL_total slots.

Function Name: `continuous_to_agent(vec_cont, successors, PL_total)`

Description:
This function, continuous_to_agent, acts as a translator, converting a long list of continuous, floating-point numbers (vec_cont) into a structured instruction set (the agent array) for a scheduling or planning problem.It first splits the input numbers into two halves: one set dictates the priority-based order of tasks (TLRS_normalize), and the other set determines the physical locations or resources for each task (linear_scale_locations). The final output combines the prioritized task sequence with the scaled location assignments, creating a complete, ready-to-use plan.

Function Name: `WOA_update_population(pop_cont, best_cont, a)`

Description:
This function, WOA_update_population, performs the central movement step of the Whale Optimization Algorithm (WOA). For every whale (solution), it randomly chooses one of three behaviors to update its position: searching widely for new prey, encircling the best-known prey, or performing a spiral bubble-net attack. This approach dynamically guides the entire population of solutions toward the best possible outcome.

Function Name: `right_frameshift_locations(loc_list, PL_min=0, PL_max=None)`

Description:
This function, right_frameshift_locations, simulates a genetic frameshift mutation on a list of locations (loc_list), which could represent task assignments or resource placements. It shifts every location one step to the right, meaning each task inherits the location of the task that immediately preceded it, while the first task is assigned a brand new, random location within the defined boundaries.

Function Name: `left_frameshift_locations(loc_list, PL_min=0, PL_max=None)`

Description:
This function, left_frameshift_locations, performs the opposite of the right shift, simulating a leftward frameshift mutation.It shifts every location one step to the left, meaning each task takes the assignment of the task that was scheduled immediately after it, while the last task in the sequence gets a new, randomly chosen location within the allowed range.

Function Name: `left_cyclic_shift_successors_in_order(task_order, successors)`

Description:
This function, left_cyclic_shift_successors_in_order, applies a specific cyclic shift to a task sequence (task_order).For any task that has multiple successors, it rotates the order of those successors within the existing schedule one step to the left. This means the successor that was listed second takes the first position, and the original first successor is moved to the last spot, all while maintaining the overall required dependency flow.

Function Name: `load_redistribution(agent, successors, Qin_MB, CC_cycles, sys_params, M, max_iters=30)`

Description:
This function, load_redistribution, is an optimization routine designed to balance the workload across different available resources or locations (PL_total) in a scheduling plan (agent).It repeatedly calculates the total time load on each location and identifies the most overloaded (m_max) and least loaded (m_min) locations. In each step, it moves the single longest-running task from the most overloaded location to the least loaded one, aiming to smooth out the work distribution and improve overall efficiency until a balanced state is

reached or the iteration limit is hit.

Function Name: `EWA_algorithm(successors, Qin_MB, Qout_MB, CC_cycles, sys_params, M, pop_size=12, iters=200)`

Description:
This function, EWA_algorithm, implements an Enhanced Whale Optimization Algorithm (WOA) for complex scheduling and resource allocation. It starts with a randomly generated population of candidate solutions (pop_cont) and iteratively refines them using the standard WOA update rules to explore better solution spaces. Crucially, the best solution found so far (best_agent) is then subjected to a series of specialized local improvements: its task sequence is subtly shifted (left_cyclic_shift), its locations are mutated (frameshift_locations), and its resource assignments are balanced (load_redistribution). This blend of global exploration (WOA) and local, targeted refinement ensures a highly optimized final task plan that minimizes a combined objective score (best_Z).

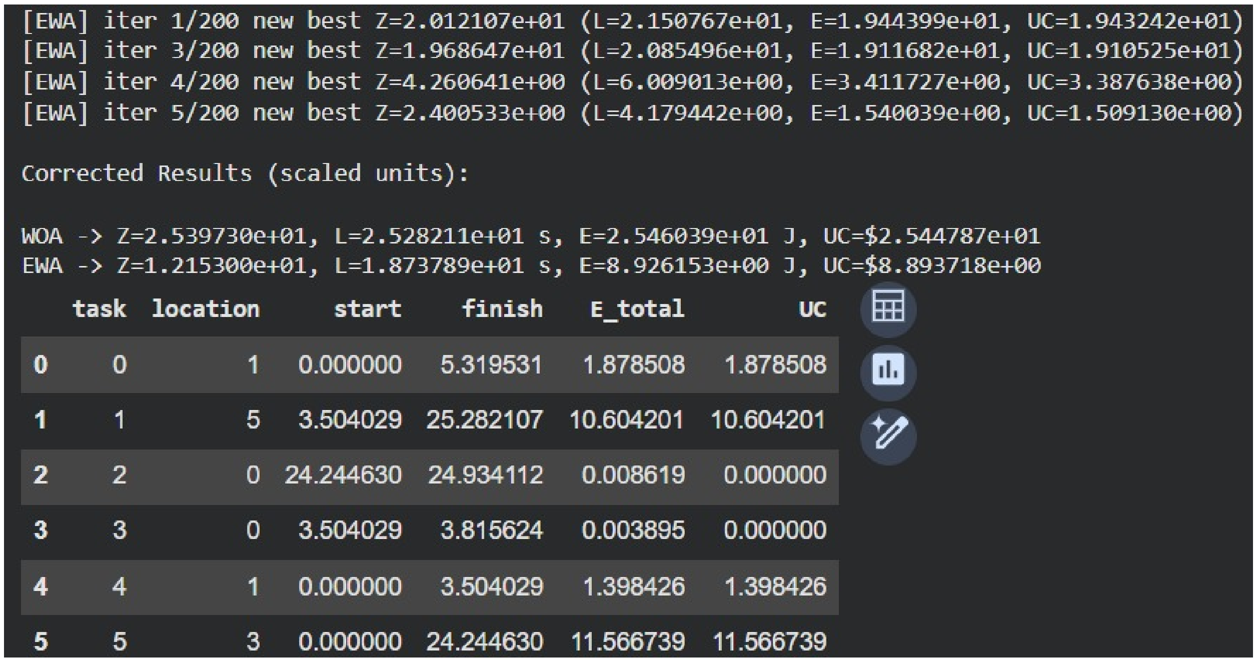
Function Name: `WOA_baseline_algorithm(successors, Qin_MB, Qout_MB, CC_cycles, sys_params, M, pop_size=12, iters=200)`

Description:
This function, WOA_baseline_algorithm, implements the standard Whale Optimization Algorithm (WOA) to find the best task schedule and resource assignment. It maintains a population of continuous candidate solutions that are converted into concrete plans (agents) for evaluation based on their combined cost metric (Z). The core of the function involves iteratively updating the population's positions using the WOA's hunting behaviors, which drives the solutions toward the known best plan found so far. This process efficiently searches for the optimal scheduling solution over many generations.

Function : `run_scheduling_optimization_comparison`

Description:
This code sets up a task scheduling problem with six tasks and four main resources, defining their dependencies (which task must follow another) and costs (data, computation cycles).It then compares the performance of two optimization techniques: the Enhanced Whale Optimization Algorithm (EWA) and the standard WOA baseline. The final output reports the minimized total cost (Z) achieved by each algorithm and displays a table showing the optimal schedule (start/finish times and assigned resource/location) generated by EWA, alongside a plot illustrating how quickly both algorithms converged to their best solution.

Optimization Results and Convergence Analysis



```
[EWA] iter 1/200 new best Z=2.012107e+01 (L=2.150767e+01, E=1.944399e+01, UC=1.943242e+01)
[EWA] iter 3/200 new best Z=1.968647e+01 (L=2.085496e+01, E=1.911682e+01, UC=1.910525e+01)
[EWA] iter 4/200 new best Z=4.260641e+00 (L=6.009013e+00, E=3.411727e+00, UC=3.387638e+00)
[EWA] iter 5/200 new best Z=2.400533e+00 (L=4.179442e+00, E=1.540039e+00, UC=1.509130e+00)

Corrected Results (scaled units):

WOA -> Z=2.539730e+01, L=2.528211e+01 s, E=2.546039e+01 J, UC=$2.544787e+01
EWA -> Z=1.215300e+01, L=1.873789e+01 s, E=8.926153e+00 J, UC=$8.893718e+00
```

	task	location	start	finish	E_total	UC
0	0	1	0.000000	5.319531	1.878508	1.878508
1	1	5	3.504029	25.282107	10.604201	10.604201
2	2	0	24.244630	24.934112	0.008619	0.000000
3	3	0	3.504029	3.815624	0.003895	0.000000
4	4	1	0.000000	3.504029	1.398426	1.398426
5	5	3	0.000000	24.244630	11.566739	11.566739

Figure 1: EWA metrics and optimal task schedule table.

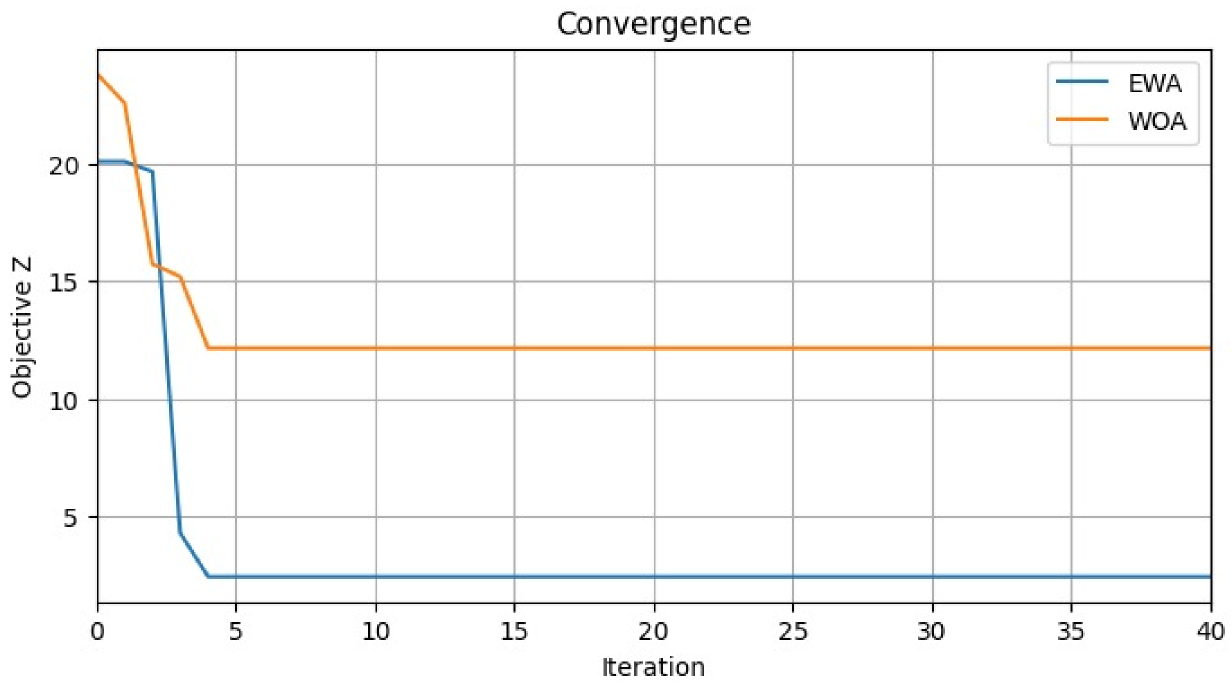


Figure 2: Convergence history of EWA vs. WOA.

Figure 3: Comparative analysis showing the performance results (a) and the objective function convergence history (b) of the EWA and WOA algorithms.