Algorithm1: LPT-Seed Greedy Scheduler

Input:

T = {t₁, t₂, …, tₘ} // set of m tasks

N = {n₁, n₂, …, nₖ} // set of k nodes

S[t][n] // execution time (or cost) of task t on node n

Output:

A[n] // for each node n, the ordered list of tasks assigned to it

L[n] // for each node n, the total load after assignment

Procedure LPT\_Seed(T, N, S):

1. // 1) Initialize loads and assignment lists

2. for each node n in N do

3. L[n] ← 0.0

4. A[n] ← [] // empty list

5. end for

6. // 2) Precompute each task’s “best possible” (minimum) time

7. Define bestTime(t) = min { S[t][n] : n ∈ N }

8. // 3) Sort tasks by descending bestTime → largest first

9. T\_sorted ← sort T by key bestTime(t), in descending order

10. // 4) Greedily assign each task to the least-loaded node

11. for each task t in T\_sorted do

12. // find the node with current smallest load

13. n\* ← arg min { L[n] : n ∈ N }

14.

15. // assign task t to node n\*

16. append t to A[n\*]

17.

18. // update that node’s load by the actual execution time on n\*

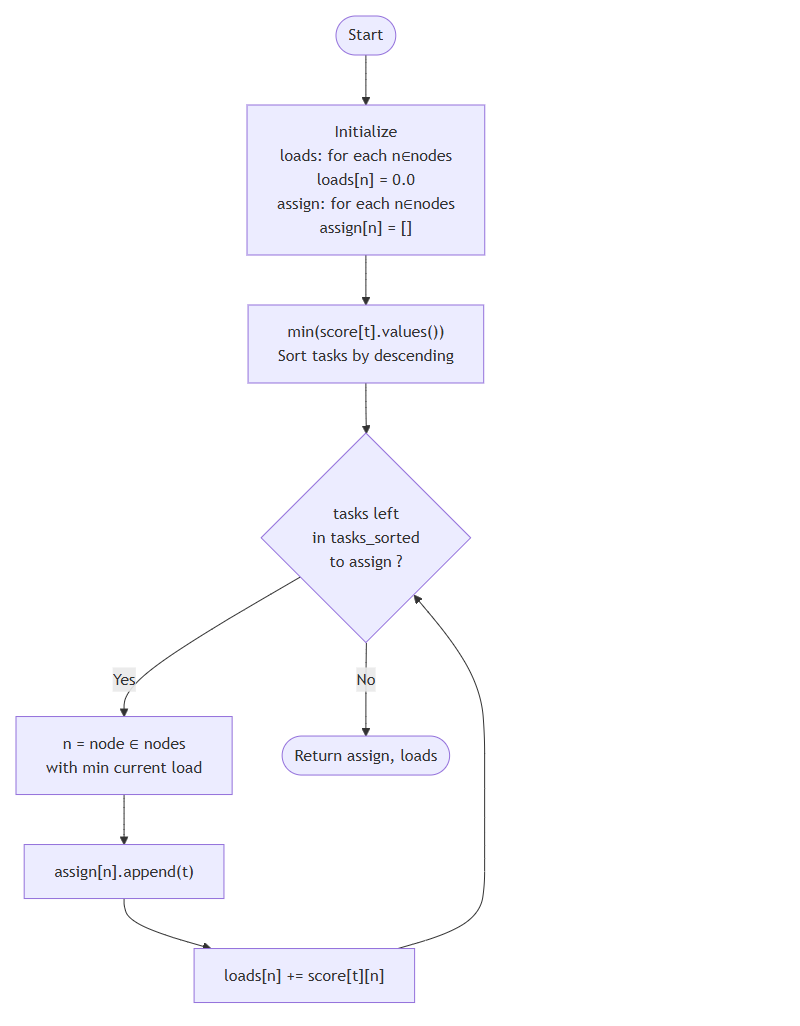
19. L[n\*] ← L[n\*] + S[t][n\*]

20. end for

21. // 5) Return the final assignment and loads

22. return A, L

End Procedure



Algorithm2: Local Search Load Balancer

──────────────────────────────────────────

Input:

• A ⟵ initial assignment, a map from each node n ∈ N to a list of tasks

• L ⟵ initial loads, a map from each node n ∈ N to its total load

• S ⟵ score matrix, where S[t][n] is the execution time of task t on node n

• N ⟵ set of all nodes

Output:

• (A, L) ⟵ an improved assignment and corresponding loads

Procedure LOCAL\_SEARCH(A, L, S, N)

1. improved ← true

2. while improved do

3. improved ← false

4. ⟶ Identify the two extreme nodes

5. heavy ← arg maxₙ∈N L[n] ⊳ node with largest load

6. light ← arg minₙ∈N L[n] ⊳ node with smallest load

7. gap₀ ← L[heavy] − L[light]

8. best\_gain ← 0.0

9. best\_op ← None

10. ⟶ Phase 1: single‐task moves from heavy → light

11. for each task t in A[heavy] do

12. new\_load\_heavy ← L[heavy] − S[t][heavy]

13. new\_load\_light ← L[light] + S[t][light]

14. other\_loads ← { L[n] : n ∈ N ∖ {heavy, light} }

15. gap\_new ← max(new\_load\_heavy, new\_load\_light, max(other\_loads))

16. − min(new\_load\_heavy, new\_load\_light, min(other\_loads))

17. gain ← gap₀ − gap\_new

18. if gain > best\_gain then

19. best\_gain ← gain

20. best\_op ← (“move”, t)

21. end if

22. end for

23. ⟶ Phase 2: two‐task swaps between heavy and light

24. for each t\_h in A[heavy] do

25. for each t\_l in A[light] do

26. new\_load\_heavy ← L[heavy] − S[t\_h][heavy] + S[t\_l][heavy]

27. new\_load\_light ← L[light] − S[t\_l][light] + S[t\_h][light]

28. gap\_new ← max(new\_load\_heavy, new\_load\_light, max(other\_loads))

29. − min(new\_load\_heavy, new\_load\_light, min(other\_loads))

30. gain ← gap₀ − gap\_new

31. if gain > best\_gain then

32. best\_gain ← gain

33. best\_op ← (“swap”, t\_h, t\_l)

34. end if

35. end for

36. end for

37. ⟶ Apply the best local improvement, if any

38. if best\_op ≠ None then

39. improved ← true

40. if best\_op[0] = “move” then

41. t ← best\_op[1]

42. remove t from A[heavy]

43. append t to A[light]

44. L[heavy] ← L[heavy] − S[t][heavy]

45. L[light] ← L[light] + S[t][light]

46. else ⊳ swap

47. (t\_h, t\_l) ← (best\_op[1], best\_op[2])

48. replace t\_h with t\_l in A[heavy]

49. replace t\_l with t\_h in A[light]

50. L[heavy] ← L[heavy] − S[t\_h][heavy] + S[t\_l][heavy]

51. L[light] ← L[light] − S[t\_l][light] + S[t\_h][light]

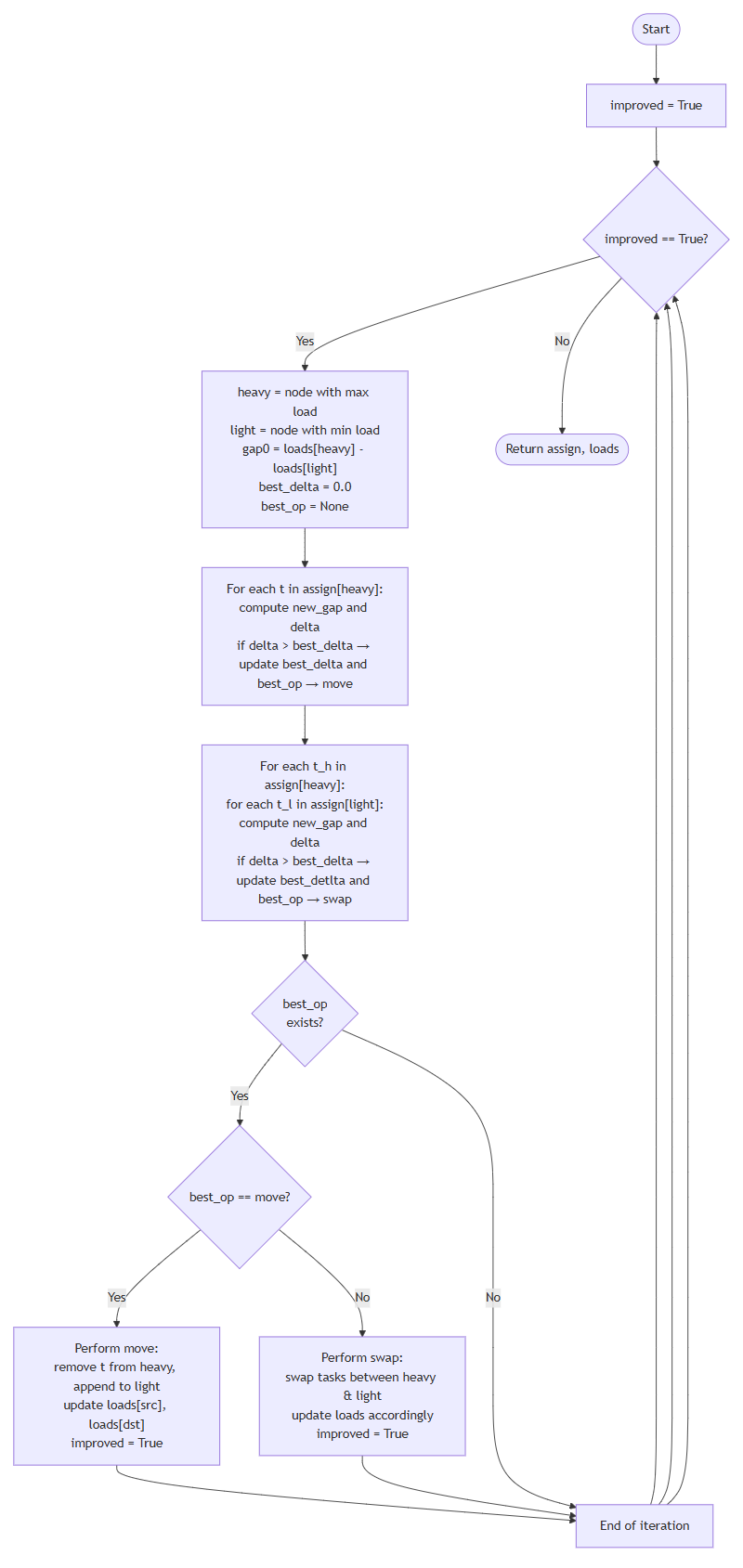
52. end if

53. end if

54. end while

55. return (A, L)

End Procedure



Algorithm 3: Multi-Start Load Balancer

────────────────────────────────────────

Input:

• T ← list of tasks

• N ← set of nodes

• S ← score matrix S[t][n] = cost of running task t on node n

• restarts ← number of random restarts (default 20)

• seed ← RNG seed for reproducibility (default 42)

Output:

• A\_best ← assignment map node → list of tasks with smallest gap

• L\_best ← corresponding node loads

• gap\_best ← load imbalance gap = max(L\_best) − min(L\_best)

Procedure BALANCE(T, N, S, restarts, seed)

1. Initialize RNG with seed

2. best\_gap ← +∞

3. A\_best ← None

4. L\_best ← None

5. for i from 1 to restarts do

6. shuffle(T) ⊳ randomize task order

7. (A, L) ← LPT\_SEED(T, N, S) ⊳ initial greedy assignment

8. (A, L) ← LOCAL\_SEARCH(A, L, S, N) ⊳ refine via local moves/swaps

9. gap ← max { L[n] : n ∈ N }

10. − min { L[n] : n ∈ N } ⊳ compute current imbalance

11. if gap < best\_gap then

12. best\_gap ← gap

13. A\_best ← copy of A

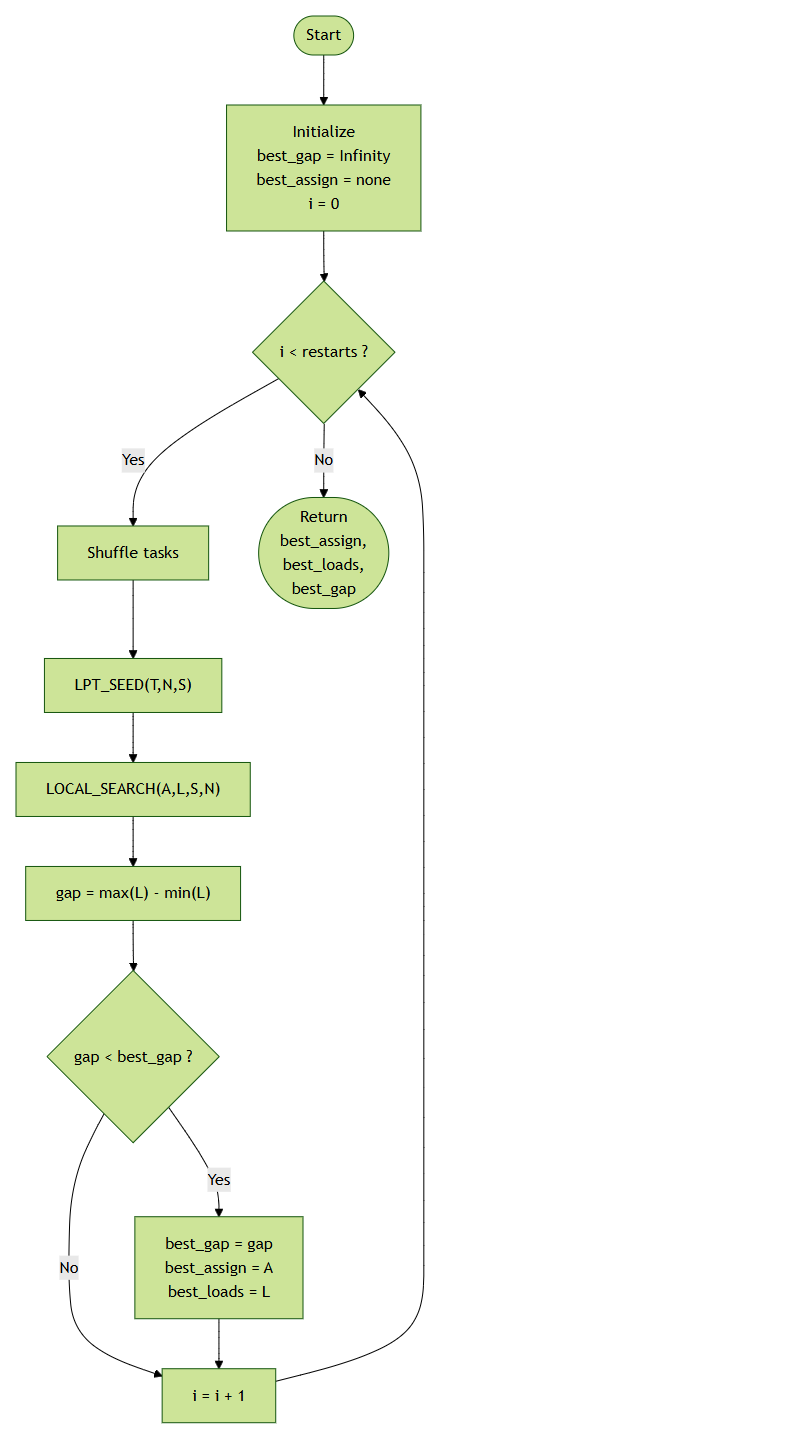
14. L\_best ← copy of L

15. end if

16. end for

17. return (A\_best, L\_best, best\_gap)

End Procedure



**All in one**

Procedure BALANCE(T, N, S, restarts, seed):

Initialize random(seed)

best\_gap ← +∞

best\_A ← None

best\_L ← None

for i = 1 to restarts do

shuffle(T)

// Stage 1: LPT Seed

(A, L) ← LPT\_SEED(T, N, S)

// Stage 2: Local Search

(A, L) ← LOCAL\_SEARCH(A, L, S, N)

// Evaluate

gap ← max { L[n] } – min { L[n] }

if gap < best\_gap then

best\_gap ← gap

best\_A ← deepcopy(A)

best\_L ← deepcopy(L)

end if

end for

return (best\_A, best\_L, best\_gap)

End Procedure

Procedure LPT\_SEED(T, N, S):

for each n in N:

load[n] ← 0

assign[n] ← empty list

end for

// sort tasks by descending “best-case” time

T\_sorted ← sort T by key t ↦ min { S[t][n] for n in N }, descending

for each t in T\_sorted do

n\* ← arg min { load[n] for n in N }

append t to assign[n\*]

load[n\*] ← load[n\*] + S[t][n\*]

end for

return (assign, load)

End Procedure

Procedure LOCAL\_SEARCH(A, L, S, N):

repeat

improved ← false

heavy ← arg max { L[n] }

light ← arg min { L[n] }

gap0 ← L[heavy] – L[light]

best\_gain ← 0

best\_op ← None

// Try moving any t from heavy → light

for t in A[heavy]:

compute gain if moved

record if gain > best\_gain

end for

// Try swapping any t\_h↔t\_l between heavy/light

for t\_h in A[heavy]:

for t\_l in A[light]:

compute gain if swapped

record if gain > best\_gain

end for

end for

if best\_op exists:

apply best\_op to (A, L)

improved ← true

end if

until not improved

return (A, L)

End Procedure