Force-Driven Minimum Latency Resource-Constrained Scheduling (ML-RCS)

Silvia Bonenti and Please Lukau

ECE/CS 565: Physical Design Automation

Fall 2025

Outline

High-Level Synthesis and Scheduling

- High-Level Synthesis (HLS) converts algorithmic descriptions into RTL hardware.
- **Scheduling** decides *when* each operation in a Data Flow Graph (DFG) executes.
- Performance depends on:
  - **Latency** – total execution time.
  - **Resource usage** – number of functional units (FUs).
- The **ML-RCS problem** generalizes the classical **MR-LCS** formulation.

From MR-LCS to ML-RCS

- **MR-LCS:** Fixed latency, minimize functional units (area-focused).
- **ML-RCS:** Fixed FU budget, minimize latency (performance-focused).
- ML-RCS reverses the classical problem — realistic when hardware limits are known but timing must be optimized.

Aspect	**MR-LCS**	**ML-RCS**
Objective	Minimize FUs	Minimize latency
Constraint	Fixed latency	Fixed FUs a_k
Focus	Area optimization	Timing optimization

Key Transition

MR-LCS $\Rightarrow$ **ML-RCS**: flip objective and constraint to move from *area minimization* to *latency minimization*.

The Need for Dual Forces

- Scheduling in HLS must balance two competing goals:
  1. **Resource usage:** keep FU utilization $\leq a_k$ per cycle.
  2. **Latency optimization:** minimize overall completion time.
- Classical Force-Directed Scheduling (FDS) considers only **resource forces** to spread operations over time.
- This scheduling algorithm for ML-RCS introduces an additional **latency force** to prioritize operations on the critical path.
- Combined, these two forces guide the scheduler to produce feasible yet fast schedules.

Key Idea

Resource force prevents overuse of functional units, while **latency force** reduces critical-path delay. Their interaction defines a balanced scheduling priority.

Motivation

- Local scheduling choices can appear optimal but increase global latency.
- Example: starting an operation too early may create resource congestion that delays later operations.
- The dual-force model avoids this pitfall by integrating:
  - **Resource balancing** — distributes FU demand evenly.
  - **Delay awareness** — preserves timing along critical paths.

Outcome

ML-RCS yields schedules that are both **fast (low latency)** and **resource-efficient (within FU limits)**.

Iterative List Scheduling Framework

- This scheduling algorithm extends list scheduling into an iterative refinement loop.
- Each iteration includes three core phases:
  1. **Priority Computation** — derive urgency from mobility and congestion.
  2. **List-Based Scheduling** — assign operations respecting FU limits.
  3. **Latency Update** — compare L_{final} and L_{target}.
- The process repeats until latency converges or the iteration limit is reached.

Essence

Priority $\rightarrow$ *Schedule* $\rightarrow$ *Update* $\rightarrow$ *Repeat* ensures progressively improved latency-aware scheduling.

Iterative Latency Adjustment

- Initialize L_{target} to the ASAP-based critical-path latency.
- After each scheduling iteration:
  - Compute achieved latency L_{final}.
  - If $L_{\text{final}} \neq L_{\text{target}}$, set $L_{\text{target}} \leftarrow L_{\text{final}}$.
- Recompute ALAP, probabilities, and priorities with the new target.
- Iterate until convergence or a maximum iteration limit is reached.

Effect

The feedback loop gradually stabilizes the schedule around a feasible minimum-latency solution.

Algorithm Forces

- The algorithm defines two interacting forces for each operation:
  1. **Latency Force (S)** — measures timing urgency (low mobility $\Rightarrow$ high priority).
  2. **Resource Force (C)** — measures congestion impact (high usage $\Rightarrow$ penalized).
- Both forces combine into a unified priority:

$$F(u) = S_{\text{norm}}(u) \cdot \left(C_{\text{norm}}(u) + \varepsilon\right)$$

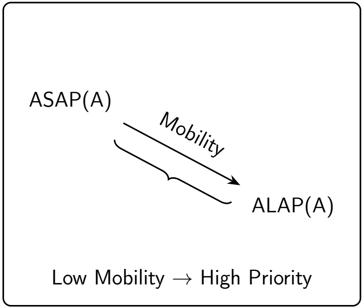
- Lower $F(u)$ means higher scheduling urgency.

Summary

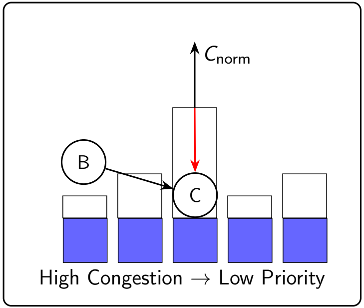
The dual-force interaction balances delay minimization and resource feasibility within a single priority metric.

Algorithm Forces

Latency Force: S_{norm} (Slack)



Resource Force: C_{norm} (Congestion)



$$F(u) = S_{\text{norm}}(\text{Mobility}) \times (C_{\text{norm}}(\text{Congestion}) + \epsilon)$$

Conceptual Summary

- Iterative refinement $\rightarrow$ progressively improved schedules.
- Each loop updates probabilities and latency target for LS.
- Combines:
    - Delay awareness (latency force)
    - Resource feasibility (resource force)
- Outcome: convergence to a **balanced, efficient** ML-RCS schedule.

Timing Bounds: ASAP and ALAP

For each operation u in the DFG:

- **ASAP time** (earliest start, given dependencies):

$$ASAP(u) = \max_{v \in Pred(u)} \big(ASAP(v) + Latency(v)\big)$$

- **ALAP time** (latest start without exceeding L_{target}):

$$ALAP(u) = L_{\text{target}} - DownLen(u)$$

- **Mobility range**:

$$[ASAP(u), ALAP(u)] \quad \Rightarrow \quad \text{feasible start window for } u.$$

- Low mobility $\Rightarrow$ high urgency in scheduling.

Resource Utilization Probability

Probabilistic FU usage for type k:

$$q_k(m) = \sum_{u:type(u)=k} p_u(m)$$

Per-operation distribution:

$$p_u(m) = \begin{cases} \dfrac{1}{ALAP(u) + Latency(u) - ASAP(u)}, & ASAP(u) \leq m \leq ALAP(u) + Latency(u) - 1 \\ 0, & \text{otherwise} \end{cases}$$

- $p_u(m)$ spreads u uniformly over its mobility range.
- $q_k(m)$ = expected occupancy of FU type k at cycle m.
- Peaks in $q_k(m)$ indicate **potential congestion**.

Non-Myopic Congestion Cost

Goal: capture downstream congestion along the critical path starting at u.

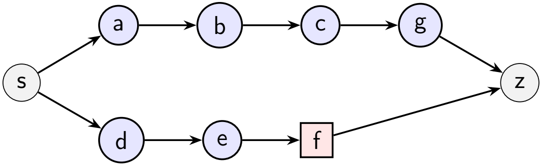
$$C(u) = \frac{1}{|P(u)|} \sum_{v \in P(u)} \frac{q^{max}_{type(v)}}{a_{type(v)}}$$

- $P(u)$: amount of operations on the critical path starting at u.
- $q^{max}_{type(v)}$: peak probability for v's FU type on the interval $[ASAP(v), ALAP(v) + Latency - 1]$
- $a_{type(v)}$: available units of that type.
- High $C(u) \Rightarrow u$ lies in a congested region.

Why non-myopic?

Decisions for u consider not only u's immediate successors, but rather its *successors along the critical path starting from u*, avoiding schedules that serialize competing paths.

Example: Myopic vs Non-Myopic Scheduling



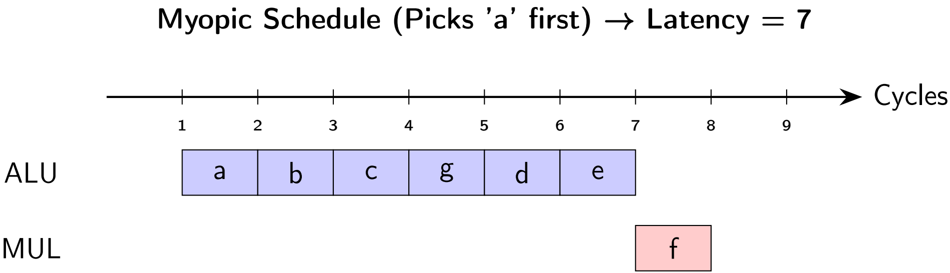
Myopic:

- Favors only critical path.
- Starves $d - e - f$ path.
- MUL f pushed late $\Rightarrow$ latency $= 7$.

Non-myopic (our $C(u)$):

- Sees ALU congestion on $a - b - c$.
- Interleaves paths; exploits parallel MUL.
- Latency improved to 6.

Example: Myopic vs Non-Myopic Scheduling

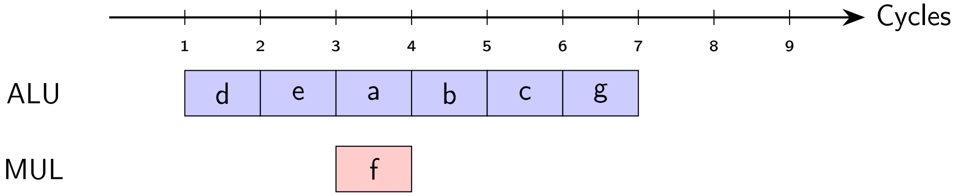


Result

A Myopic scheduler executes the path a first. This choice serializes the execution of the two main paths, resulting in a suboptimal latency of 7 cycles.

Example: Myopic vs Non-Myopic Scheduling

Non-Myopic Schedule (Picks 'd' first) $\rightarrow$ Latency = 6



Result

Our Non-Myopic scheduler correctly identifies `d` as the higher priority task. This choice allows the MUL operation `f` to be scheduled in parallel ("hidden"), resulting in a faster schedule (6 vs. 7 cycles).

Priority Function

Dual-force priority for each operation u:

$$F(u) = S_{\text{norm}}(u) \times \left(C_{\text{norm}}(u) + \varepsilon\right)$$

- $S_{\text{norm}}(u)$: normalized slack (latency force).
- $C_{\text{norm}}(u)$: normalized congestion cost (resource force).
- ε: small constant for numerical stability.

- **Lower** $F(u) \Rightarrow$ **higher** scheduling priority.
- Favors nodes that are time-critical *and* in low-congestion regions.
- Guides the list scheduler toward balanced, low-latency solutions.

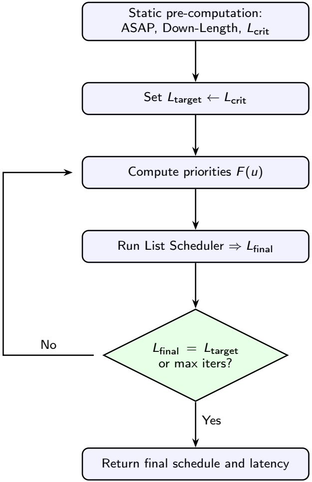
Algorithm Overview

The ML-RCS algorithm operates in four main stages:

1. **Main Scheduler**: iterative refinement of the target latency L_{target}.
2. **Priority Calculation**: compute dual-force priorities from mobility and congestion.
3. **Congestion Cost**: non-myopic cost along critical successor paths.
4. **List Scheduler**: schedule operations under dependencies and FU limits using $F(u)$.

- These components interact in a feedback loop until latency stabilizes.

High-Level Flowchart



Main Scheduler

Role: control the outer refinement loop.

- Build DFG with SOURCE/SINK.
- Compute:
  - ASAP times and Down-Length.
  - Critical-path latency L_{crit}.
- Initialize:

$$L_{\text{target}} \leftarrow L_{\text{crit}}, \quad L_{\text{final}} \leftarrow 0.$$

- Repeat (under max-iteration limit):
  1. Compute priorities using current L_{target}.
  2. Run list scheduler $\Rightarrow L_{\text{final}}$.
  3. Update $L_{\text{target}} \leftarrow L_{\text{final}}$.
- Stop when L_{final} stabilizes or `MAX_ITERATIONS` reached.

Priority Calculation

FUNCTION Calculate_All_Priorities($graph$, $resources$, L_{target})

1. Compute **ALAP** for all nodes using L_{target}.
2. Build FDS-style probability distributions $p_u(m)$ and FU profiles $q_k(m)$.
3. Compute **congestion cost** $C(u)$ via non-myopic propagation.
4. For each operation u:
   - $S_{\text{raw}}(u) \leftarrow (ALAP(u) - ASAP(u)) + 1$
   - $C_{\text{raw}}(u) \leftarrow u.\text{congestion\_cost}$
5. Normalize:
$$S_{\text{norm}},\ C_{\text{norm}} \leftarrow \text{Normalize over all ops}$$
6. Compute priority:
$$F(u) = S_{\text{norm}}(u) \cdot (C_{\text{norm}}(u) + \varepsilon).$$

- Lower $F(u) \Rightarrow$ scheduled earlier by the list scheduler.

Congestion Cost Estimation

FUNCTION Calculate_Congestion_Cost(*graph*, *dist_graphs*, *resources*)

- Process nodes in **reverse topological order** (from SINK to SOURCE).
- For each node u:
  1. If u is not SOURCE/SINK:
     $$C_{\text{local}}(u) = \frac{q^{max}_{type(u)}}{\#\ \text{FUs of type}(u)}$$
  2. Let v be the critical successor of u (if any).
  3. Accumulate:
     $$u.c\_path\_sum = C_{\text{local}}(u) + v.c\_path\_sum$$
     $$u.c\_path\_len = 1 + v.c\_path\_len$$
  4. Define:
     $$u.congestion\_cost = \frac{u.c\_path\_sum}{u.c\_path\_len}$$
- This propagates bottlenecks backwards along the critical path.

List Scheduler

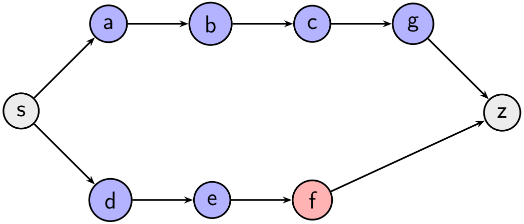
FUNCTION Run_List_Scheduler(*graph*, *resources*, F)

- Initialize:
  - $CurrentCycle \leftarrow 1$
  - ReadyList $\leftarrow$ successors of SOURCE
  - InProgressList $\leftarrow \emptyset$
- While SINK not finished:
  1. Release FUs from completed operations; update ReadyList.
  2. Sort ReadyList by increasing $F(u)$.
  3. For each u in sorted ReadyList:
     - If required FU available: schedule u at $CurrentCycle$.
  4. Increment $CurrentCycle$.
- Return final schedule and L_{final} (SINK start time - 1).

Summary

Ensures precedence + resource constraints; guided by dual-force priority.

Example 1: DFG



Resource	Symb	Latency	# Units
ALU		1 CC	1
MUL		1 CC	1
SRC/SINK		0 CC	-

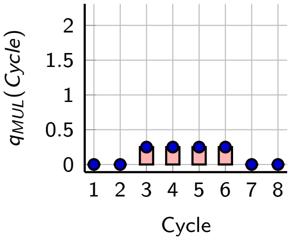
Example 1: Main Scheduler

- ASAP: $T^s_s = 1$, $T^s_a = 1$, $T^s_b = 2$, $T^s_c = 3$, $T^s_d = 1$, $T^s_e = 2$, $T^s_f = 3$, $T^s_g = 4$, $T^s_z = 5$
- Down_Length: $D_s = 4$, $D_a = 4$, $D_b = 3$, $D_c = 2$, $D_d = 3$, $D_e = 2$, $D_f = 1$, $D_g = 1$, $D_z = 0$
- $L_{target} = L_{crit} = 4$

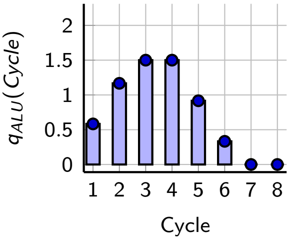
Example 1: Iteration 2 - Priority Calculation

- $L_{target} = L_{final} = 6$
- ALAP (L_{target}):
  $T_s^L = 2,\ T_a^L = 2,\ T_b^L = 3,\ T_c^L = 4,\ T_d^L = 3,\ T_e^L = 4,\ T_f^L = 5,\ T_g^L = 5,\ T_z^L = 6$

FDS Graph - MUL



FDS Graph - ALU



Example 1: Iteration 2 - Priority Calculation

Congestion Cost (C):

- $C_z = 0.0$
- $C_{f,local} = \frac{0.25}{1.0} = 0.25$. $C_f = \frac{1}{2} \cdot (C_{f,local} + C_z) = \frac{1}{2} \cdot (0.25 + 0.0) = 0.125$
- $C_{g,local} = \frac{1.5}{1.0} = 1.5$. $C_g = \frac{1}{2} \cdot (C_{g,local} + C_z) = \frac{1}{2} \cdot (1.5 + 0.0) = 0.75$
- $C_{e,local} = \frac{1.5}{1.0} = 1.5$. $C_e = \frac{1}{3} \cdot (C_{e,local} + C_{f,local} + C_z) = \frac{1}{3} \cdot (1.5 + 0.25 + 0.0) = 0.58$
- $C_{c,local} = \frac{1.5}{1.0} = 1.5$. $C_c = \frac{1}{3} \cdot (C_{c,local} + C_{g,local} + C_z) = \frac{1}{3} \cdot (1.5 + 1.5 + 0.0) = 1.0$
- $C_{d,local} = \frac{1.5}{1.0} = 1.5$.
  $C_d = \frac{1}{4} \cdot (C_{d,local} + C_{e,local} + C_{f,local} + C_z) = \frac{1}{4} \cdot (1.5 + 1.5 + 0.25 + 0.0) = 0.81$
- $C_{b,local} = \frac{1.5}{1.0} = 1.5$.
  $C_b = \frac{1}{4} \cdot (C_{b,local} + C_{c,local} + C_{g,local} + C_z) = \frac{1}{4} \cdot (1.5 + 1.5 + 1.5 + 0.0) = 1.12$
- $C_{a,local} = \frac{1.5}{1.0} = 1.5$.
  $C_a = \frac{1}{5} \cdot (C_{a,local} + C_{b,local} + C_{c,local} + C_{g,local} + C_z) = \frac{1}{5} \cdot (1.5 + 1.5 + 1.5 + 1.5 + 0.0) = 1.2$

Example 1: Iteration 2 - Priority Calculation

Latency Force (S):

- $S_a = (S_a^L - S_a^S) + 1 = (3 - 1) + 1 = 3$
- $S_b = (S_b^L - S_b^S) + 1 = (4 - 2) + 1 = 3$
- $S_c = (S_c^L - S_c^S) + 1 = (5 - 3) + 1 = 3$
- $S_d = (S_d^L - S_d^S) + 1 = (4 - 1) + 1 = 4$
- $S_e = (S_e^L - S_e^S) + 1 = (5 - 2) + 1 = 4$
- $S_f = (S_f^L - S_f^S) + 1 = (6 - 3) + 1 = 4$
- $S_g = (S_g^L - S_g^S) + 1 = (6 - 4) + 1 = 3$

Example 1: Iteration 2 - Priority Calculation

Final Priorities:

- $C_{a,norm} = \frac{C_a}{C_{max}} = 1.0$. $S_{a,norm} = \frac{S_a}{S_{max}} = 0.75$. $P_a = S_{a,norm} \cdot (C_{a,norm} + 0.0001) = 0.75$
- $P_b = S_{b,norm} \cdot (C_{b,norm} + 0.0001) = 0.75 \cdot 0.93 = 0.7$
- $P_c = S_{c,norm} \cdot (C_{c,norm} + 0.0001) = 0.75 \cdot 0.83 = 0.625$
- $P_d = S_{d,norm} \cdot (C_{d,norm} + 0.0001) = 1.0 \cdot 0.675 = 0.675$
- $P_e = S_{e,norm} \cdot (C_{e,norm} + 0.0001) = 1.0 \cdot 0.48 = 0.48$
- $P_f = S_{f,norm} \cdot (C_{f,norm} + 0.0001) = 1.0 \cdot 0.104 = 0.104$
- $P_g = S_{g,norm} \cdot (C_{g,norm} + 0.0001) = 0.75 \cdot 0.625 = 0.469$

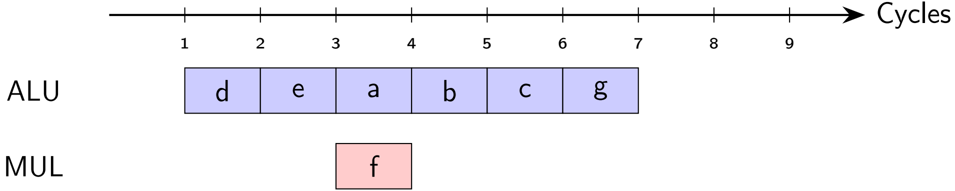
Sorted List: $[f, g, e, c, d, b, a]$

Example 1: Iteration 2 - List Scheduler

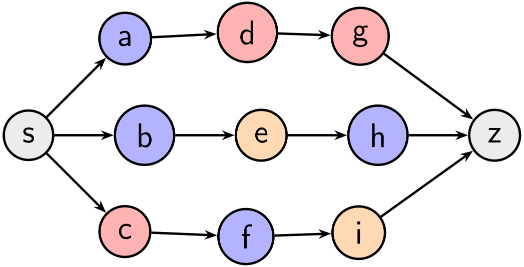
Final Priorities:

- Cycle 1: [d]
- Cycle 2: [e]
- Cycle 3: [a, f]
- Cycle 4: [b]
- Cycle 5: [c]
- Cycle 6: [g]
- Cycle 7: [z]

Final Scheduling → Latency = 6



Example 2: DFG

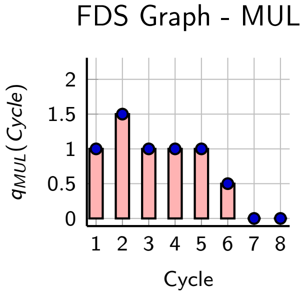


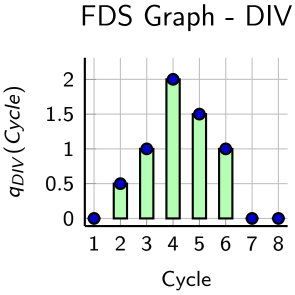
Resource	**Symb**	**Latency**	# Units
ALU		1 CC	1
MUL		2 CC	1
DIV		3 CC	1
SRC/SINK		0 CC	-

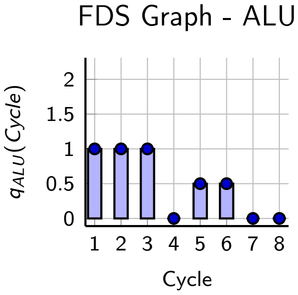
Example 2: Iteration 1 - Main Scheduler

- ASAP: $T_s^s = 1$, $T_a^s = 1$, $T_b^s = 1$, $T_c^s = 1$, $T_d^s = 2$, $T_e^s = 2$, $T_f^s = 3$, $T_g^s = 4$, $T_h^s = 5$, $T_i^s = 4$, $T_z^s = 7$
- Down_Length:
  $D_s = 6, D_a = 5, D_b = 5, D_c = 6, D_d = 4, D_e = 4, D_f = 4, D_g = 2, D_h = 1, D_i = 3, D_z = 0$
- $L_{target} = L_{crit} = 6$

- ALAP (L_{target}): $T_s^L = 1,\ T_a^L = 2,\ T_b^L = 2,\ T_c^L = 1,\ T_d^L = 3,\ T_e^L = 3,\ T_f^L = 3,\ T_g^L = 5,\ T_h^L = 6,\ T_i^L = 4,\ T_z^L = 7$







Example 2: Iteration 1 - Priority Calculation

Congestion Cost (C):

- $C_z = 0.0$
- $C_{i,local} = \frac{2.0}{1.0} = 2.0.$ $C_i = \frac{1}{2} \cdot (C_{i,local} + C_z) = \frac{1}{2} \cdot (2.0 + 0.0) = 1.0$
- $C_{h,local} = \frac{0.5}{1.0} = 0.5.$ $C_h = \frac{1}{2} \cdot (C_{h,local} + C_z) = \frac{1}{2} \cdot (0.5 + 0.0) = 0.25$
- $C_{g,local} = \frac{1.0}{1.0} = 1.0.$ $C_g = \frac{1}{2} \cdot (C_{g,local} + C_z) = \frac{1}{2} \cdot (1.0 + 0.0) = 0.5$
- $C_{f,local} = \frac{1.0}{1.0} = 1.0.$ $C_f = \frac{1}{3} \cdot (C_{f,local} + C_{i,local} + C_z) = \frac{1}{3} \cdot (1.0 + 2.0 + 0.0) = 1.0$
- $C_{e,local} = \frac{2.0}{1.0} = 2.0.$ $C_e = \frac{1}{3} \cdot (C_{e,local} + C_{h,local} + C_z) = \frac{1}{3} \cdot (2.0 + 0.5 + 0.0) = 0.83$
- $C_{d,local} = \frac{1.5}{1.0} = 1.5.$ $C_d = \frac{1}{3} \cdot (C_{d,local} + C_{g,local} + C_z) = \frac{1}{3} \cdot (1.5 + 1.0 + 0.0) = 0.83$
- $C_{c,local} = \frac{1.5}{1.0} = 1.5.$
  $C_c = \frac{1}{4} \cdot (C_{c,local} + C_{f,local} + C_{i,local} + C_z) = \frac{1}{4} \cdot (1.5 + 1.0 + 2.0 + 0.0) = 1.125$
- $C_{b,local} = \frac{1.0}{1.0} = 1.0.$
  $C_b = \frac{1}{4} \cdot (C_{b,local} + C_{e,local} + C_{h,local} + C_z) = \frac{1}{4} \cdot (1.0 + 2.0 + 0.5 + 0.0) = 0.875$
- $C_{a,local} = \frac{1.0}{1.0} = 1.0.$
  $C_a = \frac{1}{4} \cdot (C_{a,local} + C_{d,local} + C_{g,local} + C_z) = \frac{1}{4} \cdot (1.0 + 1.5 + 1.0 + 0.0) = 0.875$

Example 2: Iteration 1 - Priority Calculation

Latency Force (S):

- $S_a = (S_a^L - S_a^S) + 1 = (2 - 1) + 1 = 2$
- $S_b = (S_b^L - S_b^S) + 1 = (2 - 1) + 1 = 2$
- $S_c = (S_c^L - S_c^S) + 1 = (1 - 1) + 1 = 1$
- $S_d = (S_d^L - S_d^S) + 1 = (3 - 2) + 1 = 2$
- $S_e = (S_e^L - S_e^S) + 1 = (3 - 2) + 1 = 2$
- $S_f = (S_f^L - S_f^S) + 1 = (3 - 3) + 1 = 1$
- $S_g = (S_g^L - S_g^S) + 1 = (5 - 4) + 1 = 2$
- $S_h = (S_h^L - S_h^S) + 1 = (6 - 5) + 1 = 2$
- $S_i = (S_i^L - S_i^S) + 1 = (4 - 4) + 1 = 1$

Example 2: Iteration 1 - Priority Calculation

Final Priorities:

- $\boxed{C_{a,norm} = \frac{C_a}{C_{max}} = 0.78. \; S_{a,norm} = \frac{S_a}{S_{max}} = 1.0. \; P_a = S_{a,norm} \cdot (C_{a,norm} + 0.0001) = 0.78}$
- $\boxed{P_b = S_{b,norm} \cdot (C_{b,norm} + 0.0001) = 1.0 \cdot 0.78 = 0.78}$
- $P_c = S_{c,norm} \cdot (C_{c,norm} + 0.0001) = 0.5 \cdot 1.0 = 0.5$
- $P_d = S_{d,norm} \cdot (C_{d,norm} + 0.0001) = 1.0 \cdot 0.74 = 0.74$
- $P_e = S_{e,norm} \cdot (C_{e,norm} + 0.0001) = 1.0 \cdot 0.74 = 0.74$
- $P_f = S_{f,norm} \cdot (C_{f,norm} + 0.0001) = 0.5 \cdot 0.88 = 0.44$
- $P_g = S_{g,norm} \cdot (C_{g,norm} + 0.0001) = 1.0 \cdot 0.44 = 0.44$
- $P_h = S_{h,norm} \cdot (C_{h,norm} + 0.0001) = 1.0 \cdot 0.22 = 0.22$
- $P_i = S_{i,norm} \cdot (C_{i,norm} + 0.0001) = 0.5 \cdot 0.88 = 0.44$

Sorted List: $[h, f, g, i, c, d, e, a, b]$

PROBLEM: a should have higher priority value (lower priority) than b for optimal solution.

Let:

- N: number of operations (nodes) in the DFG.
- E: number of dependencies (edges).
- K: number of outer iterations in `Main_Scheduler`.
- L_{max}: maximum latency (clock cycles) of the schedule.

Goal

Estimate computational and space complexity for all major algorithm stages.

Time Complexity - Main Scheduler

Setup Phase

- Build Graph, ASAP, and Down-Length traversals $\Rightarrow O(N+E)$.

$$T_{\text{setup}} = O(N+E)$$

Iterative Phase

- Each iteration calls:
  - `Calculate_All_Priorities`
  - `Run_List_Scheduler`

$$T_{\text{iter}} = K \times \left(T_{\text{Priorities}} + T_{\text{Scheduler}}\right)$$

Total

$$T_{\text{Main}} = O(N+E) + K \times \left(T_{\text{Priorities}} + T_{\text{Scheduler}}\right)$$

Time Complexity - Calculate_Congestion_Cost

- Reverse topological traversal visits each node and edge once:

$$O(N + E)$$

- For each node, find $q_{\max}$ over up to $L_{\max}$ cycles:

$$O(N \times L_{\max})$$

$$T_{\text{congestion}} = O(NL_{\max} + E)$$

Interpretation: reverse traversal ensures bottlenecks propagate backward along the critical path.

Time Complexity - Calculate_All_Priorities

Steps and costs:

1. ALAP computation $\Rightarrow O(N + E)$
2. Probability graphs $(p_u, q_k) \Rightarrow O(NL_{\max})$
3. Congestion cost $\Rightarrow O(NL_{\max} + E)$
4. Normalization + combination $\Rightarrow O(N)$

$$T_{\text{Priorities}} = O(NL_{\max} + E)$$

Dominated by congestion cost and probability updates across the time horizon.

Time Complexity - Run_List_Scheduler

- Outer loop: up to $L_{\max}$ cycles.
- Per cycle:
  - Update ready list and finish ops: $O(N)$
  - Sort ready list (up to N ops): $O(N \log N)$

$$T_{\text{Scheduler}} = O(L_{\max} N \log N)$$

Sorting dominates per-cycle cost $\Rightarrow$ pseudo-polynomial behavior.

Overall Time Complexity

Combine setup + iterations:

$$
\begin{aligned}
T_{\text{Total}} &= T_{\text{setup}} + T_{\text{iter}} \\
&= O(N + E) + K \times \Big((NL_{\max} + E) + (L_{\max} N \log N) \Big)
\end{aligned}
$$

Dominant term:

$$
\boxed{T_{\text{Total}} = O\big(K(L_{\max} N \log N + E)\big)}
$$

Remarks

- Pseudo-polynomial — depends on numeric parameter $L_{\max}$.
- K is small (1–3), so iteration overhead is minor.

Overall Time Complexity

Dominant time factors:

- Probability + congestion computations: $O(NL_{\max})$
- Sorting ready list each cycle: $O(L_{\max} N \log N)$

These costs are consistent with iterative, force-directed list schedulers.

Space Complexity

Space requirements:

- Graph + dependency lists: $O(N+E)$
- ASAP/ALAP/priorities: $O(N)$
- Resource distributions: $O(L_{\text{max}})$

$$\boxed{\text{Space} = O(N + E + L_{\text{max}})}$$

Theoretical Results

Goal: minimize total latency while satisfying precedence and resource constraints.

$$G = (V, E), \quad K = \text{set of FU types}, \quad a_k = \text{resource limit per type}.$$

Define:

$$x_{il} = \begin{cases} 1, & \text{if operation } v_i \text{ starts at cycle } l \\ 0, & \text{otherwise} \end{cases}$$

$$d_i = \text{operation latency of } v_i$$

Constraints:

1. Unique Start: $\sum_l x_{il} = 1$
2. Precedence: $\sum_l l x_{il} \geq \sum_l l x_{jl} + d_j$
3. Resource: $\sum_{i:T(v_i)=k} \sum_{m=l-d_i+1}^{l} x_{im} \leq a_k$

Feasibility of the Schedule

Claim 1. The schedule from `Run_List_Scheduler` is feasible. **Proof Sketch:**

- **Precedence:** Operations enter the ReadyList only after all predecessors finish. $\Rightarrow$ start times always respect dependencies.
- **Resources:** Each FU type k maintains an availability counter a_k. An op of type k executes only if $a_k > 0$; counter decremented/incremented upon start/finish. $\Rightarrow$ never exceeds hardware capacity.

Conclusion

Schedule is precedence- and resource-feasible.

Termination of the Iterative Scheme

Claim 2. `Main_Scheduler` always terminates. **Proof Sketch:**

- Termination guaranteed by `MAX_ITERATIONS` cap.
- Ideal stopping condition: $L_{\text{final}}^{(i)} = L_{\text{target}}^{(i)}$.
- Since list scheduler and priority computation are deterministic, same L_{target} produces same L_{final}.
- Even if the sequence $\{L_{\text{target}}^{(i)}\}$ oscillates, explicit iteration bound ensures eventual stop.

Conclusion

Algorithm halts after finite iterations, ensuring termination.

Lower Bound Respect

Claim 3. The algorithm never returns a latency smaller than the critical path length.
Reasoning:

- ASAP and Down-Length analysis preserve true dependency timing.
- SINK node's ASAP equals L_{crit}.
- Any valid schedule must satisfy $L_{\text{final}} \geq L_{\text{crit}}$.

Conclusion

`Run_List_Scheduler` produces latency $\geq L_{\text{crit}}$, always respecting the fundamental lower bound.

Optimality in the Unconstrained Case

Claim 4. If all resource limits a_k are large enough to avoid conflicts, the algorithm produces the optimal schedule. **Proof Sketch:**

- With unlimited resources, all ready operations can start ASAP.
- Scheduler always places ready ops at earliest feasible cycle.
- Resulting latency equals the ASAP schedule $\Rightarrow L_{\text{final}} = L_{\text{crit}}$.

Conclusion

ML-RCS is optimal when resources do not constrain scheduling.

Summary of Theoretical Properties:

- Always produces a **valid (feasible)** schedule.
- Guaranteed to **terminate**.
- Respects the **critical path lower bound**.
- Becomes **optimal** in unconstrained resource cases.
- Remains heuristic in the general NP-hard setting.

Interpretation

Dual-force ML-RCS balances local mobility and global congestion, yielding practical, theoretically grounded solutions for complex DFGs.

Algorithm Limitations

- **Suboptimal by Design (Greedy Heuristic):**
  - Scheduling is NP-complete; we aim for a good solution, not a guaranteed optimal one.
  - Our algorithm is a **greedy** List Scheduler: it makes the best local choice (based on our heuristic) and **never backtracks** to fix strategic mistakes.
- **Heuristic Limitations (e.g., False Ties):**
  - The priority formula ($F = S \cdot C$) is an estimate, not a perfect model of cost.
  - It can create *false ties* (like a vs. d in Example. 2), forcing an arbitrary (e.g., alphabetical) tie-break.

Conclusion

- Proposed a suboptimal **Iterative List Scheduling** algorithm for the **ML-RCS problem**.
- Combines Force-Directed Scheduling (FDS) concepts with a dual-force priority model:
    - *Latency force* — urgency from scheduling mobility.
    - *Resource force* — probabilistic, non-myopic FU congestion.
- Iterative refinement of L_{target} balances delay and resource use.

Proven properties:

1. Produces valid, resource-feasible schedules.
2. Guaranteed termination.
3. Respects the critical-path lower bound.
4. Optimal when resources are unconstrained.

Complexity:

$$T_{\text{Total}} = O(K(L_{\max} N \log N + E)), \quad \text{Space} = O(N + E + L_{\max})$$