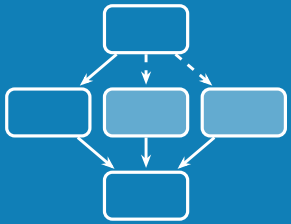
Adapting Classic List-Scheduling Heuristics for Conditional Task Graphs

Gabriel Twigg-Ho[1], **Jared Coleman**[2], Tainã Coleman[2], Mohammadali Khodabandehlou[3], Bhaskar Krishnamachari[3]

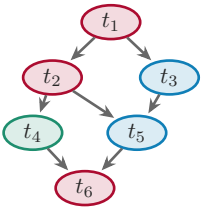
[1] Swinburne University of Technology [2] Loyola Marymount University
[3] University of Southern California



Presented by Jared Coleman · jared.coleman@lmu.edu
WORKS 2026 at SC26 · Chicago · November 15, 2026

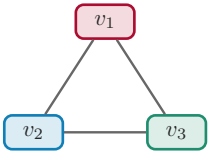
Task Graph Scheduling

Task graph



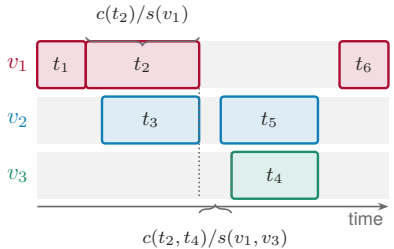
$c(t)$: task cost
$c(t, t')$: data size

Compute network



$s(v)$: compute speed
$s(v, v')$: link speed

Schedule



Minimize makespan
The time the last task finishes

Related machines
A node faster for one task is faster for all

NP-hard
So in practice we rely on heuristics

A Large Literature, Mostly Unused

Decades of heuristics

BIL · CPoP · ETF · FCP · FLB · GDL · HEFT · MaxMin · MinMin · MCT · MET · OLB · WBA · …

- Few are used in practice
- Designed for variants that do not match what practitioners run: uncertain inputs, changing resources, runtime branches

Our approach: adapt the literature

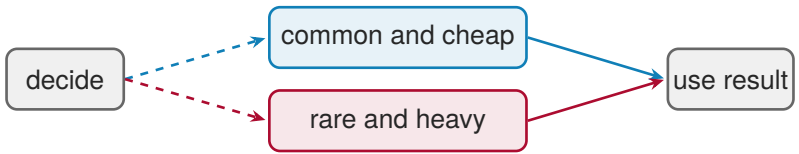
- **SAGA**: one implementation of many algorithms, for comparison and reuse
- **Parametric list scheduler**: 72 algorithms from a few interchangeable components
- **Online scheduling** under uncertainty, from the same components

Many Workflows Branch at Runtime

Multi-fidelity simulation
A cheap surrogate, or the full simulation

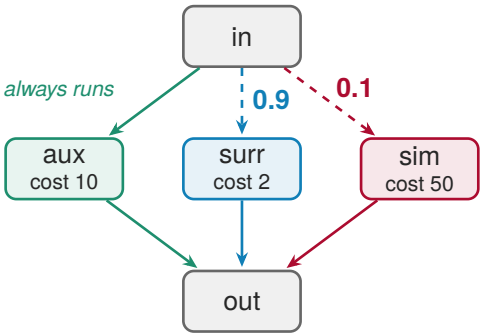
Inference cascades
A small model, or escalate to a large one

Approximate computing
An approximate kernel, or an exact one



Only one alternative runs, and we do not know which in advance.

Conditional Task Graphs



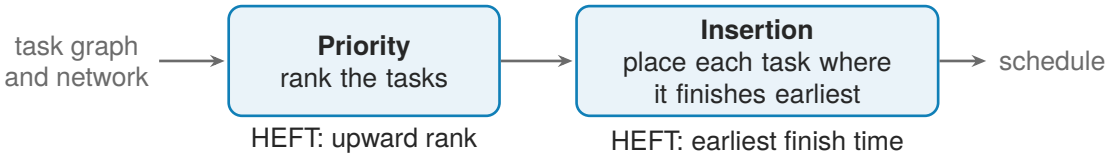
Two nodes: fast (speed 2) and slow (speed 1). Run time is cost divided by speed.

- **Conditional edges**: a branch takes exactly one, with a known probability
- **Traces**: the sets of tasks that run together

  in, **aux**, **surr**, **out** probability 0.9
  in, **aux**, **sim**, **out** probability 0.1
- **Mutually exclusive**: no trace contains both, like **surr** and **sim**
- **aux** and **sim** are *not* exclusive

Goal: find a schedule that minimizes **expected makespan**.

List Scheduling Assumes Every Task Runs



Classic heuristics

- HEFT: upward rank, then earliest finish time
- CPoP and many others differ in a component or two

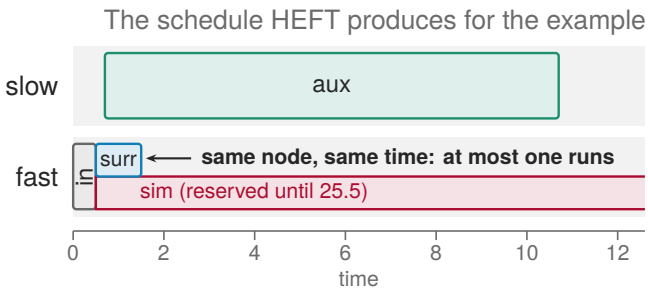
The problem

- They all assume every task in the graph executes
- With runtime branches, that is false

Can we adapt these heuristics instead of designing new ones?

Change 1: Let Exclusive Tasks Overlap

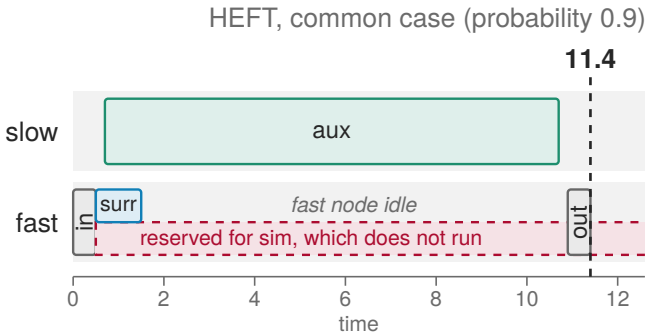
- Ordinary rule: tasks on one node never overlap in time
- New rule: they may overlap **if they are mutually exclusive**
- An **overlap policy** that the insertion step consults
- Exclusive pairs computed in polynomial time



Overlap Alone Is Not (Always) Enough

What HEFT did

- **sim** has the largest cost, so it gets top priority and claims the fast node
- **aux** is not exclusive with **sim**, so it is pushed to the slow node
- In 90% of runs **sim** never executes, but **aux** is still on the slow node

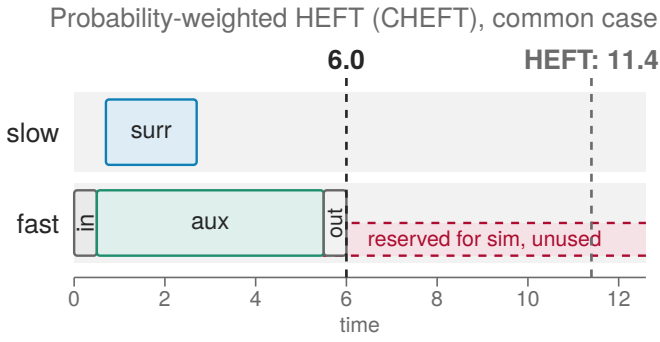


The schedule is optimized for a branch that is almost never taken.

Change 2: Weight Priority by Probability

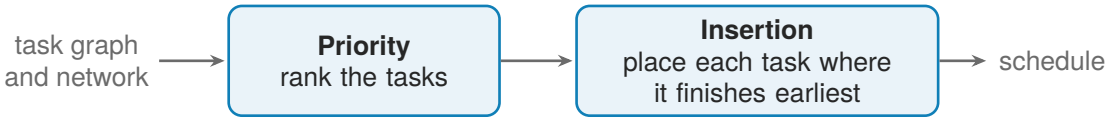
- Scale each task's cost by its **probability of executing**, then rank as usual
- **sim**: $50 \times 0.1 = 5$, below **aux** at 10
- **aux** goes first and takes the fast node

Makespan	HEFT	CHEFT
Common case (0.9)	11.4	**6.0**
Rare case (0.1)	26.0	31.0
Expected	12.9	**8.5**



Weighted HEFT is **CHEFT**. Weighted CPoP is **CCPoP**. The insertion step is unchanged.

Two Changes, Any List Scheduler



Both are components of SAGA's parametric scheduler, so every algorithm it expresses can now schedule conditional task graphs.

Scheduler	Priority	Tasks that may overlap
HEFT	upward rank	mutually exclusive tasks
CHEFT	**probability-weighted** upward rank	mutually exclusive tasks
CPoP	CPoP rank, critical path pinned	mutually exclusive tasks
CCPoP	**probability-weighted** CPoP rank	mutually exclusive tasks
No-overlap baseline	upward rank	none

Evaluation Against a Clairvoyant Scheduler

Metric: expected makespan ratio

$$\sum_{\tau} P(\tau)\,\frac{m_\tau(S)}{m^{\mathrm{off}}(\tau)}$$

- $m_\tau(S)$: makespan of our schedule S when trace τ runs
- $m^{\mathrm{off}}(\tau)$: a **clairvoyant** scheduler that knows τ in advance and schedules only its tasks (HEFT)
- A ratio of 1 means not knowing the branch cost nothing

Two datasets

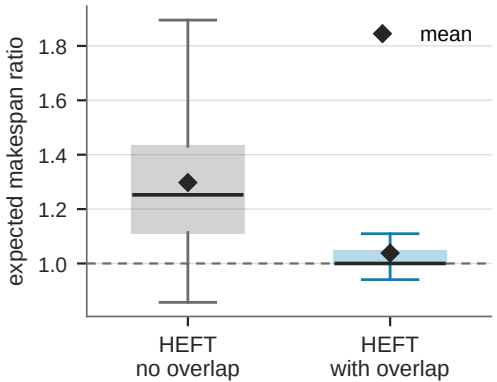
Generic conditional DAGs

Nested series, parallel, and conditional blocks, after the real-time conditional DAG model. Dirichlet(α) branch probabilities. 400 instances.

Surrogate-model family

The example's shape: a rare heavy branch, an always-run shared task, one fast and one slow node. 500 instances.

Result 1: Overlap Is the Dominant Benefit



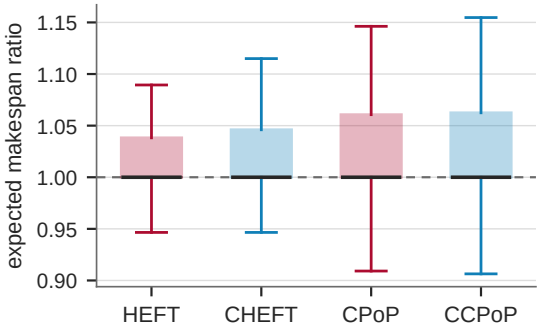
451 random conditional graphs. Boxes: median and IQR, whiskers at 1.5 times the IQR, outliers omitted. Dashed line: clairvoyant.

1.30
mean ratio when overlap is forbidden

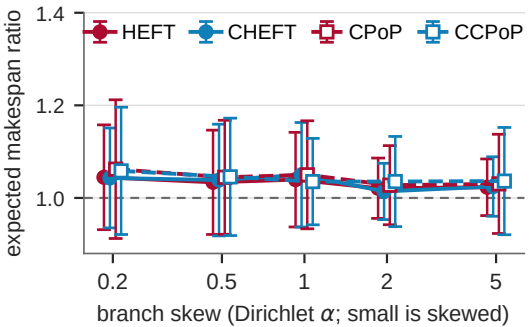
1.04
mean ratio with overlap (median 1.00)

- Treating the graph as an ordinary one costs about 30%
- Exclusive branches never contend, so one schedule serves every trace almost as well as its own
- Any list scheduler gets this benefit

Result 2: Reweighting Doesn't Always Help

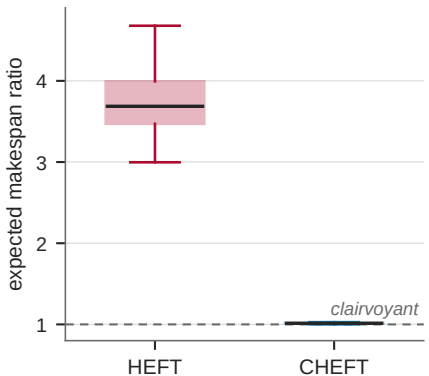


- Per instance: identical medians. CHEFT beats HEFT on only about 10% of instances



- No gain even with skewed branches: weighting only **reorders** insertion, and exclusive branches do not contend

Result 3: Reweighting Wins on Surrogates



Surrogate-model family, the same 500 instances for both

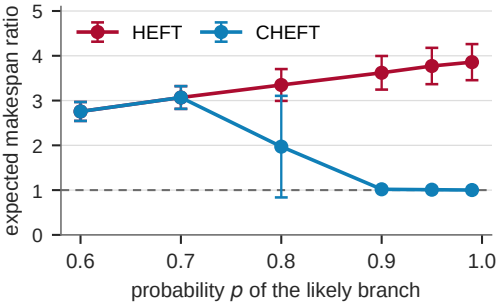
3.69
HEFT, median ratio

1.01
CHEFT, median ratio

100%
of instances: CHEFT is closer to clairvoyant

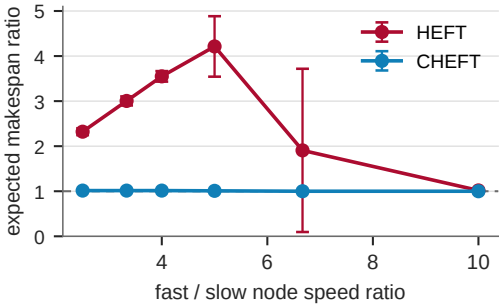
- The always-run task and the rare heavy task compete for the fast node
- Unweighted: the rare task wins, and the common case pays
- Weighted: the priorities reverse. CPoP and CCPoP behave the same way

When Does Reweighting Help?



When one branch dominates

Near-equal branches: the schedulers coincide. The gap widens as the likely branch approaches probability 1.



When the fast node is contested

The gap grows with the speed ratio, then closes once the rare task is no longer heavy enough to displace the shared task.

Conclusions

Takeaways

- **Two small changes** adapt classic list schedulers to conditional task graphs
- **Overlap** is the dominant benefit: within a few percent of clairvoyant
- **Probability weighting** is decisive only when a rare, heavy branch competes with always-run work for a fast node

Limitations and future work

- Synthetic graphs and an idealized runtime model
- Next: real conditional workflows and a live workflow system
- Next: probability-aware *insertion*

Thank you! Questions? github.com/ANRGUSC/saga · jared.coleman@lmu.edu

Supported by the National Science Foundation under Award No. 2451267 and in part by the Army Research Laboratory under Cooperative Agreement W911NF-17-2-0196.