Performance Evaluation of List Based Scheduling on Heterogeneous Systems

Hamid Arabnejad and Jorge Barbosa
Departamento de Engenharia Informática
Universidade do Porto, Faculdade de Engenharia
LIACC – Laboratório de Inteligência Artificial e Ciência de Computadores
Porto, Portugal

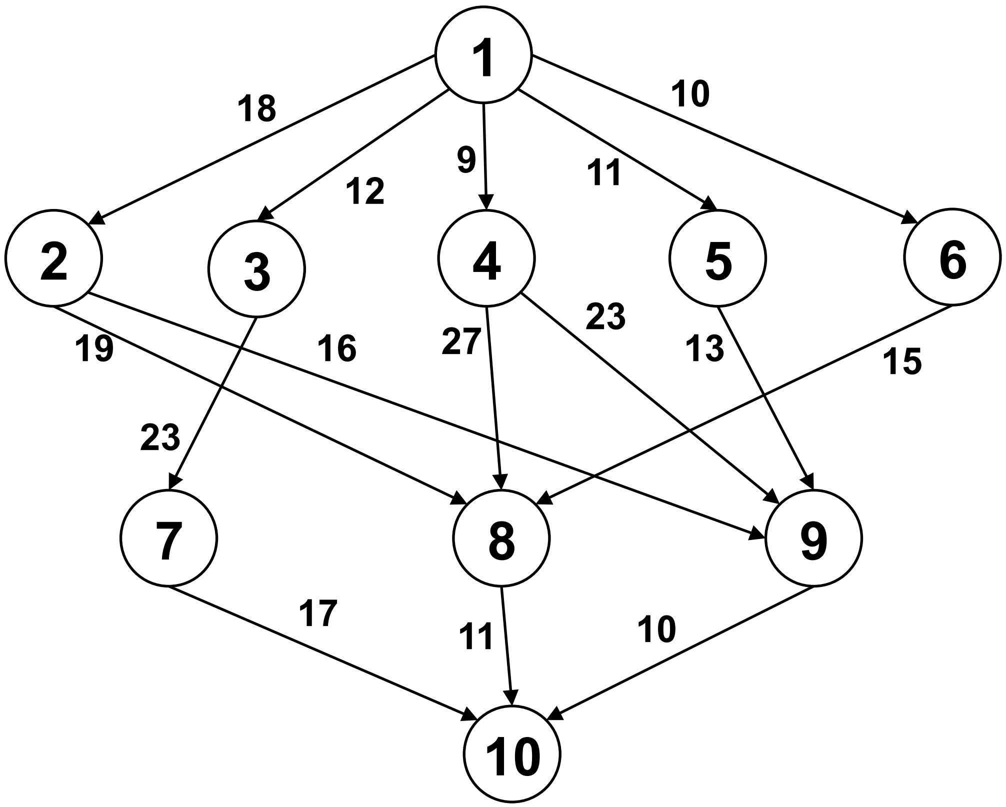
Heteropar'2011

August 29, 2011, Bordeaux, France

Contents

Introduction

Job representation by a DAG（directed acyclic graph）



Task	P1	P2	P3
T1	14	19	9
T2	13	19	18
T3	11	17	15
T4	13	8	18
T5	12	13	10
T6	12	19	13
T7	7	15	11
T8	5	11	14
T9	18	12	20
T10	17	20	11

Introduction

Each node n_i (task) has a schedule Start-time ST(n_i)
and a Finish-time FT(n_i)

Schedule length: $\max_i\{FT(n_i)\}$

Goal of scheduling: minimize $\max_i\{FT(n_i)\}$

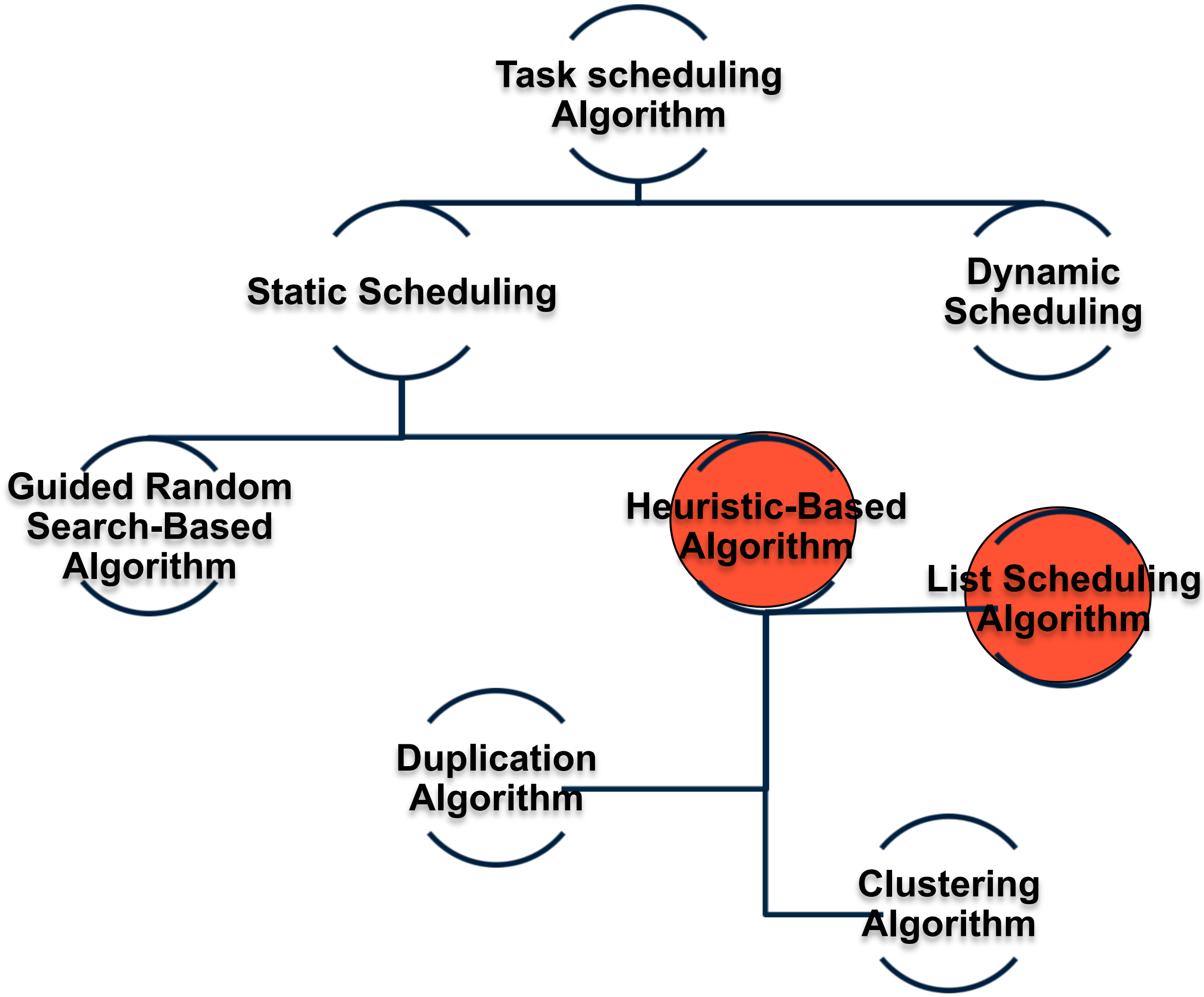
NP-Complete problem!

Common approach:

- **Heuristic based algorithms for heterogeneous systems**

Introduction

Taxonomy of task scheduling



List based algorithms

- *To each task it is **assigned a priority**, and a **list of tasks** is constructed in a decreasing priority order.*

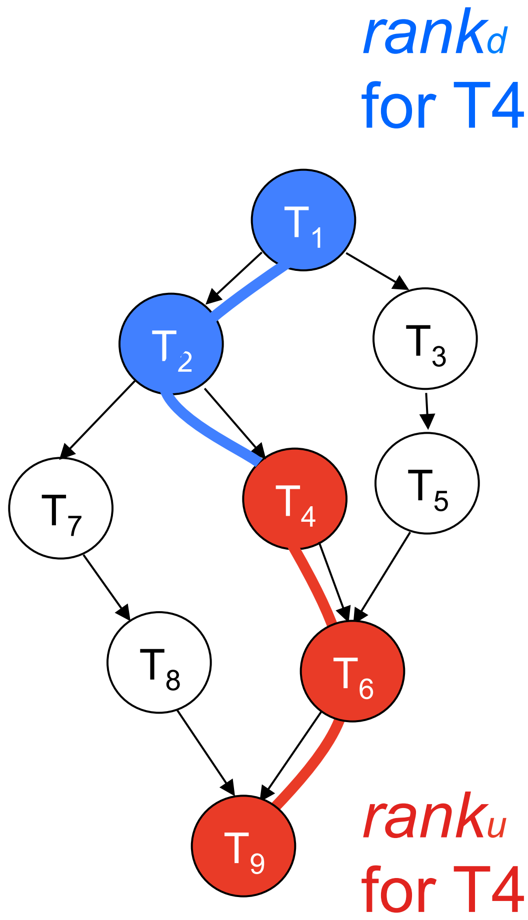
- *A task becomes **ready** for execution when its immediate predecessors in the task graph have already been executed or if it does not have any predecessors.*

- *While there are unscheduled (**ready**) tasks:*
  - *Select the task with higher priority and*
  - *Allocate the task to a processor which allows the earliest start-time (homogeneous case)*

List based algorithms: Definition of Task Priority

- ***Rank downward* of node n_i**
  - Length of the longest path from an entry node to n_i (excluding n_i)
- ***Rank upward* of node n_i**
  - Length of the longest path from n_i to an exit node

The tasks with highest ***rank$_u$*** in the DAG level belong to the Critical Path.

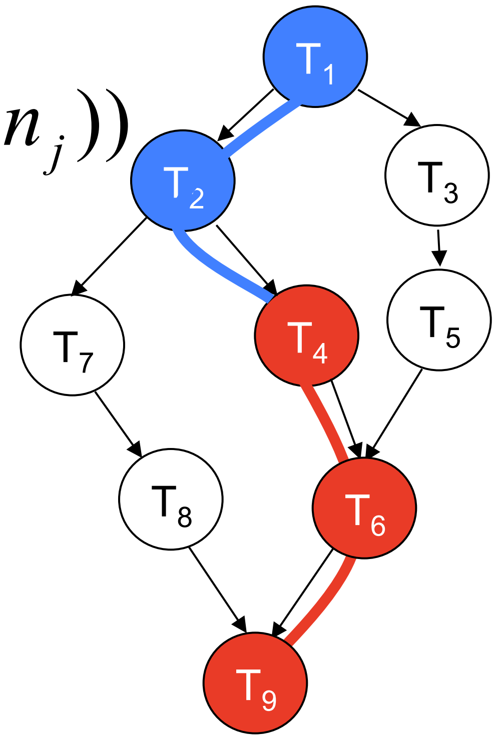


Heterogeneous Earliest Finish Time (HEFT)

- **List scheduling based heuristic**
- **Do a bottom up traversal of the graph and assign ranks to each task**

$$rank_u(n_i) = \overline{w}_i + \max_{n_j \in succ(n_i)} (\overline{c}_{i,j} + rank_u(n_j))$$

$$rank_u(n_{exit}) = \overline{w}_{exit}$$



$$priority(n_i) = rank_u(n_i)$$

(schedules first the CP tasks)

Heterogeneous Earliest Finish Time (HEFT)

Set the computation costs of tasks and communication costs of edges with mean values.
Compute $rank_u$ for all tasks by traversing graph upward, starting from the exit task.
Sort the tasks in a scheduling list by nonincreasing order of $rank_u$ values.
while there are unscheduled tasks in the list **do**
    Select the first task, n_i, from the list for scheduling.
    for each processor p_k in the processor-set ($p_k \in Q$) **do**
        Compute $EFT(n_i, p_k)$ value using the *insertion-based scheduling* policy.
    Assign task n_i to the processor p_j that minimizes EFT of task n_i.
endwhile

- **$EFT(n_i, p_k)$ Earliest execution finish time of task n_i on processor p_k**

Critical Path on a Processor (CPOP)

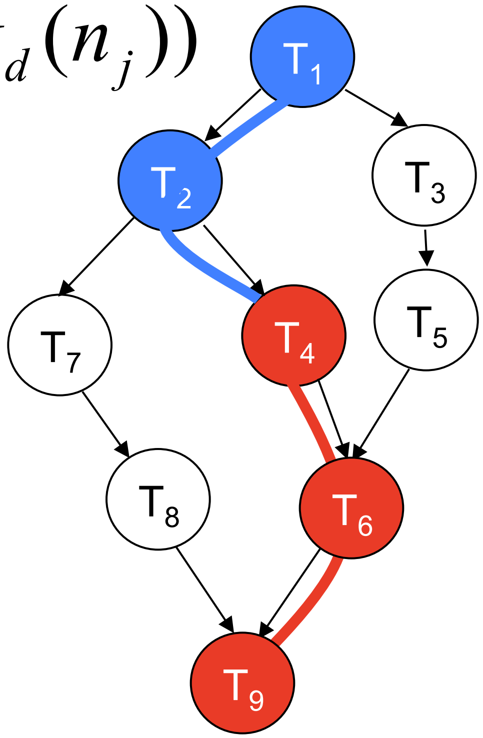
- **Upward ranking**

  …

- **Downward ranking**

$$rank_d(n_i) = \overline{w}_i + \max_{n_j \in pred(n_i)} (\overline{c}_{i,j} + \overline{w}_j + rank_d(n_j))$$

$$rank_d(n_{entry}) = 0$$



$$priority(n_i) = rank_u(n_i) + rank_d(n_i)$$

(schedules first tasks belonging to longer paths)

Critical Path on a Processor (CPOP)

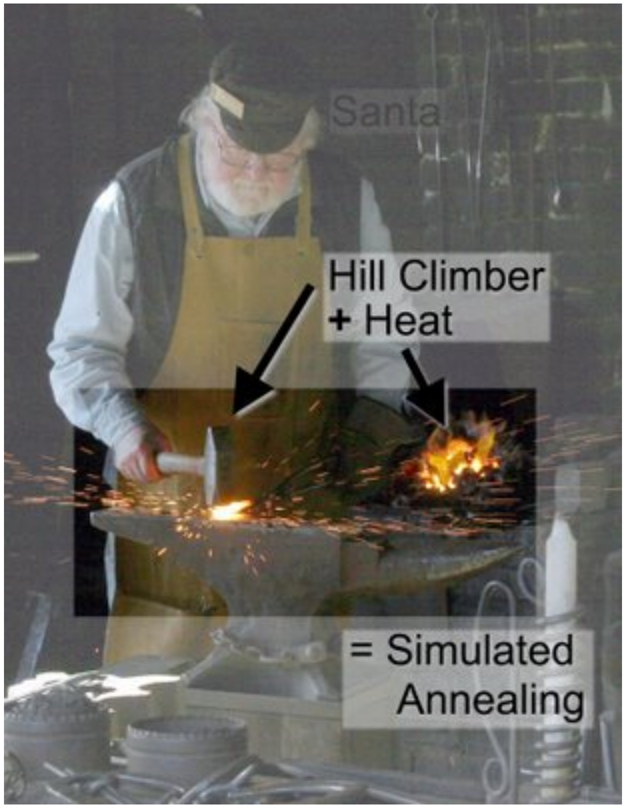
1. Set the computation costs of tasks and communication costs of edges with mean values.
2. Compute $rank_u$ of tasks by traversing graph upward, starting from the exit task.
3. Compute $rank_d$ of tasks by traversing graph downward, starting from the entry task.
4. Compute $priority(n_i) = rank_d(n_i) + rank_u(n_i)$ for each task n_i in the graph.
5. $|CP| = priority(n_{entry})$, where n_{entry} is the *entry* task.
6. $SET_{CP} = \{n_{entry}\}$, where SET_{CP} is the set of tasks on the critical path.
7. $n_k \leftarrow n_{entry}$.
8. **while** n_k is not the exit task **do**
9.   Select n_j where $((n_j \in succ(n_k))$ **and** $(priority(n_j) == |CP|))$.
10. $SET_{CP} = SET_{CP} \bigcup \{n_j\}$.
11. $n_k \leftarrow n_j$.
12. **endwhile**
13. Select the critical-path processor (p_{CP}) which minimizes $\sum_{n_i \in SET_{CP}} w_{i,j}, \quad \forall p_j \in Q$.
14. Initialize the priority queue with the entry task.
15. **while** there is an unscheduled task in the priority queue **do**
16.   Select the highest priority task n_i from priority queue.
17.   **if** $n_i \in SET_{CP}$ **then**
18.   Assign the task n_i on p_{CP}.
19.   **else**
20.   Assign the task n_i to the processor p_j which minimizes the $EFT(n_i, p_j)$.
21.   Update the priority-queue with the successors of n_i, if they become ready tasks.
22. **endwhile**

Identify CP

Select CP processor

Simulated Annealing

- Motivated by the physical annealing process
- Material is heated and slowly cooled into a uniform structure
- Simulated annealing mimics this process
- The first SA algorithm was developed in 1953 (Metropolis)

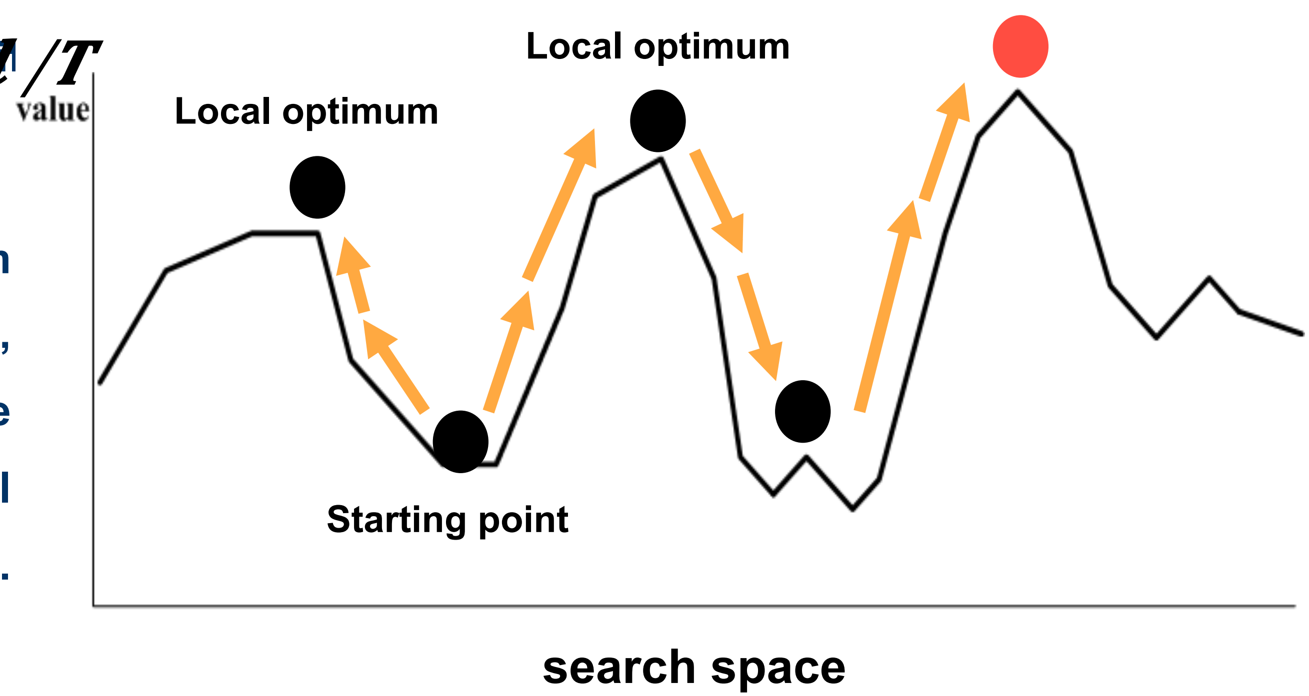


Simulated Annealing

- **Elements of SA**
  - **Representation of the solution**
  - **Evaluation function**
  - **Neighbourhood function**
  - **Neighbourhood search strategy**
  - **Acceptance criterion:**
    - **better moves are always accepted.**
    - **Worse moves are accepted by probability**

Simulated Annealing

The main feature of SA algorithm is the ability to avoid being trapped in local minimum. This is done letting the algorithm to accept not only better solutions but also worse solutions with a given probability. If the current solution (f_{new}) has an objective function value smaller than that of the old solution (f_{old}), then the current solution is accepted. Otherwise, the current solution can also be accepted if $e^{(f_{new}-f_{old})/T}$ is greater than a uniform random number in [0,1], where T is the 'temperature' control parameter.



Tabu Seach

Proposed by Glover (1986) and Hansen (1986):

- **"a meta-heuristic superimposed on another heuristic. The overall approach is to avoid entrapment in cycles by forbidding or penalizing moves which take the solution, in the next iteration, to points in the solution space previously visited (hence *tabu*)."**
- **Accepts non-improving solutions deterministically [no randomness]:**
  - **in order to escape from local optima (where all the neighbouring solutions are non-improving)**

Tabu Seach

- After evaluating a number of neighbourhoods, we accept the best one, even if it has low quality on cost function.
  - Accept worse move
- "tabu list":
  - prevent the search from revisiting previously visited solutions; The aim is to be a global optimizer rather than a local optimizer.
  - explore the unvisited areas of the solution space;

Ant Colony System

Ant Colony System

- First proposed by M. Dorigo, 1992
- Heuristic optimization method inspired by biological systems
- Multi-agent approach for solving difficult combinatorial optimization problems
  - Scheduling, Traveling Salesman, vehicle routing, sequential ordering, graph coloring, routing in communications networks

Ant Colony System

The ants

- Can explore vast areas without global view of the ground
- Can find the food and bring it back to the nest
- Will converge to the shortest path.

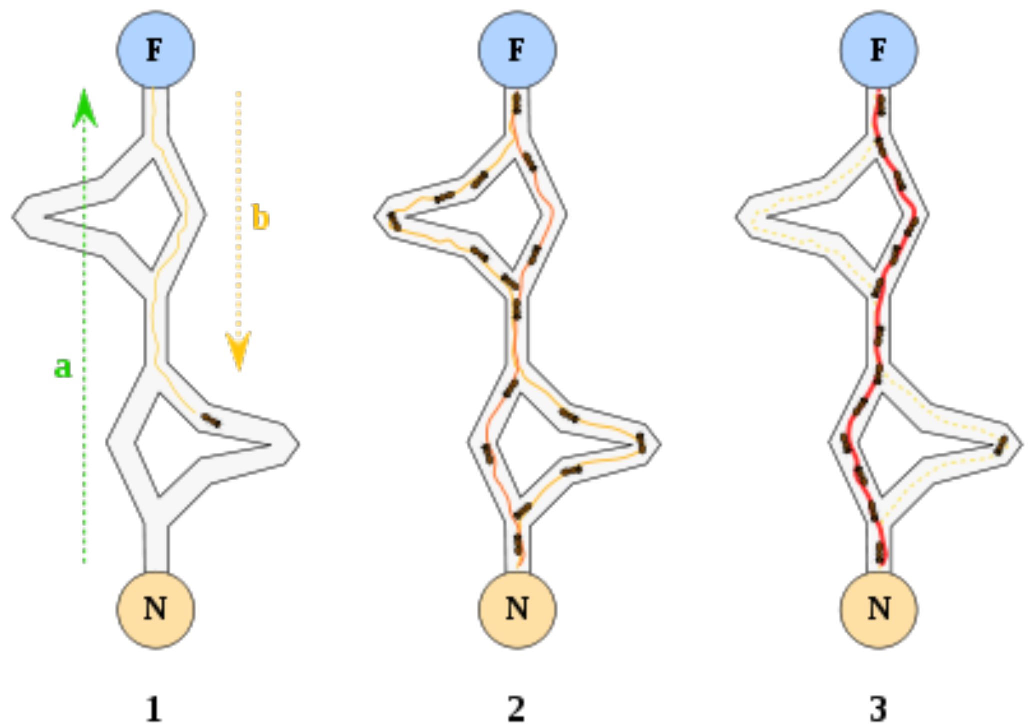
How can they manage such great tasks?

- By leaving pheromone behind them.
- Whatever they go, they let pheromones behind, marking the area as explored and communicating to the other ants that way is known.

Ant Colony System

The original idea comes from observing the exploitation of food resources among ants, in which ants' individually limited cognitive abilities have collectively been able **to find the shortest path between a food source and the nest.**

- The first ant finds the food source **(F)**, via any way **(a)**, then returns to the nest **(N)**, leaving behind a trail pheromone **(b)**
- Ants indiscriminately follow four possible ways, but the strengthening of the runway makes it more attractive as the shortest route.
- Ants take the **shortest route**, long portions of other ways lose their trail pheromones.



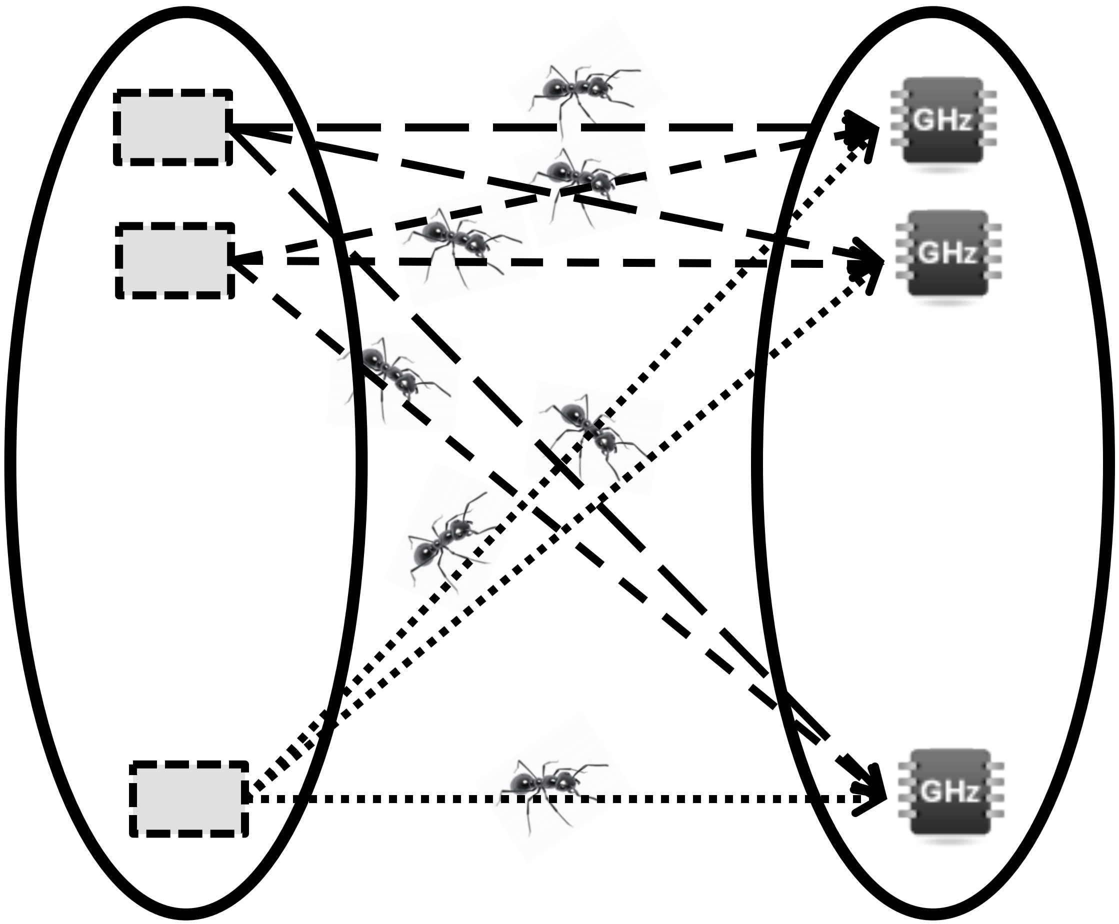
Ant Colony System

Applying ACS to Task Scheduling

- **Ants do not know the global structure of the problem - *discover* the network**
- **Limited ability to sense local environment - can only "see" adjacent nodes of immediate neighborhood.**
- **Each ant chooses an action based on *variable* probability**
  - **random choice**
  - **pheromone mediated**

Ant Colony System

Applying ACS to Task Scheduling



$G\downarrow 1$: TASKs $G\downarrow 2$ PROCESSORs

Ant Colony System

ACS phases:

1. **Initialization of ants:** a set of artificial ants is initially positioned on starting nodes according to some initialization rule.
2. **Solution construction:** Each ant builds a tour by repeatedly applying a stochastic rule based on pheromone and heuristic values using the selection rule of the ACS algorithm.
3. **Local pheromone updating:** each ant while constructing its tour, updates the amount of pheromone on the visited edges by applying the local updating rule.
4. **Global pheromone updating:** After all ants have completed their solutions at the end of each iteration, trails on the edges are modified again by applying the global updating rule.
5. **Final test**: test best solution, if it is not acceptable go to step 2.

Ant Colony System

State transition rule:

Allows to explore other tours

$$prob(i,p)=\{\blacksquare\, max[\tau(i,p)\times[\mu(i,p)]^{\beta}] \qquad if\ q_0 < q\tau(i,p$$

concentrate the search of the system around the best-so-far solution

where q is a random number uniformly distributed in $[0..1]$, q_0 is a parameter ($0\leq q_0 \leq 1$)

Ant Colony System

Local Pheromone Update Rule

During building a solution, each ant by choosing a processor p for task i can changes the pheromone between task i and processor p by applying local update rule

$$\tau(i,p)=(1-\rho)\cdot\tau(i,p)+\rho\cdot\tau_0$$

Where

ρ denotes the pheromone decay parameter

τ_0 is the initial value of pheromone on all edges.

Ant Colony System

Global Pheromone Update Rule

Only the best ants that have the shortest path from source to sink are allowed to deposit pheromone. After all ants finished their tour, we can perform global updating for current iteration. The pheromone level is updated by applying the global updating rule

$$\tau(i,p) = (1-\alpha)\cdot\tau(i,p) + \alpha\cdot\Delta\tau(i,p)$$

$$where\ \Delta\tau(i,p) = \{\blacksquare 1/CP - AFT(n\downarrow exit) \qquad if\ (i,p) \in best\_t$$

$0<\alpha<1$ is the pheromone decay parameter, CP is length of Critical Path and $AFT(n\downarrow exit)$ is makespane of the best ant in current iteration.

Results

- Comparison metrics

- Schedule length ratio

$$SLR = \frac{makespan(solution)}{\sum_{n_i \in CP_{MIN}} \min_{p_j \in P}(w_{(i,j)})}$$

- Speedup

$$Speedup = \frac{\min_{p_j \in P}\left[\sum_{n_i \in V} w_{(i,j)}\right]}{makespan(solution)}$$

Task	P1	P2	P3
T1	14	19	9
T2	13	19	18
T3	11	17	15
T4	13	8	18
T5	12	13	10
T6	12	19	13
T7	7	15	11
T8	5	11	14
T9	18	12	20
T10	17	20	11
	122	153	139

Results

- Input data

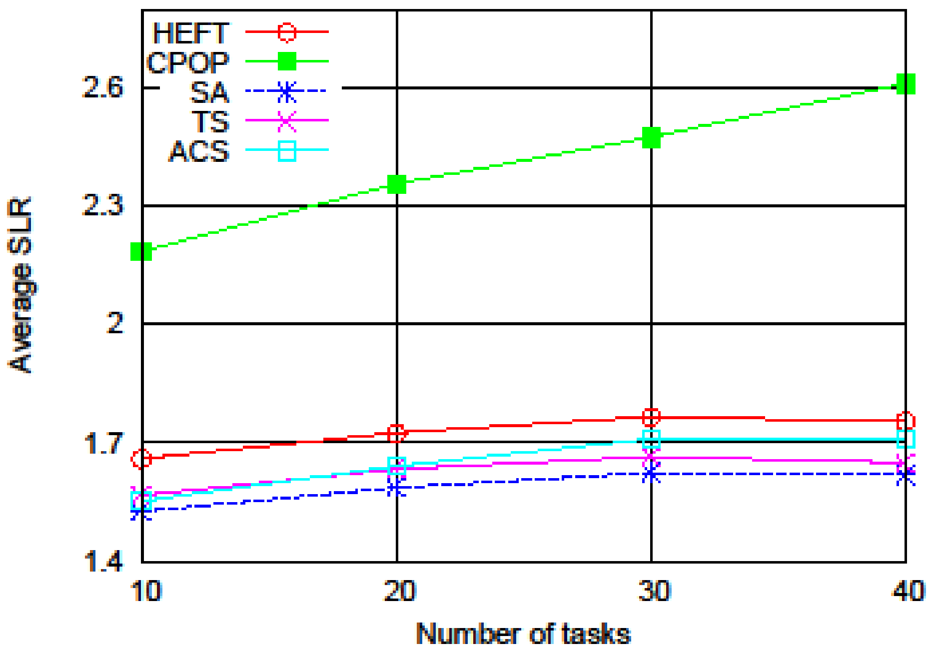
A set of 5760 DAGs were randomly generated using the program available at: *http://www.loria.fr/~suter/dags.html*

DAG generator parameters:
-*Width*: 0.1, 0.2, 0.8.
-*Regularity*: 0.8
-*Density: 0.2, 0.8*
-*Jump: 1, 2, 4*
-Number of tasks: 10, 20, 30 and 40

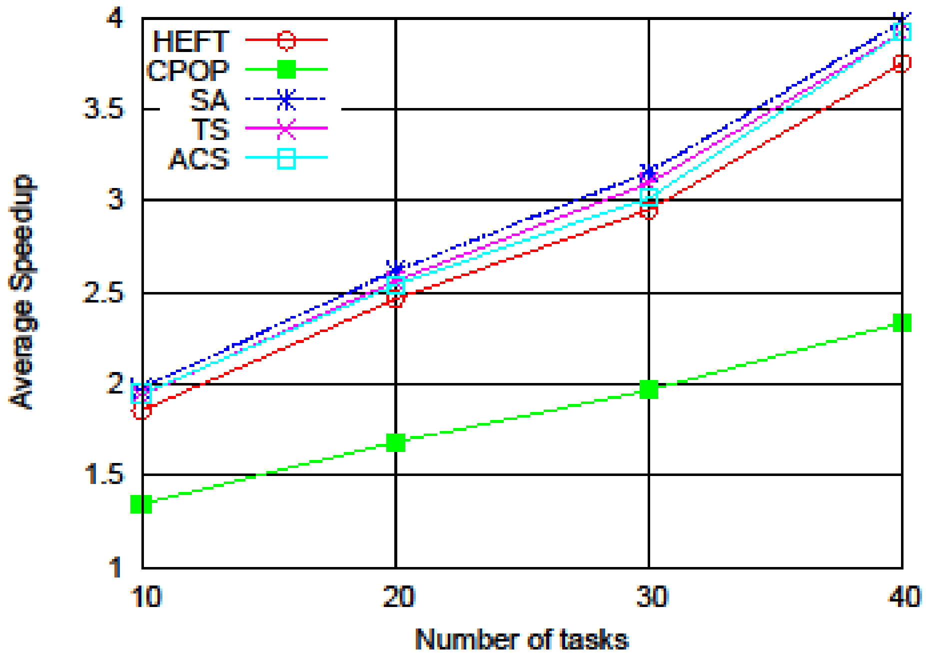
Other parameters:
-*Number of processors:* 4, 8, 16 and 32
-*CCR:* 0.1, 0.5, 0.8 and 1

Results

- Schedule length ratio



- Speedup



- HEFT produces solutions closed to metaheuristic algorithms.
- SA produces the best solutions.

Results

CCR	N=10			N=20			N=30			N=40		
	SA	TS	ACS	SA	TS	ACS	SA	TS	ACS	SA	TS	ACS
0.1	0.80%	0.60%	0.53%	1.64%	1.38%	0.62%	3.16%	2.94%	1.35%	4.07%	3.90%	1.79%
0.5	7.03%	5.70%	5.74%	7.09%	5.66%	4.04%	7.97%	6.50%	2.84%	7.93%	6.74%	2.88%
0.8	9.96%	6.91%	8.27%	10.0%	6.80%	6.45%	9.93%	6.49%	4.09%	9.48%	6.61%	1.87%
1.0	10.0%	6.23%	7.74%	10.2%	5.65%	6.14%	9.45%	5.85%	2.98%	10.9%	6.98%	2.51%

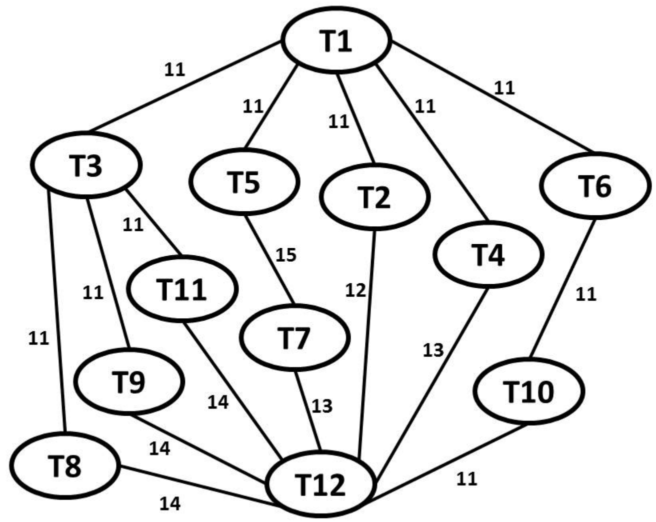
Table 1. SLR improvement observed with metaheuristic algorithms compared to HEFT

- For low CCR and small machine size, the improvement over HEFT is negligible.
- For higher CCRs, up to 1, the improvements achieved with SA are below 11%.

Results

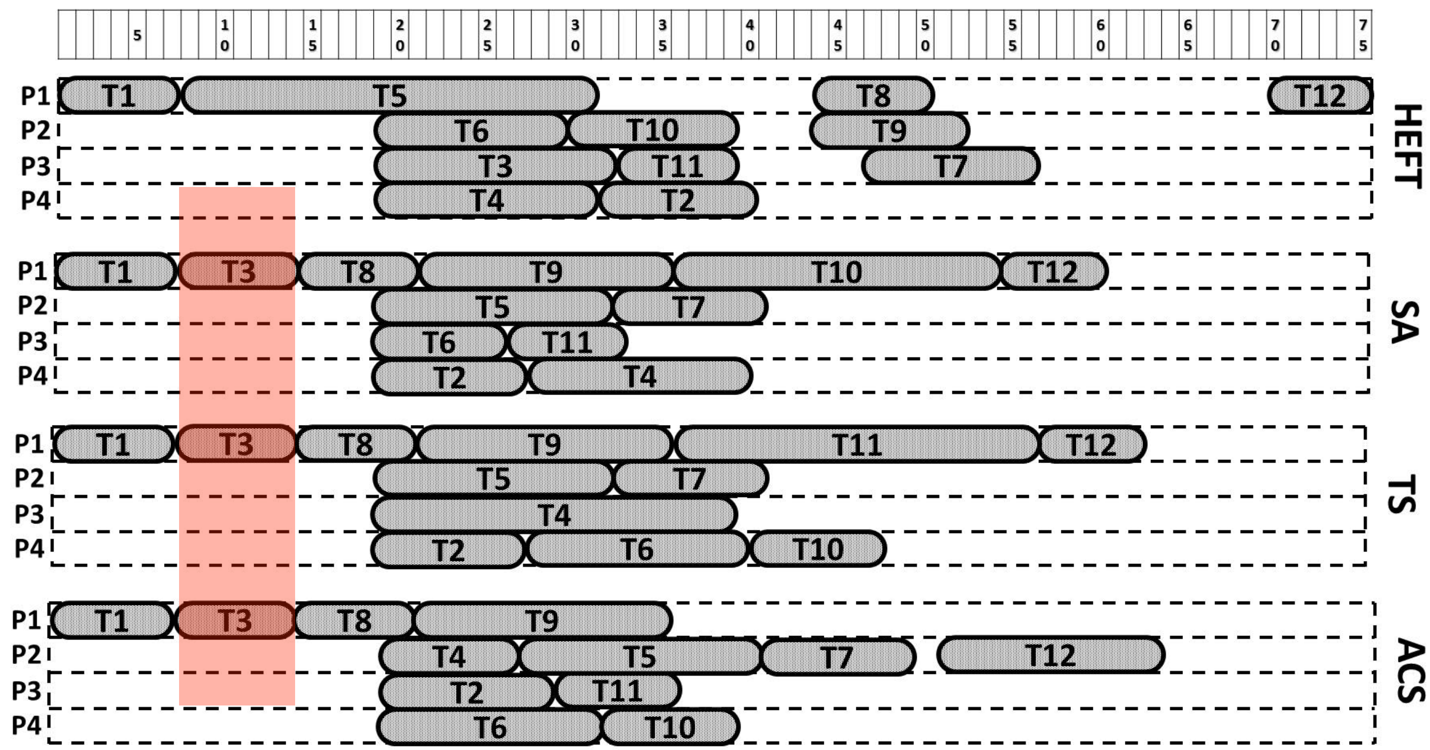
- T5 has higher rank upward
- T5 belongs to the CP

- All meta-heuristics selected first T3, a non CP task!

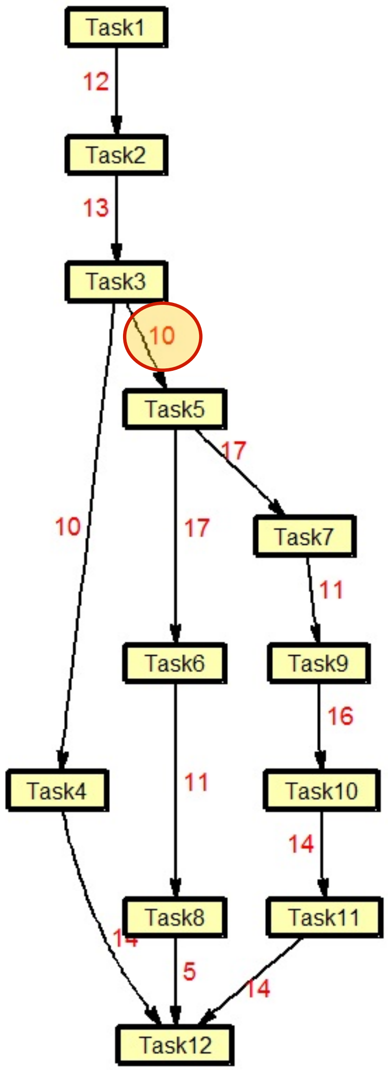


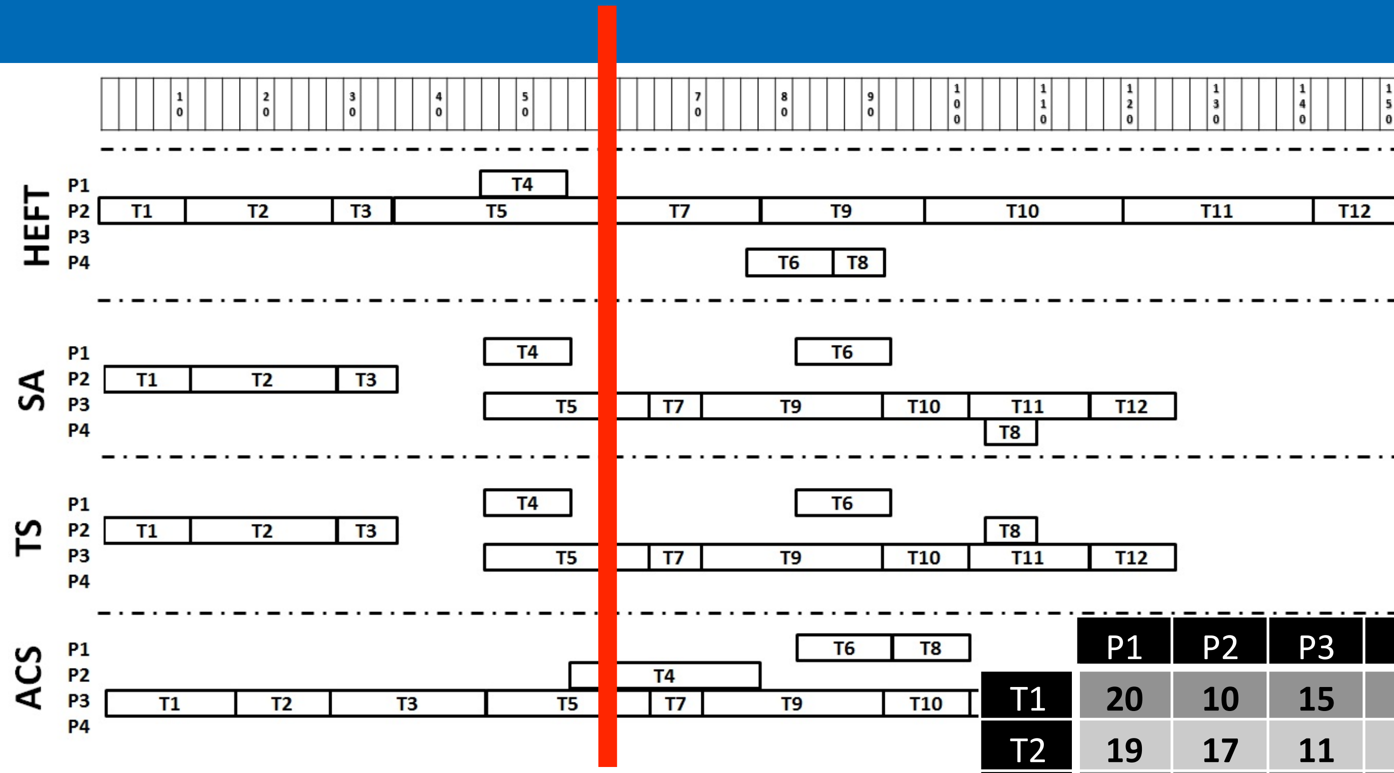
Computation Costs

	P1	P2	P3	P4
T1	7	13	11	23
T2	17	23	10	9
T3	7	20	14	16
T4	24	8	21	13
T5	24	14	21	17
T6	21	11	8	13
T7	23	9	10	24
T8	7	18	19	24
T9	14	9	24	24
T10	19	10	19	8
T11	21	15	7	25
T12	6	13	20	10



Results





- HEFT only considers information from the current level to select processors.

	P1	P2	P3	P4
T1	20	10	15	17
T2	19	17	11	17
T3	12	7	18	14
T4	10	22	19	18
T5	25	24	19	18
T6	11	21	12	10
T7	20	18	6	12
T8	9	6	6	6
T9	21	19	21	17
T10	16	23	10	21
T11	13	22	14	23
T12	15	10	10	19

Conclusions

- **HEFT produces competitive solutions for Low CCRs (0.1).**
- **For higher CCRs, up to 1, the improvements achieved with SA are below 11%.**
  - **HEFT still competitive attending the lower complexity.**
- **Metaheuristics comparison**
  - **SA achieved consistently better scheduling solutions.**
- **Challenges/Future Work:**
  - **To redefine task priority and processor selection, in accordance to the metaheuristic solutions, without increasing (significantly) the time complexity of HEFT.**