Co-Scheduling Algorithms for High-Throughput Workload Execution

Guillaume Aupy, Manu Shantharam, Anne Benoit,
Yves Robert & Padma Raghavan

Scheduling in Aussois 2014

Applications

p identical processors
n moldable applications, or tasks, T_i, s.t.,

- p_i is the minimal number of processors needed for T_i;
- $t_{i,j}$ is the execution time of task T_i with j processors;
- $work(i,j) = j \times t_{i,j}$ is the corresponding work.

We assume the following for $1 \leq i \leq n$ and $p_i \leq j < p$:

Non increasing execution time: $t_{i,j+1} \leq t_{i,j}$
Non decreasing work: $work(i, j+1) \geq work(i,j)$

Co-schedules

A co-schedule partitions the n tasks into groups (called *packs*), so that (i) all tasks from a given pack start their execution at the same time; and (ii) two tasks from different packs have disjoint execution intervals.

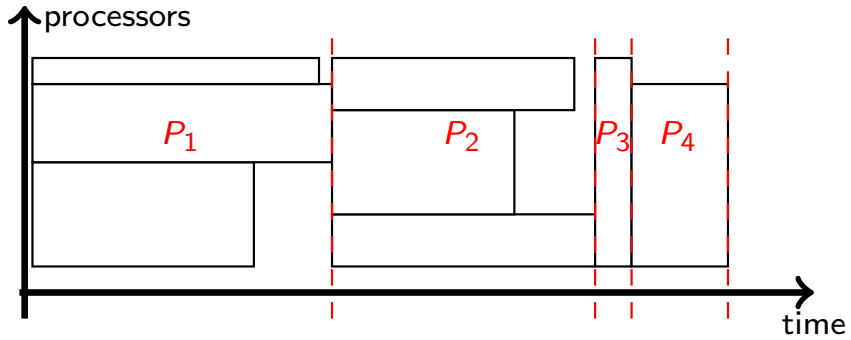


Figure : A co-schedule with four packs P_1 to P_4.

Definition (k-IN-p-CoSchedule optimization problem)

Given a fixed constant $k \leq p$, find a co-schedule with at most k tasks per pack that minimizes the execution time.

The most general problem is when $k = p$, but in some frameworks we may have an upper bound $k < p$ on the maximum number of tasks within each pack.

Related work

Performance bounds for level-oriented two-dimensional packing algorithms, Coffman, Garey, Johnson:
Strip-packing problem, parallel tasks (fixed number of processors), approx-algo based on "shelves".

Scheduling parallel tasks: Approximation algorithms, Dutot, Mounié, Trystram:
Use this model to approximate the moldable model; they studied the p-IN-p-CoSchedule for identical moldable tasks (poly with DP).

Widely studied for sequential tasks.

Polynomial instances

Theorem

The 1-IN-*p*-CoSchedule *and* 2-IN-*p*-CoSchedule *problems can both be solved in polynomial time.*

Polynomial instances

Theorem

The 1-IN-p-COSCHEDULE *and* 2-IN-p-COSCHEDULE *problems can both be solved in polynomial time.*

Proof.

If there is a pack with exactly task T_i and T_j, then its execution time is $\min_{j=1..p} \left(\max(t_{i,p-j}, t_{i',j})\right)$.

□

Polynomial instances

Theorem

The 1-IN-p-CoSchedule *and* 2-IN-p-CoSchedule *problems can both be solved in polynomial time.*

Proof.

If there is a pack with exactly task T_i and T_j, then its execution time is $\min_{j=1..p}\left(\max(t_{i,p-j}, t_{i',j})\right)$.

We then construct the complete weighted graph $G = (V, E)$, where $|V| = n$, and

$$e_{i,j} = \begin{cases} t_{i,p} & \text{if } i = j \\ \min_{j=1..p}\left(\max(t_{i,p-j}, t_{i',j})\right) & \text{otherwise} \end{cases}$$

□

Polynomial instances

Theorem

The 1-IN-p-COSCHEDULE *and* 2-IN-p-COSCHEDULE *problems can both be solved in polynomial time.*

Proof.

If there is a pack with exactly task T_i and T_j, then its execution time is $\min_{j=1..p}\left(\max(t_{i,p-j}, t_{i',j})\right)$.

We then construct the complete weighted graph $G = (V, E)$, where $|V| = n$, and

$$e_{i,j} = \begin{cases} t_{i,p} & \text{if } i = j \\ \min_{j=1..p}\left(\max(t_{i,p-j}, t_{i',j})\right) & \text{otherwise} \end{cases}$$

Finally, solving MIN-WEIGHT-PERFECT-MATCHING on G gives the best solution for 2-IN-p-COSCHEDULE. □

Polynomial instances

Theorem

Given k tasks to be scheduled on p processors as one pack, we can find in time $O(p \log k)$ the schedule that minimizes the cost of the pack.

Polynomial instances

Theorem

Given k tasks to be scheduled on p processors as one pack, we can find in time $O(p \log k)$ the schedule that minimizes the cost of the pack.

The greedy algorithm that, while there remains available processors, assigns one to the largest task (with their current processor assignement), returns an optimal solution. We call this procedure Optimal-1-pack-schedule

Intractability

Theorem

The 3-IN-p-COSCHEDULE *problem is strongly* $\mathcal{NP}$*-complete.*

Intractability

Theorem

The 3-IN-p-COSCHEDULE *problem is strongly* $\mathcal{NP}$*-complete.*

Proof.

We reduce this problem to 3-PARTITION: Given an integer B and $3n$ integers $a_1, \ldots, a_{3n}$, can we partition the $3n$ integers into n triplets, each of sum B? This problem is strongly $\mathcal{NP}$-hard so we can encode the a_i's and B in unary.

Intractability

Theorem

The 3-IN-p-CoSchedule problem is strongly $\mathcal{NP}$-complete.

Proof.

We reduce this problem to 3-Partition: Given an integer B and $3n$ integers $a_1, \ldots, a_{3n}$, can we partition the $3n$ integers into n triplets, each of sum B? This problem is strongly $\mathcal{NP}$-hard so we can encode the a_i's and B in unary.

We build the following instance $\mathcal{I}_2$ of 3-IN-p-CoSchedule:

- $p = B$ processors
- a deadline $D = n$
- $3n$ tasks T_i such that $t_{i,j} = 1 + \frac{1}{a_i}$ if $j < a_i$, $t_{i,j} = 1$ otherwise.

(See that the $t_{i,j}$'s verify the constraints on the work and execution time.)

Intractability

Theorem

The 3-IN-p-CoSchedule *problem is strongly* $\mathcal{NP}$*-complete.*

Proof.

We reduce this problem to 3-Partition: Given an integer B and $3n$ integers $a_1, \ldots, a_{3n}$, can we partition the $3n$ integers into n triplets, each of sum B? This problem is strongly $\mathcal{NP}$-hard so we can encode the a_i's and B in unary.

We build the following instance $\mathcal{I}_2$ of 3-IN-p-CoSchedule:

- $p = B$ processors
- a deadline $D = n$
- $3n$ tasks T_i such that $t_{i,j} = 1 + \frac{1}{a_i}$ if $j < a_i$, $t_{i,j} = 1$ otherwise.

Then necessarily any solution of $\mathcal{I}_2$ has n packs each of size 1 with exactly 3 tasks in it, and the sum of the weight of those tasks sums up to B.

□

Theorem

For $k \geq 3$, The k-IN-p-CoSchedule problem is strongly $\mathcal{NP}$-complete.

Theorem

For $k \geq 3$, The k-IN-p-CoSchedule problem is strongly $\mathcal{NP}$-complete.

We reduce those problems to the same instance of the 3-IN-p-CoSchedule problem, to which we further add:

- $n(k-3)$ buffer tasks such that $t_{i,j} = \max\left(\frac{B+1}{j}, 1\right)$;
- the number of processors is now $p = B + (k-3)(B+1)$;
- the deadline remains $D = n$.

Again, we need to execute each pack in unit time and at most n packs. The only mean to do so is to execute within each pack $k-3$ buffer tasks on $B+1$ processors.

Open problems?

k-IN-p-CoSchedule for $k \geq 3$ is $\mathcal{NP}$-complete, 2-IN-p-CoSchedule $\in \mathcal{P}$. What about 3-IN-3-CoSchedule?

Some ideas:

- Obvious solution to "exactly-3-IN-3-CoSchedule", however "exactly-3-IN-p-CoSchedule" is $\mathcal{NP}$-complete
- DP to solve the 3-IN-3-CoSchedule problem where a batch can have 1 or 3 elements (but not two).

Chances are, it is solvable in polynomial time.

Optimal solution

Theorem

The following integer linear program characterizes the k-IN-p-CoSchedule problem, where the unknown variables are the $x_{i,j,b}$'s (Boolean variables) and the y_b's (rational variables), for $1 \leq i, b \leq n$ and $1 \leq j \leq p$:

$$
\begin{array}{lll}
\text{Minimize} \sum_{b=1}^{n} y_b & \text{subject to} & \\
(i)\ \sum_{j,b} x_{i,j,b} = 1, & 1 \leq i \leq n & \\
(ii)\ \sum_{i,j} x_{i,j,b} \leq k, & 1 \leq b \leq n & (1) \\
(iii)\ \sum_{i,j} j \cdot x_{i,j,b} \leq p, & 1 \leq b \leq n & \\
(iv)\ x_{i,j,b} \cdot t_{i,j} \leq y_b, & 1 \leq i, b \leq n, 1 \leq j \leq p &
\end{array}
$$

Approximation Results

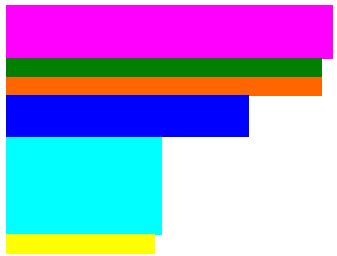
We propose an algorithm that is a 3-approximation algorithm for the problem p-IN-p-CoSchedule.

procedure PACK-APPROX($T_1, \ldots, T_n$)
begin
  COST $= +\infty$;
  for $j = 1$ *to* n **do** $\sigma(j) \leftarrow p_j$;
  for $i = 0$ *to* $\sum_j (p - p_j) - 1$ **do**
    Call MAKE-PACK (n, p, p, σ);
    Let COST$_i$ be the cost of the co-schedule;
    if $COST_i < COST_{\text{OPT}}$ **then** COST $\leftarrow$ COST$_i$;
    Let $A_{\text{tot}}(i) = \sum_{j=1}^{n} t_{j,\sigma(j)}\sigma(j)$;
    Let $T_{j^\star}$ be one task that maximizes $t_{j,\sigma(j)}$;
    if $\left(A_{tot}(i) > p \cdot t_{j^\star,\sigma(j^\star)}\right)$ *or* $(\sigma(j^\star) = p)$ **then**
      return *COST*
    else
      $\sigma(j^\star) \leftarrow \sigma(j^\star) + 1$
    end
  end
  return *COST;*
end

Approximation Results

We propose an algorithm that is a 3-approximation algorithm for the problem p-IN-p-CoSchedule.

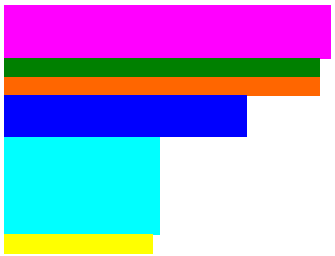
procedure PACK-APPROX($T_1, \ldots, T_n$)
begin
  COST $= +\infty$;
  for $j = 1$ *to* n **do** $\sigma(j) \leftarrow p_j$;
  for $i = 0$ *to* $\sum_j (p - p_j) - 1$ **do**
    Call MAKE-PACK (n, p, p, σ);
    Let COST$_i$ be the cost of the co-schedule;
    if $COST_i < COST_{\text{OPT}}$ **then** COST $\leftarrow$ COST$_i$;
    Let $A_{\text{tot}}(i) = \sum_{j=1}^{n} t_{j,\sigma(j)}\sigma(j)$;
    Let $T_{j^\star}$ be one task that maximizes $t_{j,\sigma(j)}$;
    if $\left(A_{tot}(i) > p \cdot t_{j^\star,\sigma(j^\star)}\right)$ *or* $(\sigma(j^\star) = p)$ **then**
      return *COST*
    else
      $\sigma(j^\star) \leftarrow \sigma(j^\star) + 1$
    end
  end
  return *COST*;
end

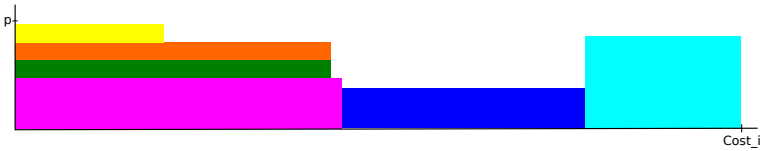


Approximation Results

We propose an algorithm that is a 3-approximation algorithm for the problem p-IN-p-CoSchedule.

procedure PACK-APPROX($T_1, \ldots, T_n$)
begin
  COST $= +\infty$;
  for $j = 1$ *to* n **do** $\sigma(j) \leftarrow p_j$;
  for $i = 0$ *to* $\sum_j (p - p_j) - 1$ **do**
    Call MAKE-PACK (n, p, p, σ);
    Let COST$_i$ be the cost of the co-schedule;
    if $COST_i < COST_{\text{OPT}}$ **then** COST $\leftarrow$ COST$_i$;
    Let $A_{\text{tot}}(i) = \sum_{j=1}^{n} t_{j,\sigma(j)}\sigma(j)$;
    Let $T_{j^\star}$ be one task that maximizes $t_{j,\sigma(j)}$;
    if $\left(A_{tot}(i) > p \cdot t_{j^\star,\sigma(j^\star)}\right)$ *or* $(\sigma(j^\star) = p)$ **then**
      return *COST*
    else
      $\sigma(j^\star) \leftarrow \sigma(j^\star) + 1$
    end
  end
  return *COST*;
end

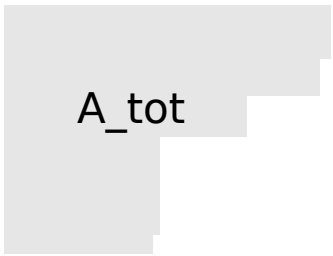


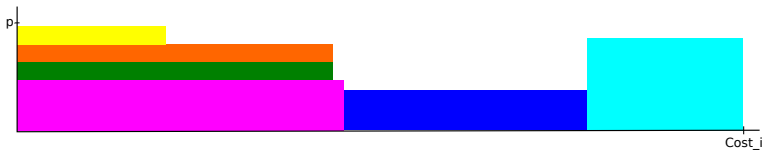


Approximation Results

We propose an algorithm that is a 3-approximation algorithm for the problem p-IN-p-CoSchedule.

procedure PACK-APPROX($T_1, \ldots, T_n$)
begin
  COST $= +\infty$;
  for $j = 1$ *to* n **do** $\sigma(j) \leftarrow p_j$;
  for $i = 0$ *to* $\sum_j (p - p_j) - 1$ **do**
    Call MAKE-PACK (n, p, p, σ);
    Let COST$_i$ be the cost of the co-schedule;
    if $COST_i < COST_{\text{OPT}}$ **then** COST $\leftarrow$ COST$_i$;
    Let $A_{\text{tot}}(i) = \sum_{j=1}^{n} t_{j,\sigma(j)}\sigma(j)$;
    Let $T_{j^\star}$ be one task that maximizes $t_{j,\sigma(j)}$;
    if $\left(A_{tot}(i) > p \cdot t_{j^\star,\sigma(j^\star)}\right)$ *or* $(\sigma(j^\star) = p)$ **then**
      return *COST*
    else
      $\sigma(j^\star) \leftarrow \sigma(j^\star) + 1$
    end
  end
  return *COST*;
end

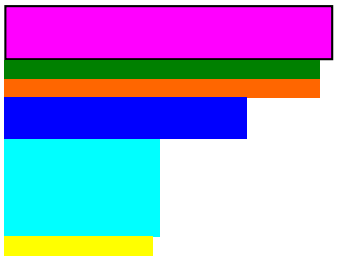


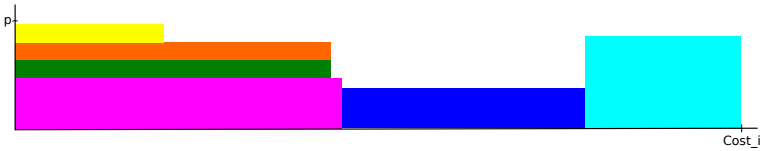


Approximation Results

We propose an algorithm that is a 3-approximation algorithm for the problem p-IN-p-CoSchedule.

procedure PACK-APPROX($T_1, \ldots, T_n$)
begin
  COST $= +\infty$;
  for $j = 1$ *to* n **do** $\sigma(j) \leftarrow p_j$;
  for $i = 0$ *to* $\sum_j (p - p_j) - 1$ **do**
    Call MAKE-PACK (n, p, p, σ);
    Let COST$_i$ be the cost of the co-schedule;
    if $COST_i < COST_{\text{OPT}}$ **then** COST $\leftarrow$ COST$_i$;
    Let $A_{\text{tot}}(i) = \sum_{j=1}^{n} t_{j,\sigma(j)}\sigma(j)$;
    Let $T_{j^\star}$ be one task that maximizes $t_{j,\sigma(j)}$;
    if $\left(A_{tot}(i) > p \cdot t_{j^\star,\sigma(j^\star)}\right)$ *or* $(\sigma(j^\star) = p)$ **then**
      return *COST*
    else
      $\sigma(j^\star) \leftarrow \sigma(j^\star) + 1$
    end
  end
  return *COST*;
end

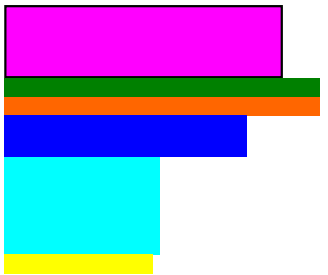


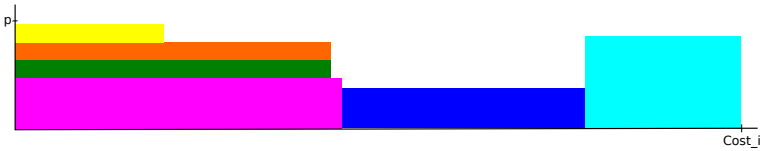


Approximation Results

We propose an algorithm that is a 3-approximation algorithm for the problem p-IN-p-CoSchedule.

procedure PACK-APPROX($T_1, \ldots, T_n$)
begin
  COST $= +\infty$;
  for $j = 1$ *to* n **do** $\sigma(j) \leftarrow p_j$;
  for $i = 0$ *to* $\sum_j (p - p_j) - 1$ **do**
    Call MAKE-PACK (n, p, p, σ);
    Let COST$_i$ be the cost of the co-schedule;
    if $COST_i < COST_{\text{OPT}}$ **then** COST $\leftarrow$ COST$_i$;
    Let $A_{\text{tot}}(i) = \sum_{j=1}^{n} t_{j,\sigma(j)}\sigma(j)$;
    Let $T_{j^\star}$ be one task that maximizes $t_{j,\sigma(j)}$;
    if $\left(A_{tot}(i) > p \cdot t_{j^\star,\sigma(j^\star)}\right)$ *or* $(\sigma(j^\star) = p)$ **then**
      return *COST*
    else
      $\sigma(j^\star) \leftarrow \sigma(j^\star) + 1$
    end
  end
  return *COST*;
end

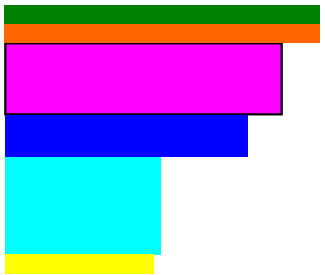


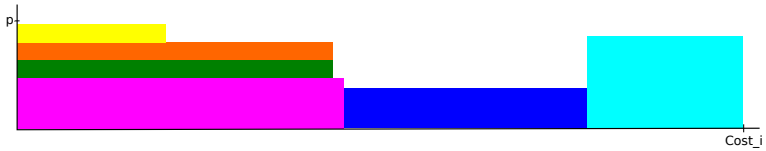


Approximation Results

We propose an algorithm that is a 3-approximation algorithm for the problem p-IN-p-CoSchedule.

procedure PACK-APPROX($T_1, \ldots, T_n$)
begin
  COST $= +\infty$;
  for $j = 1$ *to* n **do** $\sigma(j) \leftarrow p_j$;
  for $i = 0$ *to* $\sum_j (p - p_j) - 1$ **do**
    Call MAKE-PACK (n, p, p, σ);
    Let COST$_i$ be the cost of the co-schedule;
    if $COST_i < COST_{\text{OPT}}$ **then** COST $\leftarrow$ COST$_i$;
    Let $A_{\text{tot}}(i) = \sum_{j=1}^{n} t_{j,\sigma(j)}\sigma(j)$;
    Let $T_{j^\star}$ be one task that maximizes $t_{j,\sigma(j)}$;
    if $\left(A_{tot}(i) > p \cdot t_{j^\star,\sigma(j^\star)}\right)$ *or* $(\sigma(j^\star) = p)$ **then**
      return *COST*
    else
      $\sigma(j^\star) \leftarrow \sigma(j^\star) + 1$
    end
  end
  return *COST*;
end

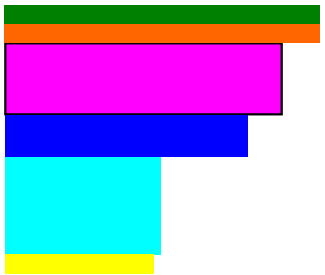


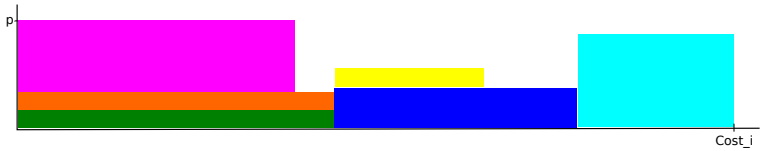


Approximation Results

We propose an algorithm that is a 3-approximation algorithm for the problem p-IN-p-CoSchedule.

procedure PACK-APPROX($T_1, \ldots, T_n$)
begin
  COST $= +\infty$;
  for $j = 1$ *to* n **do** $\sigma(j) \leftarrow p_j$;
  for $i = 0$ *to* $\sum_j (p - p_j) - 1$ **do**
    Call MAKE-PACK (n, p, p, σ);
    Let COST$_i$ be the cost of the co-schedule;
    if $COST_i < COST_{\text{OPT}}$ **then** COST $\leftarrow$ COST$_i$;
    Let $A_{\text{tot}}(i) = \sum_{j=1}^{n} t_{j,\sigma(j)}\sigma(j)$;
    Let $T_{j^\star}$ be one task that maximizes $t_{j,\sigma(j)}$;
    if $\left(A_{tot}(i) > p \cdot t_{j^\star,\sigma(j^\star)}\right)$ *or* $(\sigma(j^\star) = p)$ **then**
      return *COST*
    else
      $\sigma(j^\star) \leftarrow \sigma(j^\star) + 1$
    end
  end
  return *COST*;
end

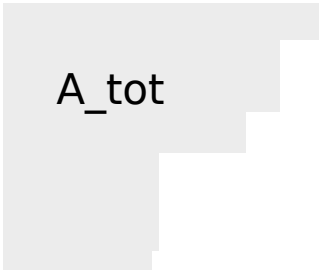


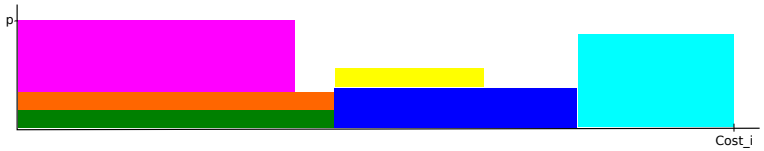


Approximation Results

We propose an algorithm that is a 3-approximation algorithm for the problem p-IN-p-CoSchedule.

procedure PACK-APPROX($T_1, \ldots, T_n$)
begin
  COST $= +\infty$;
  for $j = 1$ *to* n **do** $\sigma(j) \leftarrow p_j$;
  for $i = 0$ *to* $\sum_j (p - p_j) - 1$ **do**
    Call MAKE-PACK (n, p, p, σ);
    Let COST$_i$ be the cost of the co-schedule;
    if $COST_i < COST_{\text{OPT}}$ **then** COST $\leftarrow$ COST$_i$;
    Let $A_{\text{tot}}(i) = \sum_{j=1}^{n} t_{j,\sigma(j)}\sigma(j)$;
    Let $T_{j^\star}$ be one task that maximizes $t_{j,\sigma(j)}$;
    if $\left(A_{tot}(i) > p \cdot t_{j^\star,\sigma(j^\star)}\right)$ *or* $(\sigma(j^\star) = p)$ **then**
      return *COST*
    else
      $\sigma(j^\star) \leftarrow \sigma(j^\star) + 1$
    end
  end
  return *COST*;
end

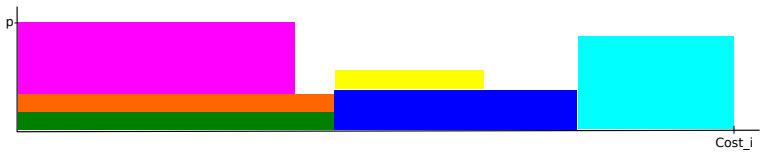




Approximation Results

We propose an algorithm that is a 3-approximation algorithm for the problem p-IN-p-CoSchedule.

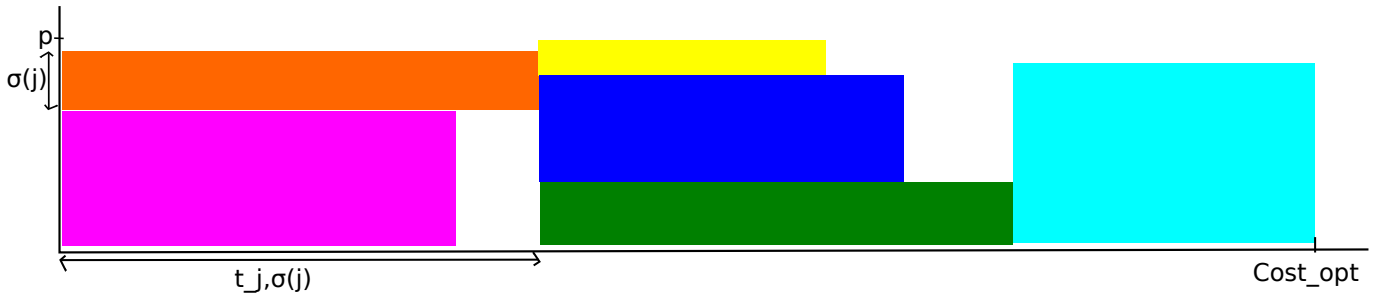
procedure PACK-Approx($T_1, \ldots, T_n$)
begin
  COST $= +\infty$;
  for $j = 1$ *to* n **do** $\sigma(j) \leftarrow p_j$;
  for $i = 0$ *to* $\sum_j (p - p_j) - 1$ **do**
    Call Make-pack (n, p, p, σ);
    Let COST$_i$ be the cost of the co-schedule;
    if $COST_i < COST_{\text{OPT}}$ **then** COST $\leftarrow$ COST$_i$;
    Let $A_{\text{tot}}(i) = \sum_{j=1}^{n} t_{j,\sigma(j)}\sigma(j)$;
    Let $T_{j^\star}$ be one task that maximizes $t_{j,\sigma(j)}$;
    if $\left(A_{tot}(i) > p \cdot t_{j^\star,\sigma(j^\star)}\right)$ *or* $(\sigma(j^\star) = p)$ **then**
      return *COST*
    else
      $\sigma(j^\star) \leftarrow \sigma(j^\star) + 1$
    end
  end
  return *COST*;
end



Analysis

A couple of important points:

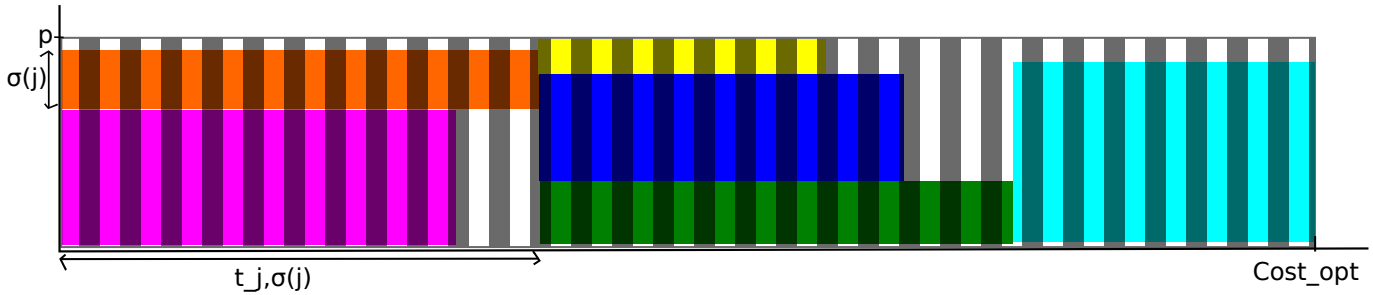
- Given an optimal solution OPT , $\forall j, t_{j,\sigma^{(\mathrm{OPT})}(j)} \leq \mathrm{COST}_{\mathrm{OPT}}$ and $A_{\mathrm{OPT}} \leq p\mathrm{COST}_{\mathrm{OPT}}$.



Analysis

A couple of important points:

- Given an optimal solution OPT , $\forall j, t_{j,\sigma^{(\mathrm{OPT})}(j)} \leq \mathrm{COST}_{\mathrm{OPT}}$ and $A_{\mathrm{OPT}} \leq p\mathrm{COST}_{\mathrm{OPT}}$.



Analysis

A couple of important points:

- Given an optimal solution OPT , $\forall j, t_{j,\sigma^{(\mathrm{OPT})}(j)} \leq \mathrm{COST}_{\mathrm{OPT}}$ and $A_{\mathrm{OPT}} \leq p\mathrm{COST}_{\mathrm{OPT}}$.
- At each step i, $A_{\mathrm{tot}}(i+1) \geq A_{\mathrm{tot}}(i)$ and $t_{\mathrm{max}}(i+1) \leq t_{\mathrm{max}}(i)$. Remember, $\forall i, j, work(i, j+1) \geq work(i, j)$

Analysis

A couple of important points:

- Given an optimal solution OPT , $\forall j, t_{j,\sigma^{(\mathrm{OPT})}(j)} \leq \mathrm{COST}_{\mathrm{OPT}}$ and $A_{\mathrm{OPT}} \leq p\mathrm{COST}_{\mathrm{OPT}}$.
- At each step i, $A_{\mathrm{tot}}(i+1) \geq A_{\mathrm{tot}}(i)$ and $t_{\mathrm{max}}(i+1) \leq t_{\mathrm{max}}(i)$.

- For any step i such that $t_{\mathrm{max}}(i) > \mathrm{COST}_{\mathrm{OPT}}$, then $\forall j, \sigma^{(i)}(j) \leq \sigma^{(\mathrm{OPT})}(j)$, and $A_{\mathrm{tot}}(i) \leq A_{\mathrm{OPT}}$.
  $\sigma^{(i)}(j) = p_j$ or $t_{j,\sigma^{(i)}(j)-1} \geq t_{\mathrm{max}}(i) > \mathrm{COST}_{\mathrm{OPT}} \geq t_{j,\sigma^{(\mathrm{OPT})}(j)}$

Analysis

A couple of important points:

- Given an optimal solution OPT , $\forall j, t_{j,\sigma^{(\mathrm{OPT})}(j)} \leq \mathrm{COST}_{\mathrm{OPT}}$ and $A_{\mathrm{OPT}} \leq p\mathrm{COST}_{\mathrm{OPT}}$.
- At each step i, $A_{\mathrm{tot}}(i+1) \geq A_{\mathrm{tot}}(i)$ and $t_{\mathrm{max}}(i+1) \leq t_{\mathrm{max}}(i)$.
- For any step i such that $t_{\mathrm{max}}(i) > \mathrm{COST}_{\mathrm{OPT}}$, then $\forall j, \sigma^{(i)}(j) \leq \sigma^{(\mathrm{OPT})}(j)$, and $A_{\mathrm{tot}}(i) \leq A_{\mathrm{OPT}}$.
- For any step i such that $t_{\mathrm{max}}(i) > \mathrm{COST}_{\mathrm{OPT}}$, then $A_{\mathrm{tot}}(i) \leq p \cdot t_{\mathrm{max}}(i)$.

Analysis

A couple of important points:

- Given an optimal solution OPT , $\forall j, t_{j,\sigma^{(\mathrm{OPT})}(j)} \leq \mathrm{COST}_{\mathrm{OPT}}$ and $A_{\mathrm{OPT}} \leq p\mathrm{COST}_{\mathrm{OPT}}$.
- At each step i, $A_{\mathrm{tot}}(i+1) \geq A_{\mathrm{tot}}(i)$ and $t_{\max}(i+1) \leq t_{\max}(i)$.
- For any step i such that $t_{\max}(i) > \mathrm{COST}_{\mathrm{OPT}}$, then $\forall j, \sigma^{(i)}(j) \leq \sigma^{(\mathrm{OPT})}(j)$, and $A_{\mathrm{tot}}(i) \leq A_{\mathrm{OPT}}$.
- For any step i such that $t_{\max}(i) > \mathrm{COST}_{\mathrm{OPT}}$, then $A_{\mathrm{tot}}(i) \leq p \cdot t_{\max}(i)$.
- There exists $i_0 \geq 0$ such that $t_{\max}(i_0 - 1) > \mathrm{COST}_{\mathrm{OPT}} \geq t_{\max}(i_0)$.

Analysis

A couple of important points:

- Given an optimal solution OPT , $\forall j, t_{j,\sigma^{(\mathrm{OPT})}(j)} \leq \mathrm{COST}_{\mathrm{OPT}}$ and $A_{\mathrm{OPT}} \leq p\mathrm{COST}_{\mathrm{OPT}}$.
- At each step i, $A_{\mathrm{tot}}(i+1) \geq A_{\mathrm{tot}}(i)$ and $t_{\max}(i+1) \leq t_{\max}(i)$.
- For any step i such that $t_{\max}(i) > \mathrm{COST}_{\mathrm{OPT}}$, then $\forall j, \sigma^{(i)}(j) \leq \sigma^{(\mathrm{OPT})}(j)$, and $A_{\mathrm{tot}}(i) \leq A_{\mathrm{OPT}}$.
- For any step i such that $t_{\max}(i) > \mathrm{COST}_{\mathrm{OPT}}$, then $A_{\mathrm{tot}}(i) \leq p \cdot t_{\max}(i)$.
- There exists $i_0 \geq 0$ such that $t_{\max}(i_0 - 1) > \mathrm{COST}_{\mathrm{OPT}} \geq t_{\max}(i_0)$.
- At the end of step i, $\mathrm{COST}_i \leq 3\max\left(t_{\max}(i), \frac{A_{\mathrm{tot}}(i)}{p}\right)$.

Analysis

A couple of important points:

- Given an optimal solution OPT , $\forall j, t_{j,\sigma^{(\mathrm{OPT})}(j)} \leq \mathrm{COST}_{\mathrm{OPT}}$ and $A_{\mathrm{OPT}} \leq p\mathrm{COST}_{\mathrm{OPT}}$.
- At each step i, $A_{\mathrm{tot}}(i+1) \geq A_{\mathrm{tot}}(i)$ and $t_{\max}(i+1) \leq t_{\max}(i)$.
- For any step i such that $t_{\max}(i) > \mathrm{COST}_{\mathrm{OPT}}$, then $\forall j, \sigma^{(i)}(j) \leq \sigma^{(\mathrm{OPT})}(j)$, and $A_{\mathrm{tot}}(i) \leq A_{\mathrm{OPT}}$.
- For any step i such that $t_{\max}(i) > \mathrm{COST}_{\mathrm{OPT}}$, then $A_{\mathrm{tot}}(i) \leq p \cdot t_{\max}(i)$.
- There exists $i_0 \geq 0$ such that $t_{\max}(i_0 - 1) > \mathrm{COST}_{\mathrm{OPT}} \geq t_{\max}(i_0)$.
- At the end of step i, $\mathrm{COST}_i \leq 3\max\left(t_{\max}(i), \frac{A_{\mathrm{tot}}(i)}{p}\right)$.

$$\begin{aligned}\mathrm{COST}_{i_0} &\leq 3\max\left(t_{\max}(i_0), \frac{A_{\mathrm{tot}}(i_0)}{p}\right)\\ &\leq 3\max\left(\mathrm{COST}_{\mathrm{OPT}}, \frac{A_{\mathrm{OPT}}}{p}\right) \leq 3\mathrm{COST}_{\mathrm{OPT}}\end{aligned}$$

Heuristics

In all heuristics (even randoms), once the different packs are chosen, we always run Optimal-1-pack-schedule on each pack.

Random-Pack: generates the packs randomly: randomly chooses an integer j between 1 and k, and then randomly selects j tasks to form a pack.

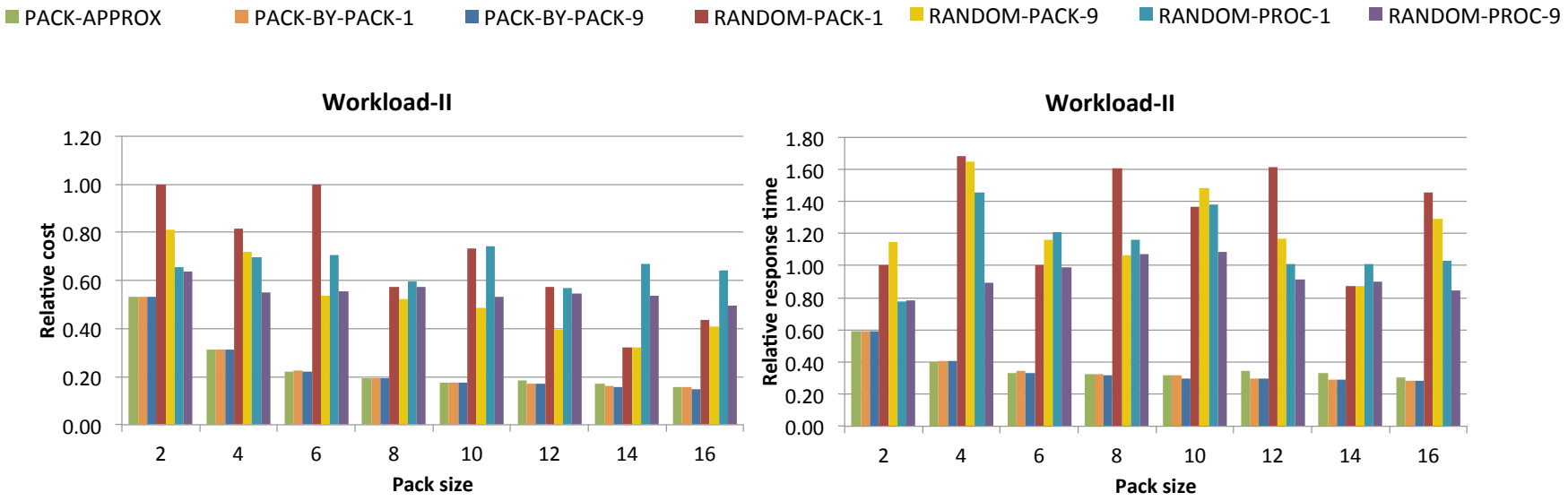
Random-Proc: assigns the number of processors to each task randomly, then calls MAKE-PACK to generate the packs.

pack-by-pack (ε): creates packs that are "well-balanced": the difference between smallest and longest execution times of a pack is small (ratio of $1+\varepsilon$).

pack-Approx: An extension in the case where there are at most k tasks in a pack.

We compare to the heuristics that assigns the maximum number of processors to each task (= scheduler used in practice).

Results

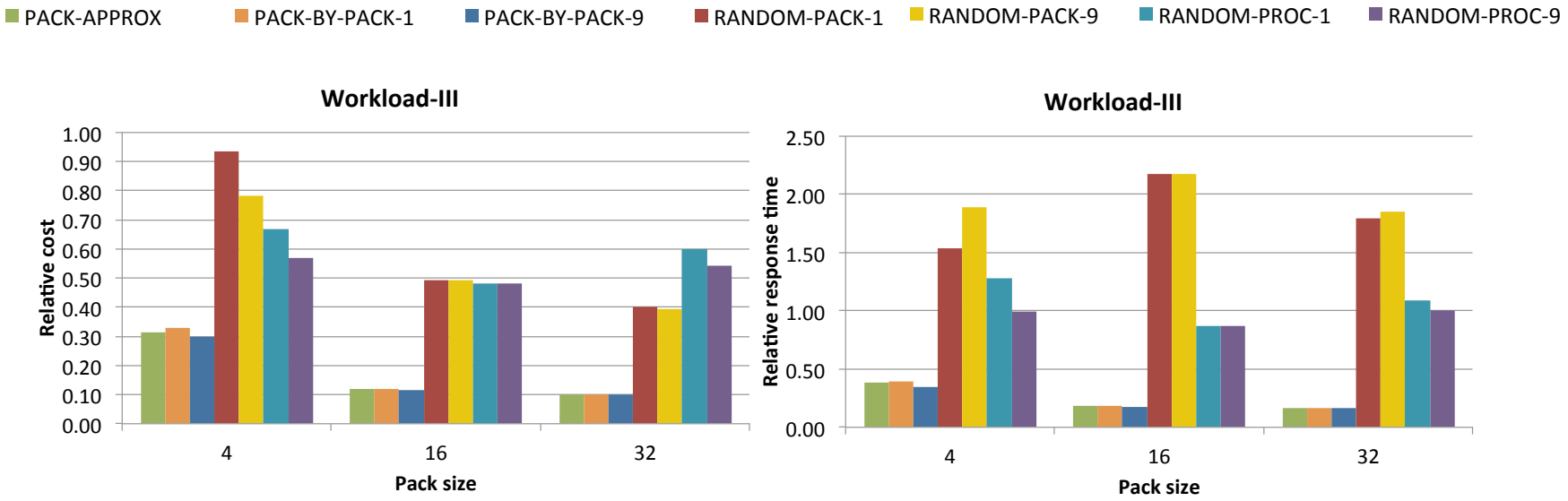


(a) Cost

(b) Response Time

Figure : 65 tasks executing on 16 processors (128 cores).

Results



(a) Cost (b) Response Time

Figure : 260 tasks executing on 32 processors (256 cores).

Conclusion and future work

- *Theoretically:* Hardness + Approximation results
- *Experimentally:* Great improvements compared to existing co-schedulers

Conclusion and future work

- *Theoretically:* Hardness + Approximation results
- *Experimentally:* Great improvements compared to existing co-schedulers

- The main direction is: can we improve the approximation factor? Right now, it does not take pack scheduling into account. We need to bridge the difference "packs/no-packs".
- Approximation on relative response time?
- We need larger simulations.