Online scheduling

Frédéric Vivien
Frederic.Vivien@inria.fr

October 16, 2013
Outline

1. Introduction and first results
2. Lower bound on the competitive ratio of any algorithm: the clairvoyant max-stretch case
3. The non-clairvoyant case
4. How to derive a lower bound: the max-flow case with communications
1. Introduction and first results

2. Lower bound on the competitive ratio of any algorithm: the clairvoyant max-stretch case

3. The non-clairvoyant case

4. How to derive a lower bound: the max-flow case with communications
Offline vs. online algorithmics

Nature of the problem
Known

Objective function
Known

Characteristics of the instance
Known beforehand

Offline
Offline vs. online algorithmics

Nature of the problem
Known

Objective function
Known

Characteristics of the instance
Known
Discovered during execution

Offline
Offline vs. online algorithmics

Nature of the problem
- Known

Objective function
- Known

Characteristics of the instance
- Known beforehand
- Discovered during execution
- Characteristics of a job discovered when the job is released

Offline (Clairvoyant) Online
Offline vs. online algorithmics

Nature of the problem
Known

Objective function
Known

Characteristics of the instance

Known
Beforehand

Discovered during execution
Characteristics of a job discovered
When the job is released
When the job completes

Offline
(Clairvoyant) Online
Non-clairvoyant online
Notation and hypotheses

Notation

- Jobs $J_1, \ldots, J_n$
  - Job $J_j$ arrives in the system at the release date $r_j$
  - Job $J_j$ has a weight (or a priority) $w_j$
  - Job $J_j$ has an execution time $p_j$
  - $\Delta$ is the ratio of the largest to the shortest execution time

- Completion time of job $J_j$: $C_j$
  - Flow of job $J_j$: $F_j = C_j - r_j$ (time spent in the system)

Hypotheses

- Jobs may be preempted
- One machine (1 | $pmtn$ | ???)
What should we optimize?

- Makespan: $\max_j C_j$

$$\text{Inconvenient: starvation}$$

- Maximum flow or maximum response time:
  $$\max_j (C_j - r_j)$$
  No starvation. Favor long jobs. Worst-case optimization.

- Maximum weighted flow:
  $$\max_j w_j (C_j - r_j)$$
  Gives back some importance to short jobs.

Particular case of the stretch or slowdown:
$$w_j = \frac{1}{\text{running time of the job on empty platform}}$$
What should we optimize?

- **Makespan:** $\max_j C_j$

![Diagram showing schedule comparison]

---

- **Average flow or response time:**
  $$\sum_j (C_j - r_j)$$

- **Maximum flow or maximum response time:**
  $$\max_j (C_j - r_j)$$

- **Maximum weighted flow:**
  $$\max_j w_j (C_j - r_j)$$

Gives back some importance to short jobs.

---

Particular case of the stretch or slowdown:
$$w_j = \frac{1}{\text{running time of the job on empty platform}}.$$
What should we optimize?

- **Makespan**: \(\max_j C_j\)
  Release dates are not taken into account

- **Average flow or response time**: \(\sum_j (C_j - r_j)\)
  Inconvenient: starvation

- **Maximum flow or maximum response time**: \(\max_j (C_j - r_j)\)
  No starvation. Favor long jobs. Worst-case optimization.

- **Maximum weighted flow**: \(\max_j w_j (C_j - r_j)\)
  Gives back some importance to short jobs.
  Particular case of the *stretch* or *slowdown*:
  \(w_j = 1/\text{running time of the job on empty platform.}\)
FIFO is optimal for max-flow

Consider any instance and a schedule $\Theta$ s.t. there exists two jobs executed consecutively: $J_i$ and $J_j$ with $r_i < r_j$ and $C_i \geq C_j$.
FIFO is optimal for max-flow

Consider any instance and a schedule $\Theta$ s.t. there exists two jobs executed consecutively: $J_i$ and $J_j$ with $r_i < r_j$ and $C_i \geq C_j$

In schedule $\Theta'$ we exchange the execution order of $J_i$ and $J_j$

$$\max_{1 \leq k \leq n} C'_k - r_k = \max \left\{ \max_{1 \leq k \leq n} C_k - r_k, C'_i - r_i, C'_j - r_j \right\}$$

$$C'_i - r_i \leq C_i - r_i \quad \text{and} \quad C'_j - r_j = C_i - r_j < C_i - r_i$$

$$\Rightarrow \quad \max_{1 \leq k \leq n} C'_k - r_k \leq \max_{1 \leq k \leq n} C_k - r_k$$
FIFO is sub-optimal for max-stretch

Max-stretch of FIFO: \( \max\{1, \frac{4-1}{1}\} = 3. \)

Optimal max-stretch: \( \max\{\frac{5-0}{3}, 1\} = \frac{5}{3}. \)
An online algorithm has a competitive factor $\rho$ if and only if

Whatever the set of jobs $J_1, \ldots, J_n$

Online schedule cost($J_1, \ldots, J_N$) $\leq$ $\rho \times$ Optimal off-line schedule cost($J_1, \ldots, J_N$)
A peculiar framework: tasks are presented one by one to the scheduler that must schedule each task on a processor before seeing the next submitted task (online-list).

Theorem

Any list scheduling algorithm is $2 - \frac{1}{p}$-competitive for the online minimization of the makespan on $p$ processors, and this bound is tight.
The case of list schedules (2/2)

**Theorem**

*If the platform contains 2 or 3 processors (i.e., \( p = 2 \) or \( p = 3 \)), then any list scheduling algorithm achieves the best possible competitive ratio for the online minimization of the makespan.*

\[ p = 2. \] We consider the instances \( \mathcal{I}_1 = (1, 1) \) and \( \mathcal{I}_2 = (1, 1, 2) \).

\[ p = 3. \] We consider three instances: \( \mathcal{I}_1 = (1, 1, 1) \), \( \mathcal{I}_2 = (1, 1, 1, 3, 3, 3) \), and \( \mathcal{I}_3 = (1, 1, 1, 3, 3, 3, 6) \).
FIFO competitiveness

**Theorem**

*First come, first served is:*

- optimal for the online minimization of max-flow
- $\Delta$-competitive for the online minimization of sum-flow
- $\Delta$-competitive for the online minimization of max-stretch
- $\Delta^2$-competitive for the online minimization of sum-stretch
FIFO competitiveness

Theorem

First come, first served is:

- optimal for the online minimization of max-flow
- $\Delta$-competitive for the online minimization of sum-flow
- $\Delta$-competitive for the online minimization of max-stretch
- $\Delta^2$-competitive for the online minimization of sum-stretch
FIFO competitiveness for max-stretch

Theorem

FIFO is $\Delta$ competitive for maximum stretch minimization

This means that

1. FIFO has a competitive factor of $\Delta$ (i.e., on no instance is FIFO’s max-stretch more than $\Delta$ that of the optimal solution)
2. This bound is tight (cannot be improved)
Upper bound for max-stretch

During \( [r, C] \), FIFO exactly executes \( J_i, J_{i+1}, \ldots, J_{l-1}, J_l \). As \( C^* < C_l \), there is a job \( J_k, i \leq k \leq l-1 \) s.t. \( C^*_k \geq C_l \). Then:

\[
S^* = \max_j S^*_j \geq S^*_k = C^*_k - r_k p_k \geq C_l - r_l p_l = C_l \times 1 \Delta \forall l, S_l > S^*_l \Rightarrow \Delta \times S^*_j \geq S_l
\]
During $[r_i, C^*_l]$, FIFO exactly executes $J_i, J_{i+1}, \ldots, J_{l-1}, J_l$.

Optimal $\Theta^*$ graph(142,325),(757,633)

- **Optimal $\Theta^*$**
  - During $[r_i, C^*_l]$, there is a job $J_k$, $i \leq k \leq l-1$ such that $C^*_k \geq C_l$.
  - Then:
    
    $S^*_l = \max_j S^*_j \geq S^*_k = C^*_k - r_k \geq C_l - r_l \geq S_l \times 1$  
  
    $\forall l$, $S_l > S^*_l \Rightarrow \Delta \times S^*_l \geq S_l$
Upper bound for max-stretch

Any job $J_l$ s.t. $S_l > S^*_l$ ($\Leftrightarrow C_l > C^*_l$)
t last time before $C_l$ s.t. the processor was idle under FIFO.
t is the release date $r_i$ of some job $J_i$. 

During $[r_i, C_l]$, FIFO exactly executes $J_i, J_i+1, \ldots, J_{l-1}, J_l$.

As $C^*_l < C_l$, there is a job $J_k$, $i \leq k \leq l-1$ s.t. $C^*_k \geq C_l$.

Then:

$S^*_l = \max_j S^*_j \geq S^*_k = C^*_k - r_k p_k \geq C_l - r_l p_l = C_l - r_l p_l \geq S_l \times 1 \Delta \forall l, S_l > S^*_l \Rightarrow \Delta \times S^*_l \geq S_l$
Upper bound for max-stretch

Any job $J_l$ s.t. $S_l > S_l^*$ ($\iff C_l > C_l^*$)
During $[r_i, C_l]$, FIFO exactly executes $J_i, J_{i+1}, \ldots, J_{l-1}, J_l$. 
Upper bound for max-stretch

Any job \( J_l \) s.t. \( S_l > S_l^\ast \) (\( \Leftrightarrow C_l > C_l^\ast \))

During \([r_i, C_l]\), FIFO exactly executes \( J_i, J_{i+1}, \ldots, J_{l-1}, J_l \).

As \( C_l^\ast < C_l \), there is a job \( J_k, i \leq k \leq l-1 \) s.t. \( C_k^\ast \geq C_l \). Then:

\[
S^\ast = \max_j S_j^\ast \geq S_k^\ast = \frac{C_k^\ast - r_k}{p_k} \geq \frac{C_l - r_l}{p_k} = \frac{C_l - r_l}{p_l} \frac{p_l}{p_k} \geq S_l \times \frac{1}{\Delta}
\]

\( \forall l, S_l > S_l^\ast \Rightarrow \Delta \times S^\ast \geq S_l \)
The bound is tight

\[
\text{Competitive ratio: } 1 + \Delta - \epsilon = \Delta - \epsilon
\]

\[
1\quad \Delta
\]

\[
0\quad \epsilon\quad \text{time}
\]
The bound is tight

\[ \text{Max-stretch} = 1 + \Delta - \epsilon \]

FIFO

\[ \text{Max-stretch} = \frac{1 + \Delta - \epsilon}{1} \]
The bound is tight

\[
\text{Max-stretch} = 1 + \Delta - \epsilon
\]

\[
\text{Optimal Max-stretch} = \frac{1 + \Delta - \epsilon}{1}
\]

\[
\text{Competitive ratio: } 1 + \Delta - \epsilon
\]
The bound is tight

FIFO  Max-stretch = $\frac{1+\Delta-\epsilon}{1}$

Optimal  Max-stretch = $\frac{1+\Delta}{\Delta}$

Competitive ratio: $\frac{1+\Delta-\epsilon}{\frac{1+\Delta}{1+\Delta}} = \Delta \frac{1+\Delta-\epsilon}{1+\Delta} = \Delta - \epsilon \frac{\Delta}{1+\Delta} \geq \Delta - \epsilon$
Outline

1. Introduction and first results

2. Lower bound on the competitive ratio of any algorithm: the clairvoyant max-stretch case

3. The non-clairvoyant case

4. How to derive a lower bound: the max-flow case with communications
Bound on the competitive ratio

**Theorem**

On one processor, any online scheduling algorithm with preemption minimizing the max-stretch has a competitive ratio greater than $\frac{1}{2} \Delta \sqrt{2} - 1$, if the system receives at least jobs of three different sizes, and if $\Delta$ is the ratio between the size of the largest and the smallest job.
Bound on the competitive ratio

Theorem

On one processor, any online scheduling algorithm with preemption minimizing the max-stretch has a competitive ratio greater than $\frac{1}{2} \Delta \sqrt{2} - 1$, if the system receives at least jobs of three different sizes, and if $\Delta$ is the ratio between the size of the largest and the smallest job.

Proof principle: by contradiction we assume that there exists an algorithm and we build a sequence of jobs and a scenario to make the algorithm fail.
The adversary
The adversary
The adversary

Achievable stretch: \( \frac{2\delta - 0}{\delta} = 2. \)
The adversary

\[ k \delta^2 - \delta \]
The adversary

The job $J_{2+j}$ arrives at time $2\delta + (j - 2)k$. 

\[
\text{The adversary} \\
\delta \\
\alpha \text{ tasks of size } k \\
0 \quad \delta \quad 2\delta - k \quad 2\delta \quad 2\delta + (\alpha - 2)k \\
\text{The job } J_{2+j} \text{ arrives at time } 2\delta + (j - 2)k.
\]
The adversary

\[ \delta \quad \alpha \text{ tasks of size } k \]

The job \( J_{2+j} \) arrives at time \( 2\delta + (j - 2)k \).
The adversary

The job $J_{2+j}$ arrives at time $2\delta + (j - 2)k$.

Achievable stretch: $\frac{(2\delta + jk) - (2\delta + (j - 2)k)}{k} = 2$. 
In practice: we do not know what happens after $2\delta - k$. 
The adversary

We want to forbid this case (each size-$k$ job being executed at its release date).
The adversary

We want to forbid this case (each size-$k$ job being executed at its release date).

The algorithm being $\frac{1}{2} \Delta^{\sqrt{2}-1}$-competitive, $J_1$ and $J_2$ must be completed at the latest at time: $2 \cdot \frac{1}{2} \Delta^{\sqrt{2}-1} \cdot \delta = 2 \cdot \frac{1}{2} \left( \frac{\delta}{k} \right)^{\sqrt{2}-1} \cdot \delta$
The adversary

We want to forbid this case (each size-$k$ job being executed at its release date).

The algorithm being $\frac{1}{2} \Delta \sqrt{2} - 1$-competitive, $J_1$ and $J_2$ must be completed at the latest at time: $2 \cdot \frac{1}{2} \Delta \sqrt{2} - 1 \cdot \delta = 2 \cdot \frac{1}{2} \left( \frac{\delta}{k} \right)^{\sqrt{2} - 1} \cdot \delta$

We let $\alpha = \left\lceil 1 + k - \frac{2\delta}{k} \right\rceil$ and then $2\delta + (\alpha - 1)k \geq 2 \cdot \frac{1}{2} \left( \frac{\delta}{k} \right)^{\sqrt{2} - 1} \cdot \delta$. 
The adversary

We want to forbid this case (each size-$k$ job being executed at its release date).

The algorithm being $\frac{1}{2} \Delta \sqrt{2} - 1$-competitive, $J_1$ and $J_2$ must be completed at the latest at time: $2 \cdot \frac{1}{2} \Delta \sqrt{2} - 1 \cdot \delta = 2 \cdot \frac{1}{2} \left( \frac{\delta}{k} \right) \sqrt{2} - 1 \cdot \delta$

We let $\alpha = \left\lceil 1 + \frac{k - 2\delta}{k} \right\rceil$ and then $2\delta + (\alpha - 1)k \geq 2 \cdot \frac{1}{2} \left( \frac{\delta}{k} \right) \sqrt{2} - 1 \cdot \delta$. 
The adversary

\[ \delta \quad \alpha \text{ tasks of size } k \]

Diagram:

- Blue bars: \( 0 \) to \( \delta \)
- Red bars: \( 2\delta - k \) to \( 2\delta + (\alpha - 2)k \)
The adversary

The job $J_{2+\alpha+j}$ arrives at time $2\delta + (\alpha - 1)k + (j - 1)$. 
The adversary

Achievable stretch (off-line)

Stretch of each job of size $k$ or 1: 1.

Stretch of $J_1$ or $J_2$: $\frac{2\delta + \alpha k + \beta}{\delta}$

Optimal stretch $\leq \frac{2\delta + \alpha k + \beta}{\delta}$
The adversary

\[\delta \quad \alpha \text{ tasks of size } k \quad \beta \text{ tasks of size 1}\]

Achievable stretch (online)
The adversary

\[ \delta \quad \alpha \text{ tasks of size } k \quad \beta \text{ tasks of size 1} \]

Achievable stretch (online)

The last completed job is of size \( k \).

\[
\text{Stretch} \geq \frac{(2\delta + \alpha k + \beta) - (2\delta + (\alpha - 2)k)}{k} = 2 + \frac{\beta}{k}.
\]
The adversary

Achievable stretch (online)

The last completed job is of size 1.

\[
\text{Stretch} \geq \frac{(2\delta + \alpha k + \beta) - (2\delta + (\alpha - 1)k + (\beta - 1))}{1} = k + 1.
\]
The adversary

Achievable stretch (online)

\[ \text{Stretch} \geq \min \left\{ 2 + \frac{\beta}{k}, k + 1 \right\} \]

We let: \( \beta = \lceil k(k - 1) \rceil \)

Then: stretch \( \geq k + 1 \).
The adversary: summing things up

\[ \alpha = \left\lceil 1 + k - \frac{2\delta}{k} \right\rceil \]

\[ \beta = \lceil k(k - 1) \rceil \]

Optimal stretch \( \leq \frac{2\delta + \alpha k + \beta}{\delta} \)

Achieved stretch \( \geq k + 1 \).
The adversary: summing things up

\[ \alpha = \left\lceil 1 + k - \frac{2\delta}{k} \right\rceil \]

\[ \beta = \left\lceil k(k - 1) \right\rceil \]

Optimal stretch \( \leq \frac{2\delta + \alpha k + \beta}{\delta} \)

Achieved stretch \( \geq k + 1 \).

We let \( k = \delta^{2-\sqrt{2}} \)
The adversary: summing things up

\[ \alpha = \left\lceil 1 + k - \frac{2\delta}{k} \right\rceil \]

\[ \beta = \left\lfloor k(k - 1) \right\rfloor \]

Optimal stretch \( \leq \frac{2\delta + \alpha k + \beta}{\delta} \)

Achieved stretch \( \geq k + 1 \).

We let \( k = \delta^{2-\sqrt{2}} \)

Therefore \( k + 1 > \left( \frac{1}{2} \delta^{\sqrt{2}-1} \right) \left( \frac{2\delta + \alpha k + \beta}{\delta} \right) \)
Outline

1. Introduction and first results
2. Lower bound on the competitive ratio of any algorithm: the clairvoyant max-stretch case
3. The non-clairvoyant case
4. How to derive a lower bound: the max-flow case with communications
Theorem

First come, first served is:

- optimal for the online minimization of max-flow
- $\Delta$-competitive for the online minimization of sum-flow
- $\Delta$-competitive for the online minimization of max-stretch
- $\Delta^2$-competitive for the online minimization of sum-stretch
Lower bound as a function of $n$

**Theorem**

There is no $c$-competitive preemptive online algorithm minimizing the maximum stretch with $c < n$

**Principle of the proof**

- We suppose there exists an algorithm whose ratio $c = n - \epsilon$
- $n$ jobs are released at time 0
- Whatever the scheduler does, no job completes before time $n$
- Jobs are sorted by non-decreasing cumulative computation time computed at time $n$: the $i$-th job is of size $\lambda^{i-1}$
- The maximum stretch is at least $n$ (first job has size 1 and is not completed at $n$)
- Optimal: execute jobs in Shortest Processing Time first order:

$$\frac{\sum_{j=1}^{i} \lambda^{j-1}}{\lambda^{i-1}} = \frac{\lambda^{i} - 1}{\lambda^{i-1}(\lambda - 1)} \xrightarrow[\lambda \to +\infty]{} 1$$
EquiPartition

Theorem

*EquiPartition is \( n\)-competitive for the minimization of maximum stretch.*

However, EquiPartition is at best \( \frac{\Delta+1}{2+\ln(\Delta)} \) competitive (when FIFO is \( \Delta \) competitive)
Outline

1. Introduction and first results
2. Lower bound on the competitive ratio of any algorithm: the clairvoyant max-stretch case
3. The non-clairvoyant case
4. How to derive a lower bound: the max-flow case with communications
The scheduling problem

The scheduler
- Gather the jobs
- Send them to the processors

The aim
Distribute the identical jobs to the processors, for the jobs to be processed in the best possible way.
The scheduling problem

Formally

- $n$ jobs, $m$ processors
- $p_j$: processing time of a job on processor $j$
- $c_j$: time to send a job from the master to the worker $j$
- $r_i$: release date of job $J_i$
- $C_i$: completion time of job $J_i$

The objective function:
- maximal flow: $\max C_i - r_i$
Finding a lower bound on the competitiveness (1)

Idea:

- A fast processor with slow communications ($c_1 > 1$)
- Two identical and slow processors, with fast communications
- If only one job, one must choose the fast processor ($c_1 + p_1 < 1 + p_2$)
Finding a lower bound on the competitiveness (1)
Finding a lower bound on the competitiveness (1)

We look at time $\tau \geq 1$ to see what has happened. Three possibilities:
Finding a lower bound on the competitiveness (1)

We look at time $\tau \geq 1$ to see what has happened. Three possibilities:

1. **Optimal**: job on $P_1$, max-flow $\geq c_1 + p_1$. 

$$P_3(1, p_2)$$

$$P_2(1, p_2)$$

$$P_1(c_1, p_1)$$

0 $\tau$ time
Finding a lower bound on the competitiveness (1)

We look at time $\tau \geq 1$ to see what has happened. Three possibilities:

1. Optimal: job on $P_1$, max-flow $\geq c_1 + p_1$.
2. Nothing done: max-flow $\geq \tau + c_1 + p_1$, ratio $\geq \frac{\tau + c_1 + p_1}{c_1 + p_1}$.
Finding a lower bound on the competitiveness (1)

We look at time $\tau \geq 1$ to see what has happened. Three possibilities:

1. Optimal: job on $P_1$, max-flow $\geq c_1 + p_1$.
2. Nothing done: max-flow $\geq \tau + c_1 + p_1$, ratio $\geq \frac{\tau + c_1 + p_1}{c_1 + p_1}$.
3. Job sent to $P_2$, max-flow $\geq 1 + p_2$. Ratio $\geq \frac{1 + p_2}{c_1 + p_1}$.

We want to force the algorithm to process the first job on $P_1$. 
Finding a lower bound on the competitiveness (1)

We look at time $\tau \geq 1$ to see what has happened. If the scheduler did not pick the first possibility, the adversary sends no more jobs. Later we will choose $\tau$, $c_1$, $p_1$ and $p_2$ such that the ratio achieved,

$$\min \left\{ \frac{1 + p_2}{c_1 + p_1}, \frac{\tau + c_1 + p_1}{c_1 + p_1} \right\},$$

is as large as possible.
Finding a lower bound on the competitiveness (1)

At time $\tau$ we send two new jobs.
We consider all the possible cases.
Finding a lower bound on the competitiveness (1)

At time $\tau$ we send two new jobs. The two jobs are executed on $P_1$:

$$\max\{c_1 + p_1, \max\{\max\{c_1, \tau\} + c_1 + p_1, c_1 + 2p_1\} - \tau, \max\{\max\{c_1, \tau\} + c_1 + p_1 + \max\{c_1, p_1\}, c_1 + 3p_1\} - \tau\}$$
Finding a lower bound on the competitiveness (1)

At time $\tau$ we send two new jobs. The first of the two jobs is executed on $P_2$ (or $P_3$), and the other one on $P_1$.

$$\max\{c_1 + p_1, (\max\{c_1, \tau\} + c_2 + p_2) - \tau, \max\{\max\{c_1, \tau\} + c_2 + c_1 + p_1, c_1 + 2p_1\} - \tau\}$$
At time $\tau$ we send two new jobs.
The first of the two jobs is executed on $P_1$, and the other one on $P_2$ (or $P_3$).

\[
\max\{c_1 + p_1, \\
\max\{\max\{c_1, \tau\} + c_1 + p_1, c_1 + 2p_1\} - \tau, \\
(\max\{c_1, \tau\} + c_1 + c_2 + p_2) - \tau\} 
\]
Finding a lower bound on the competitiveness (1)

At time $\tau$ we send two new jobs. One of the two jobs is executed on $P_2$ and the other one on $P_3$.

$$\max\{c_1+p_1, (\max\{c_1, \tau\}+c_2+p_2)-\tau, (\max\{c_1, \tau\}+c_2+c_2+p_2)-\tau\}$$
Finding a lower bound on the competitiveness (1)

At time $\tau$ we send two new jobs.
The case where both jobs are executed on $P_2$ (or both on $P_3$) is worse than the previous one, therefore, we do not need to study it.
At time $\tau$ we send two new jobs. The (desired) optimal: the first job on $P_2$, the second on $P_3$, and the third on $P_1$.

$$\max\{c_2+p_2, (\max\{c_2, \tau\}+c_2+p_2) - \tau, (\max\{c_2, \tau\}+c_2+c_1+p_1) - \tau\}$$
Finding a lower bound on the competitiveness (2)

Lower bound on the competitiveness of any online algorithm:

\[
\min \left\{ \frac{\tau + c_1 + p_1}{c_1 + p_1}, \frac{1 + p_2}{c_1 + p_1} \right\}
\]

\[
= \min \left\{ \max \{c_1 + p_1, \max\{\max\{c_1, \tau\} + c_1 + p_1, c_1 + 2p_1\} - \tau, \max\{\max\{c_1, \tau\} + c_1 + p_1 + \max\{c_1, p_1\}, c_1 + 3p_1\} - \tau\} \right\}
\]

Problem: to find \(\tau, c_1, p_1,\) and \(p_2\) (as \(c_2 = 1\)) which maximizes this lower bound.
Constraints: \(c_1 + p_1 < 1 + p_2.\)
Finding a lower bound on the competitiveness (3)

1. Numeric resolution
Finding a lower bound on the competitiveness (3)

1. Numeric resolution
2. Characterization of the shape of the optimal: \( \tau < c_1, p_1 = 0, \) etc.
Finding a lower bound on the competitiveness (3)

1. Numeric resolution
2. Characterization of the shape of the optimal: \( \tau < c_1, p_1 = 0 \), etc.
3. New system:

\[
\begin{align*}
\min \left\{ \frac{\tau + c_1}{c_1}, \frac{1 + p_2}{c_1} \right\} = \min \left\{ \frac{\tau + c_1}{c_1}, \frac{1 + p_2}{c_1}, \frac{c_1 + 1 - \tau + p_2}{1 + p_2} \right\}
\end{align*}
\]
Finding a lower bound on the competitiveness (3)

1. Numeric resolution
2. Characterization of the shape of the optimal: \( \tau < c_1, \ p_1 = 0, \) etc.
3. New system:

\[
\begin{align*}
\min & \quad \left\{ \begin{array}{l}
\frac{\tau + c_1}{c_1} \\
\frac{1 + p_2}{c_1} \\
\min & \quad \left\{ \begin{array}{l}
3c_1 - \tau \\
c_1 + 1 - \tau + p_2 \\
2c_1 - \tau + 1 + p_2 \\
c_1 + 2 + p_2 - \tau \\
\frac{1 + p_2}{1 + p_2} \\
\end{array} \right. \\
\end{array} \right. \\
= \min & \quad \left\{ \begin{array}{l}
\frac{\tau + c_1}{c_1} \\
\frac{1 + p_2}{c_1} \\
c_1 + 1 - \tau + p_2 \\
\frac{1 + p_2}{1 + p_2} \\
\end{array} \right.
\end{align*}
\]

4. Solution: \( c_1 = 2(1 + \sqrt{2}), \ p_2 = \sqrt{2}c_1 - 1, \ \tau = 2, \ \rho = \sqrt{2}. \)