The Relative Worst Order Ratio Applied to Paging

Joan Boyar

Department of Mathematics and Computer Science

University of Southern Denmark, Odense

Joint work with

Lene M. Favrholdt

Kim S. Larsen

University of Southern Denmark

Paging Problem

- Cache: k pages
- Slow memory: $N > k$ pages

- Request sequence: sequence of page numbers
- Fault: page requested not in cache
- Cost: 1 per fault to bring page into cache
- Goal: minimize cost

Refinements of competitive analysis

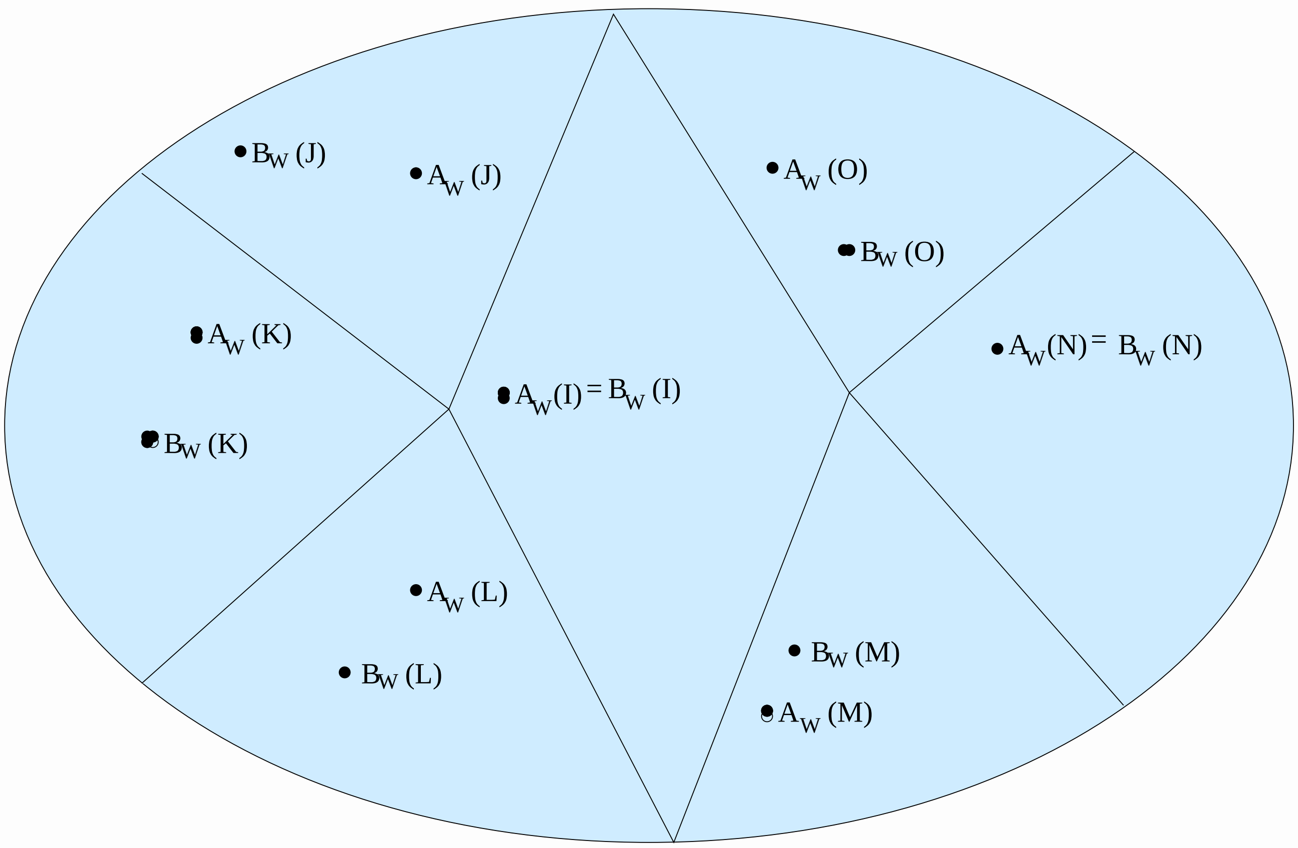
Max/Max Ratio
[Ben-David, Borodin 94]
Compares $\mathbb{A}$ to OPT
on worst sequences of length n.

Random Order Ratio
[Kenyon 95]
Compares $\mathbb{A}$ to OPT
on random ordering of same sequence.

Relative Worst Order Ratio

$\mathbb{A}_W(I)$: $\mathbb{A}'s$ performance on worst permutation of I wrt. $\mathbb{A}$

Intuitively: $\text{WR}_{\mathbb{A},\mathbb{B}} =$ worst-case $\frac{\mathbb{A}_W(I)}{\mathbb{B}_W(I)}$ on long I



Relative Worst Order Ratio

[Boyar,Favrholdt 03]
Formally:

$$c_{\mathsf{l}}(\mathbb{A},\mathbb{B}) \;=\; \sup\,\{c \mid \exists b\colon \forall I\colon \mathbb{A}_{\mathsf{W}}(I) \geq c\,\mathbb{B}_{\mathsf{W}}(I) - b\}$$
$$c_{\mathsf{u}}(\mathbb{A},\mathbb{B}) \;=\; \inf\,\{c \mid \exists b\colon \forall I\colon \mathbb{A}_{\mathsf{W}}(I) \leq c\,\mathbb{B}_{\mathsf{W}}(I) + b\}\;.$$

If $c_{\mathsf{l}}(\mathbb{A},\mathbb{B}) \geq 1$ or $c_{\mathsf{u}}(\mathbb{A},\mathbb{B}) \leq 1$, the algorithms are *comparable*. Then the *relative worst-order ratio* $\mathrm{WR}_{\mathbb{A},\mathbb{B}}$ is defined.

Otherwise, $\mathrm{WR}_{\mathbb{A},\mathbb{B}}$ is undefined.

Relative Worst Order Ratio

$$c_{\mathsf{l}}(\mathbb{A},\mathbb{B}) = \sup\{c \mid \exists b\colon \forall I: \mathbb{A}_{\mathsf{W}}(I) \geq c\,\mathbb{B}_{\mathsf{W}}(I) - b\}$$
$$c_{\mathsf{u}}(\mathbb{A},\mathbb{B}) = \inf\{c \mid \exists b\colon \forall I: \mathbb{A}_{\mathsf{W}}(I) \leq c\,\mathbb{B}_{\mathsf{W}}(I) + b\}\ .$$

If $c_{\mathsf{l}}(\mathbb{A},\mathbb{B}) \geq 1,$ then $\mathrm{WR}_{\mathbb{A},\mathbb{B}} = c_{\mathsf{u}}(\mathbb{A},\mathbb{B}),$ and

if $c_{\mathsf{u}}(\mathbb{A},\mathbb{B}) \leq 1,$ then $\mathrm{WR}_{\mathbb{A},\mathbb{B}} = c_{\mathsf{l}}(\mathbb{A},\mathbb{B})\,.$

Relative Worst Order Ratio

$c_{\mathsf{l}}(\mathbb{A},\mathbb{B}) \geq 1$ or $c_{\mathsf{u}}(\mathbb{A},\mathbb{B}) \leq 1$:
One algorithm is at least as good as the other.

$\mathrm{WR}_{\mathbb{A},\mathbb{B}}$ bounds how much better.

Values of $\mathrm{WR}_{\mathbb{A},\mathbb{B}}$:

	minimization	maximization
$\mathbb{A}$ better than $\mathbb{B}$	< 1	> 1
$\mathbb{B}$ better than $\mathbb{A}$	> 1	< 1

Algorithms: LRU vs. FWF

LRU – Least Recently Used
FWF – Flush When Full
Both have competitive ratio k.

Example sequence, $k = 5$:

$$\langle 1, 2, 3, 4, 5, 6, 5, 4, 3, 2, 1, 2, 3, 4, 5, 6, 5, 4, 3, 2 \rangle$$

Total cost LRU = 8
Total cost FWF = 20

FWF vs. LRU

I_{LRU} – worst ordering of I for LRU

$\forall I \quad \text{FWF}_W(I) \geq \text{FWF}(I_{\text{LRU}}) \geq \text{LRU}_W(I)$

Thus, $c_{\text{l}}(\text{FWF}, \text{LRU}) \geq 1$ holds.

FWF vs. LRU

$$I^n = \langle 1, 2, .., k, k+1, k, ...3, 2 \rangle^n$$

$\text{FWF}_W(I^n) = 2kn$

Worst ordering for LRU:

$$\langle 2, ..., k, k+1, 1 \rangle^n, \langle 2, ..., k \rangle^n$$

$\text{LRU}_W(I^n) = n(k+1) + k - 1$

Theorem. $\text{WR}_{\text{FWF,LRU}} \geq \frac{2k}{k+1}$

Theorem. $\text{WR}_{\text{FWF,LRU}} = \frac{2k}{k+1}$

Look-Ahead

Model: $\mathbb{A}$ sees request + next l requests: Look-ahead(l)

On-line $\longrightarrow$ Look-ahead(l) $\longrightarrow$ OPT

Fact 3: k is still best possible competitive ratio, even with look-ahead l.

Other Models of Look-Ahead

Resource-bounded look-ahead [Young 91]

Strong look-ahead [Albers 93]

Natural look-head [Breslauer 98]

Look-ahead

LRU(ℓ):

- Sees current page and next l pages.
- Avoids evicting pages it sees.
- Evicts l.r.u. among others in cache.

First show $c_{\mathsf{l}}(\text{LRU}, \text{LRU}(\ell)) \geq 1$ holds:
Theorem. For any sequence I,
$\text{LRU}_W(I) \geq \text{LRU}(\ell)_W(I)$.

LRU vs. LRU(ℓ)

Sequence I. Partition into phases:
LRU(ℓ) faults $k+1$ times per phase.
Suppose $\leq k$ distinct pages in phase P.

$$\langle \ldots \underbrace{p_1, \ldots, p, \ldots, q, \ldots, p, \ldots, p_s}_{\text{phase } P;\ k+1 \text{ faults for LRU}(\ell)}, p_{s+1}, \ldots \rangle$$

Page p evicted when q requested.

Least recently used not among next ℓ.

LRU vs. LRU(ℓ)

Case p not among next ℓ:

$$\langle \ldots p_1, \ldots, p \underbrace{, \ldots, q, \ldots,}_{P' \subset P} p, \ldots, p_s, p_{s+1}, \ldots \rangle$$

P' has q and $\geq k-1$ distinct pages.

Phase P has $\geq k+1$ distinct pages.

LRU vs. LRU(ℓ)

Case p not among next ℓ:

$$\langle \ldots p_1, \ldots, p \underbrace{, \ldots, q, \ldots,}_{P' \subset P} p, \ldots, p_s, p_{s+1}, \ldots \rangle$$

P' has q and $\geq k-1$ distinct pages.
Phase P has $\geq k+1$ distinct pages.

Case p among next ℓ:

$$\langle \ldots p_1, \ldots, p, \ldots, q \underbrace{, \ldots,}_{P'' \subset P} p, \ldots, p_s, p_{s+1}, \ldots \rangle$$

$\geq k-1$ distinct in P''; $\geq k+1$ in P.

LRU vs. LRU(ℓ)

Process I by phases.
Example sequence, $k = 5$ and $\ell = 2$:

$$\langle 1, 2, 3, 4, 5, 6, \quad || \; 5, 7, 1, 8, 4, 2, 5, 9, 3 \rangle$$

Reorder phase with new pages first;
others in order from last phase.

$$\langle 1, 2, 3, 4, 5, 6, \quad || \; 7, 8, 9, 1, 2, 3, 4, 5, 5 \rangle$$

LRU faults on $\geq$ as many as LRU(ℓ).

LRU vs. LRU(ℓ)

Consider $I^n = \langle 1, 2, .., k, k+1 \rangle^n$.
I^n has only $k+1$ pages.
LRU faults on every page.

Suppose $l \leq k-1$.
Whenever LRU(ℓ) faults (after first k faults),
it doesn't fault on next l requests.

Suppose $l \geq k$.
LRU(ℓ) faults on $\leq$ 1 page out of k.

Theorem. $\text{WR}_{\text{LRU},\text{LRU}(\ell)} \geq \min\{l+1, k\}$.

Retrospective-LRU

Mimic the optimal algorithm, LFD.

Phases with marking:

Basic Ideas

- Remove marks at start of new phase.
- Mark a requested page if in LFD's cache.
- Avoid evicting marked pages if possible.
- Within the marked/unmarked groups, evict using LRU.
- Start new phase if 2nd fault on same page.

RLRU: request r to page p

if p is not in cache **then**

  if there is no unmarked page **then**

    evict the least recently used page in cache

  else

    evict the least recently used unmarked page

  if second fault on p in current phase **then**

    unmark all pages and start a new phase with r

  if p was in LFD's cache just before this request **then**

    mark p

else

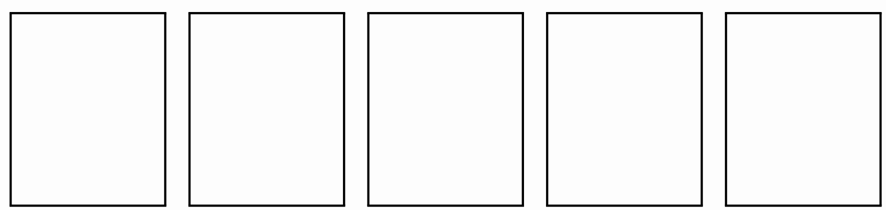
  if p is different from the previous page **then**

    mark p

RLRU – Execution

Example sequence, $k = 5$:

$$\langle 1, 2, 3, 4, 5, 6, 1, 2, 3, 4, 7, 8, 1, 2, 3, 4, 9, 10 \rangle$$



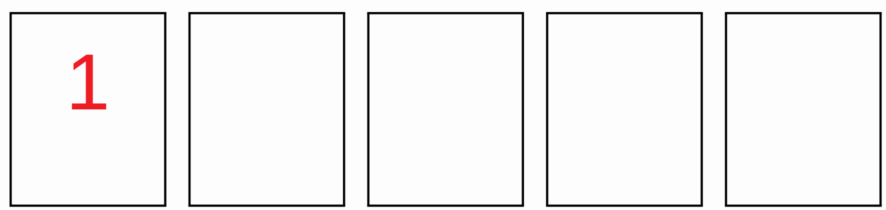
Total cost = 0

Cache initially empty.

RLRU – Execution

Example sequence, $k = 5$:

$$\langle 1, 2, 3, 4, 5, 6, 1, 2, 3, 4, 7, 8, 1, 2, 3, 4, 9, 10 \rangle$$



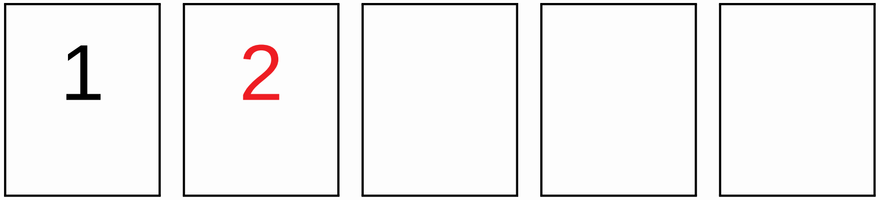
Total cost = 1

Cache filling up.

RLRU – Execution

Example sequence, $k = 5$:

$$\langle 1, 2, 3, 4, 5, 6, 1, 2, 3, 4, 7, 8, 1, 2, 3, 4, 9, 10 \rangle$$



Total cost = 2

Cache filling up.

RLRU – Execution

Example sequence, $k = 5$:

$$\langle 1, 2, 3, 4, 5, 6, 1, 2, 3, 4, 7, 8, 1, 2, 3, 4, 9, 10 \rangle$$

1	2	3		

Total cost = 3

Cache filling up.

RLRU – Execution

Example sequence, $k = 5$:

$$\langle 1, 2, 3, 4, 5, 6, 1, 2, 3, 4, 7, 8, 1, 2, 3, 4, 9, 10 \rangle$$

1	2	3	4	

Total cost = 4

Cache filling up.

RLRU – Execution

Example sequence, $k = 5$:

$$\langle 1, 2, 3, 4, 5, 6, 1, 2, 3, 4, 7, 8, 1, 2, 3, 4, 9, 10 \rangle$$

1	2	3	4	5

Total cost = 5

Cache filling up.

RLRU – Execution

Example sequence, $k = 5$:

$$\langle 1, 2, 3, 4, 5, 6, 1, 2, 3, 4, 7, 8, 1, 2, 3, 4, 9, 10 \rangle$$

6	2	3	4	5

Total cost = 6

Least recently used evicted.

RLRU – Execution

Example sequence, $k = 5$:

$$\langle 1, 2, 3, 4, 5, 6, 1, 2, 3, 4, 7, 8, 1, 2, 3, 4, 9, 10 \rangle$$

6	1	3	4	5

Total cost = 7

Least recently used evicted.
Page marked.

RLRU – Execution

Example sequence, $k = 5$:

$$\langle 1, 2, 3, 4, 5, 6, 1, 2, 3, 4, 7, 8, 1, 2, 3, 4, 9, 10 \rangle$$

6	1	2	4	5

Total cost = 8

Least recently used unmarked evicted.
Page marked.

RLRU – Execution

Example sequence, $k = 5$:

$$\langle 1, 2, 3, 4, 5, 6, 1, 2, 3, 4, 7, 8, 1, 2, 3, 4, 9, 10 \rangle$$

6	1	2	3	5

Total cost = 9

Least recently used unmarked evicted.
Page marked.

RLRU – Execution

Example sequence, $k = 5$:

$$\langle 1, 2, 3, 4, 5, 6, 1, 2, 3, 4, 7, 8, 1, 2, 3, 4, 9, 10 \rangle$$

6	1	2	3	4

Total cost = 10

Least recently used unmarked evicted.
Page marked.

RLRU – Execution

Example sequence, $k = 5$:

$$\langle 1, 2, 3, 4, 5, 6, 1, 2, 3, 4, 7, 8, 1, 2, 3, 4, 9, 10 \rangle$$

7	1	2	3	4

Total cost = 11

Least recently used unmarked evicted.

RLRU – Execution

Example sequence, $k = 5$:

$$\langle 1, 2, 3, 4, 5, 6, 1, 2, 3, 4, 7, 8, 1, 2, 3, 4, 9, 10 \rangle$$

8	1	2	3	4

Total cost = 12

Least recently used unmarked evicted.

RLRU – Execution

Example sequence, $k = 5$:

$$\langle 1, 2, 3, 4, 5, 6, 1, 2, 3, 4, 7, 8, 1, 2, 3, 4, 9, 10 \rangle$$

8	1	2	3	4

Total cost = 12

No fault!

RLRU – Execution

Example sequence, $k = 5$:

$$\langle 1, 2, 3, 4, 5, 6, 1, 2, 3, 4, 7, 8, 1, 2, 3, 4, 9, 10 \rangle$$

8	1	2	3	4

Total cost = 12

No fault!

RLRU – Execution

Example sequence, $k = 5$:

$$\langle 1, 2, 3, 4, 5, 6, 1, 2, 3, 4, 7, 8, 1, 2, 3, 4, 9, 10 \rangle$$

8	1	2	3	4

Total cost = 12

No fault!

RLRU – Execution

Example sequence, $k = 5$:

$$\langle 1, 2, 3, 4, 5, 6, 1, 2, 3, 4, 7, 8, 1, 2, 3, 4, 9, 10 \rangle$$

8	1	2	3	4

Total cost = 12

No fault!

RLRU – Execution

Example sequence, $k = 5$:

$$\langle 1, 2, 3, 4, 5, 6, 1, 2, 3, 4, 7, 8, 1, 2, 3, 4, 9, 10 \rangle$$

9	1	2	3	4

Total cost = 13

Least recently used unmarked evicted.

RLRU – Execution

Example sequence, $k = 5$:

$$\langle 1, 2, 3, 4, 5, 6, 1, 2, 3, 4, 7, 8, 1, 2, 3, 4, 9, 10 \rangle$$

10	1	2	3	4

Total cost = 14

Least recently used unmarked evicted.

RLRU – Execution

Example sequence, $k = 5$:

$$\langle 1, 2, 3, 4, 5, 6, 1, 2, 3, 4, 7, 8, 1, 2, 3, 4, 9, 10 \rangle$$

Asymptotically, RLRU faults on 2 pages per group (regardless of ordering).

LRU faults on $k + 1$ pages per group.

So LRU can be a factor $\frac{k+1}{2}$ worse than RLRU.

Experimental Results

Tested on a collection of traces from various applications:

- key word searches in text files
- selections and joins in Postgres
- external sorting
- various kernel operations

Trace lengths vary from 18,533 to 95,723 requests.

Cache sizes powers of two from 8 through 2048.

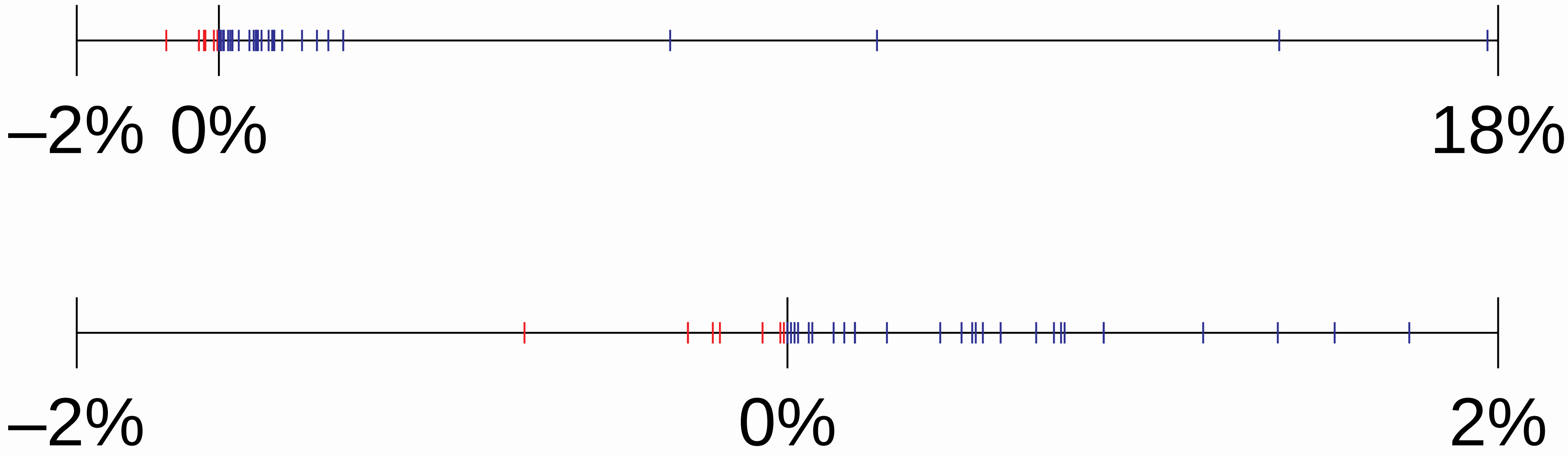
For higher powers, all pages can fit in cache

(for most sequences).

Experimental Results

k	sort	j1	j2	j3	j4	j5	j6	join	pq7	xds
	12619	470	8177	4243	7201	25332	4596	7718	9277	10762
16	10736	468	8134	4255	7221	25326	4525	7003	9259	10709
	14.92	**0.43**	**0.53**	**-0.28**	**-0.28**	**0.02**	**1.54**	**9.26**	**0.19**	**0.49**
	10587	136	8120	4230	7135	25276	4505	6879	9185	10754
64	10402	137	8057	4239	7140	25278	4506	6838	9103	10695
	1.75	**-0.74**	**0.78**	**-0.21**	**-0.07**	**-0.01**	**-0.02**	**0.60**	**0.89**	**0.55**
	10238	126	8118	4213	7039	25209	4499	6793	8989	10564
256	10166	126	8057	4221	7038	24913	4492	6780	8984	10534
	0.70	**0.00**	**0.75**	**-0.19**	**0.01**	**1.17**	**0.16**	**0.19**	**0.06**	**0.28**
	9618	126	5060	1921	6709	24024	4476	6042	8674	10190
1024	9532	126	4157	1799	6674	23693	4470	6040	8607	10183
	0.89	**0.00**	**17.85**	**6.35**	**0.52**	**1.38**	**0.13**	**0.03**	**0.77**	**0.07**

Experimental Results



Other Results with Relative Worst Order Ratio

1. Bin Packing: Worst-Fit better than Next-Fit.
2. Dual Bin Packing:
   First-Fit better than Worst-Fit.
3. Scheduling – minimizing makespan:
   Post-Greedy better than putting all jobs on fast machine, for two related machines.
4. Bin Coloring:
   Greedy better than keeping only one open bin.
5. Proportional Price Seat Reservation:
   First-Fit better than Worst-Fit.