Scheduling Single AGV in Blocking Flow-Shop with Identical Jobs : *a dynamic programming approach*

Alessandro Minoli

University of Milan

06/02/2025

UNIVERSITÀ
DEGLI STUDI
DI MILANO

Outline

Problem description

Data :

- J identical jobs (products)
- a loading station s_0 (pick-up)
- M workstations $\in \{s_1, s_2, ..., s_M\}$
- an unloading station s_{M+1} (delivery)

- $\forall i \in \{1, 2, ..., M\}, p_i \in \mathbb{N}^+$: processing time on workstation s_i

- $\forall i, j \in \{0, 1, ..., M+1\}, t_{ij} \in \mathbb{N}$: AGV moving time from s_i to s_j
  - moving times are symmetrical
  - moving times respect the triangle inequality
  - moving times are the same regardless of whether the AGV is carrying a product or not

Constraints :

- initially, all the products are in the loading station s_0
- the manufacturing process of each product requires it to pass through all the workstations sequentially from s_1 to s_M
- the products will eventually end up in the unloading station s_{M+1}
  - s_0 and s_{M+1} have unlimited capacity
  - all workstations can hold a maximum of one product at a time and have no buffer
- when a product arrives at workstation s_i it must be processed for p_i time units; it cannot be moved to the next station until its processing is completed
- an AVG can take a product from s_i and move it to s_{i+1} $(i \leq M)$ only if s_{i+1} is an empty workstation or the unloading station
  - the AGV can hold a maximum of one product at a time
  - we assume that loading products onto the AGV and unloading products from it takes no time

Objective :

- find the scheduling of AGV operations that minimizes
  the time to complete the manufacturing process of all products
  i.e. the time to get all products to the unloading station

Remarks :

- since all the jobs are identical, the order in which they are picked up from s_0 has no influence on the completion time
- since we are dealing with a flow-shop (all products have the same sequence of workstations) and there are no buffers, no product can overtake another product on its way through the workstations
- only the order in which the products are moved between the stations influences the completion time

Sample instance

- $J = 4$, $M = 3$
-

p_1	p_2	p_3
11	54	4

-

t_{ij}	0	1	2	3	4
0	0	25	16	15	19
1	25	0	18	18	17
2	16	18	0	16	25
3	15	18	16	0	23
4	19	17	25	23	0

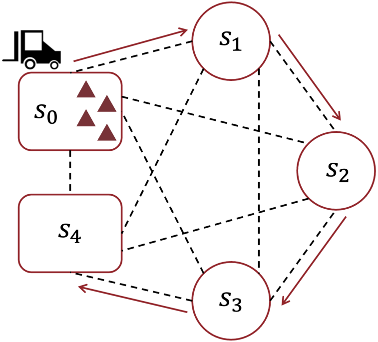


Figure:
Graphical representation of the instance. Arrows indicate the path that each product must take through the workstations

State of the art

- The reference article for this problem is :

  Scheduling Single AGV in Blocking Flow-Shop with Identical Jobs.

  E. Boom, M. Mihalák, F. Thuijsman, M.H.M. Winands
  (February 2024)

- The authors proposed for this problem :
  - an exact ILP-based algorithm
  - an ILP-based heuristic
  - two greedy heuristics

- The use of a dynamic programming technique seems promising for tackling the problem and it is exactly what I am experimenting with in this project

Trivial feasible solution

- Every instance of the problem has a trivial feasible solution, the cost of which is an upper bound on its optimal completion time
- This solution consists of moving one product at a time to the unloading station, thus waiting for its processing time on each workstation
- It has the following cost :

$$UB = J \cdot \left(\sum_{i=0}^{M} t_{i,\,i+1} + \sum_{i=1}^{M} p_i \right) + (\, J - 1 \,) \cdot t_{M+1,\,0}$$

- The formula applied to the sample instance gives :

$$UB = 4 \cdot (\, 69 + 82 \,) + 3 \cdot 19 = 661$$

DP Algorithm 1 (definitions)

States :

- $\langle\ t\ ,\ x\ ,\ l\ ,\ w_1\ ,\ \dots\ ,\ w_M\ ,\ u\ \rangle$ where :

$$
\begin{array}{rlll}
t & \in \{0 \dots UB\} & & \text{elapsed time} \\
x & \in \{0 \dots M+1\} & & \text{AGV position (station)} \\
l & \in \{0 \dots J\} & & \#\ \text{products in loading station} \\
w_i & \in \{\perp\} \cup \{0 \dots p_i\} & & \forall i \in \{1 \dots M\} \\
 & \text{if } s_i \text{ is empty :} & & \text{equal to } \perp \\
 & \text{if } s_i \text{ is non-empty :} & & \text{equal to the residual processing time} \\
 & & & \text{of the product in workstation } s_i \\
u & \in \{0 \dots J\} & & \#\ \text{products in unloading station}
\end{array}
$$

- Given $P = \max_{i \in \{1 \dots M\}} p_i$, number of states is $O\left(\ UB \cdot M \cdot J^2 \cdot P^M\ \right)$

Initialization : $\langle\ 0\ ,\ 0\ ,\ J\ ,\ \perp\ ,\ \dots\ ,\ \perp\ ,\ 0\ \rangle$

Label extension (idea) :

- From each state we can make moves composed of these actions :
  - from the current station s_x we move the AGV to a station s_y, which can be :
    - a) the loading station s_0 — if it is non-empty and s_1 is empty
    - b) a workstation $s_{i \in \{1 \ldots M-1\}}$ — if it is non-empty and s_{i+1} is empty
    - c) the workstation s_M — if it is non-empty

    (null-movement is possible)
  - if $s_y \neq s_0$, we wait the residual processing time of the product on s_y
  - we transport the product from s_y to s_{y+1} by AGV (or, if $s_y = s_0$, any of the products in s_0)
- For each state : $0 < \#$ possible extensions $\leq \lfloor M/2 \rfloor + 1$
- The cost of the optimal solution is :

$$z^* = \min \{ \ t \mid \langle \ t \ , \ M+1 \ , \ 0 \ , \ \perp \ , \ \ldots \ , \ \perp \ , \ J \ \rangle \text{ is reached } \}$$

Label extension (formalized) :

- notation :

$$w'_i(e) = \begin{cases} \perp & \text{if } w_i = \perp \\ \max\{\, 0\,, w_i - e \,\} & \text{otherwise} \end{cases} \tag{1}$$

- case a) movement towards the loading station s_0

$S = \langle\ t\ ,\ x\ ,\ l > 0\ ,\ w_1 = \perp\ ,\ w_2\ ,\ \ldots\ ,\ w_M\ ,\ u\ \rangle$

  - $t_{x,0}$ passes
  - at s_0 no processing time passes
  - $t_{0,1}$ passes
  - $e := t_{x,0} + t_{0,1}$

$S_{next} = \langle\ t + e\ ,\ 1\ ,\ l - 1\ ,\ w_1 = p_1\ ,\ w'_2(e)\ ,\ \ldots\ ,\ w'_M(e)\ ,\ u\ \rangle$

- case b) movement towards a workstation $s_{i \in \{1 \dots M-1\}}$

$$S = \langle\ t\ ,\ x\ ,\ l\ ,\ w_1\ ,\ \dots\ ,\ w_i \neq \bot\ ,\ w_{i+1} = \bot\ ,\ \dots\ ,\ w_M\ ,\ u\ \rangle$$

  - $t_{x,\,i}$ passes
  - at s_i passes $t_p = \max\ \{\ 0\ , w_i - t_{x,\,i}\ \}$
  - $t_{i,\,i+1}$ passes
  - $e := t_{x,\,i} + t_p + t_{i,\,i+1}$

$$S_{next} = \langle\ t + e\ ,\ i+1\ ,\ l\ ,\ w_1'(e)\ ,\ \dots\ ,\ w_i = \bot\ ,\ w_{i+1} = p_{i+1}\ ,\ \dots\ ,\ w_M'(e)\ ,\ u\ \rangle$$

- case c) movement towards the workstation s_M

$$S = \langle\ t\ ,\ x\ ,\ l\ ,\ w_1\ ,\ \dots\ ,\ w_{M-1}\ ,\ w_M \neq \bot\ ,\ u\ \rangle$$

  - $t_{x,\,M}$ passes
  - at s_M passes $t_p = \max\ \{\ 0\ , w_M - t_{x,\,M}\ \}$
  - $t_{M,\,M+1}$ passes
  - $e := t_{x,\,M} + t_p + t_{M,\,M+1}$

$$S_{next} = \langle\ t + e\ ,\ M+1\ ,\ l\ ,\ w_1'(e)\ ,\ \dots\ ,\ w_{M-1}'(e)\ ,\ w_M = \bot\ ,\ u+1\ \rangle$$

Dominance :

- Given two states

$$S' = \langle\ t',\ x',\ l',\ w'_1,\ \dots,\ w'_M,\ u'\ \rangle$$
$$S'' = \langle\ t'',\ x'',\ l'',\ w''_1,\ \dots,\ w''_M,\ u''\ \rangle$$

$$S' \prec S'' \quad \Leftrightarrow \quad \begin{aligned} & t' < t'' \\ & x' = x'' \\ & l' = l'' \\ & w'_i = w''_i \quad \forall i \in \{1 \dots M\} \\ & u' = u'' \end{aligned}$$

DP Algorithm 1 (implementation details)

- label correcting
- for each state we keep track of the predecessor, so we can reconstruct the optimal AGV route at the end of the algorithm
- states with lower t are extended first
  - popping a state with minimum t from the pool is $O(1)$
  - adding a state to the pool is $O(1)$

Dominance checks :

- dominance checks done with a map H (hash-table implementation)
  - when a state $S = \langle\ t\ ,\ x\ ,\ l\ ,\ w_1\ ,\ \ldots\ ,\ w_M\ ,\ u\ \rangle$ is visited, we consider all its components except t as key k and t as value
  - if k is not in H
    - then we set $H[k] = t$
  - if k is in H and $H[k] = t'$,
    - if $t < t'$, then we set $H[k] = t$
    - otherwise, S is dominated by a previously visited state and must not be extended

DP Algorithm 1 (sample instance solution)

- DP algorithm 1 applied to the sample instance finds an optimal AGV route represented by this sequence of states :

t	x	l	w_1	w_2	w_3	u
0	0	4	$\perp$	$\perp$	$\perp$	0
25	1	3	11	$\perp$	$\perp$	0
54	2	3	$\perp$	54	$\perp$	0
95	1	2	11	13	$\perp$	0
129	3	2	0	$\perp$	4	0
156	4	2	0	$\perp$	$\perp$	1
191	2	2	$\perp$	54	$\perp$	1
232	1	1	11	13	$\perp$	1
266	3	1	0	$\perp$	4	1
293	4	1	0	$\perp$	$\perp$	2
328	2	1	$\perp$	54	$\perp$	2
369	1	0	11	13	$\perp$	2
403	3	0	0	$\perp$	4	2
439	2	0	$\perp$	54	0	2
478	4	0	$\perp$	15	$\perp$	3
519	3	0	$\perp$	$\perp$	4	3
546	4	0	$\perp$	$\perp$	$\perp$	4

- $z^* = 546$ (17.4 % improvement wrt $UB = 661$)

DP Algorithm 2 (definitions)

States :

- Given C_k := completion time of product k on its current station , we define the earliest service time of k as $E_k := \max\ \{\ C_k\ , t + t_{x, x_k}\ \}$ i.e. the first instant at which k can be moved to $x_k + 1$
- $\langle\ t\ ,\ x\ ,\ x_1\ ,\ \dots\ ,\ x_J\ ,\ e_1\ ,\ \dots\ ,\ e_J\ \rangle$ where :

t	$\in \{0 \dots UB\}$	elapsed time
x	$\in \{0 \dots M+1\}$	AGV position
x_k	$\in \{0 \dots M+1\}$	$\forall k \in \{1 \dots J\}$, product position
		$\forall k \in \{1 \dots J-1\}$, $x_k \leq x_{k+1}$
e_k	$\in \{\perp\} \cup \{0 \dots UB\}$	$\forall k \in \{1 \dots J\}$
	if $x_k < M+1$:	equal to E_k
	if $x_k = M+1$:	equal to $\perp$

- The number of states is $O\left(\ (UB \cdot M)^{J+1}\ \right)$

Initialization : $\langle\, 0\,,\ 0\,,\ 0\,,\ \ldots,\ 0\,,\ 0\,,\ \ldots,\ 0\,\rangle$

Label extension (idea) :

- From each state the possible moves and the # possible extensions are the same as in the previous algorithm
- The cost of the optimal solution is :

$$z^* = \min \{\, t \mid \langle\, t\,,\ M{+}1\,,\ M{+}1\,,\ \ldots,\ M{+}1\,,\ \perp\,,\ \ldots,\ \perp\,\rangle \text{ is reached} \,\}$$

Label extension (formalized) :

- $S = \langle\ t\ ,\ x\ ,\ x_k\ \ \forall k \in \{1 \dots J\}\ \ ,\ e_k\ \ \forall k \in \{1 \dots J\}\ \ \rangle$
- we move towards x_α i.e. the current station of product $\alpha \in \{1 \dots J\}$
  - if $x_\alpha = 0$ we can move α forward
    only if no other product $\alpha' > \alpha$ has $x_{\alpha'} = 0$
  - if $x_\alpha = M + 1$ we cannot move α forward
  - if $x_{\alpha+1} > x_\alpha + 1\ \lor\ x_\alpha + 1 = M + 1$ we can move α forward
- $S_{next} = S' = \langle\ t'\ ,\ x'\ ,\ x'_k\ \ \forall k \in \{1 \dots J\}\ \ ,\ e'_k\ \ \forall k \in \{1 \dots J\}\ \ \rangle$
  - $t' = \max\ \{\ t + t_{x, x_\alpha}\ , e_\alpha\ \} + t_{x_\alpha, x_\alpha + 1}$
  - $x' = x_\alpha + 1$
  - $x'_k = \begin{cases} x_k + 1 & \text{if } k = \alpha \\ x_k & \text{otherwise} \end{cases}$
  - $e'_k = \begin{cases} \perp & \text{if } e_k = \perp \lor (\, k = \alpha \land x'_\alpha = M + 1\,) \\ t' + p_{x'_\alpha} & \text{if } k = \alpha \land x'_\alpha \neq M + 1 \\ \max\ \{\ e_k\ , t' + t_{x', x'_k}\ \} & \text{otherwise} \end{cases}$

Dominance :

- Given two states

$$S' = \langle\ t',\ x',\ x'_k\ \ \forall k \in \{1 \dots J\}\ \ ,\ e'_k\ \ \forall k \in \{1 \dots J\}\ \ \rangle$$
$$S'' = \langle\ t'',\ x'',\ x''_k\ \ \forall k \in \{1 \dots J\}\ \ ,\ e''_k\ \ \forall k \in \{1 \dots J\}\ \ \rangle$$

$$\begin{aligned} S' \prec S'' \quad \Leftrightarrow \quad & \forall k \in \{1 \dots J\} \mid x'_k < M + 1 \\ & (\ x'_k > x''_k \ \wedge\ t' \leq t''\ ) \ \vee\ (\ x'_k = x''_k \ \wedge\ e'_k \leq e''_k\ ) \end{aligned}$$

- As a consequence :

$$\exists k \in \{1 \dots J\} \mid x''_k > x'_k \ \Rightarrow\ \ S' \nprec S''$$

DP Algorithm 2 (implementation details)

- Like in the previous algorithm :
  - label correcting
  - we keep track of each state's predecessors
  - states with lower t are extended first

Dominance checks :

- States that are ready to be extended are kept in a pool Q
- A state S can be added to Q only if $\nexists S' \in Q \mid S' \prec S$
- When S is added to Q then we set $Q = Q - \{\ X \in Q \mid S \prec X\ \}$
- Computing all the required dominance checks every time we have to add a new state to the pool is too computationally expensive
- $\forall$ product $k \in \{1 \dots J\}$, $\forall$ station $i \in \{0 \dots M+1\}$ we keep :
  - a set $ON[k][i]$ of states currently $\in Q$ in which k is in i
  - a set $GEQ[k][i]$ of states currently $\in Q$ in which k is in a station $\geq i$
  - a set $LEQ[k][i]$ of states currently $\in Q$ in which k is in a station $\leq i$

- We are trying to add to Q a state

  $S = \langle\ t\,,\ x\,,\ x_k\ \ \forall k \in \{1 \dots J\}\ \ ,\ e_k\ \ \forall k \in \{1 \dots J\}\ \ \rangle$
- if $\exists S' \in \bigcap_{k=1}^{J} ON\,[k][x_k] \mid S' \prec S$, then S cannot be added to Q
- if $\exists S' \in \bigcap_{k=1}^{J} GEQ\,[k][x_k] \mid S' \prec S$, then S cannot be added to Q
- otherwise S is added to Q
- $\forall S' \in \bigcap_{k=1}^{J} ON\,[k][x_k] \mid S \prec S'$ must be removed from Q
- $\forall S' \in \bigcap_{k=1}^{J} LEQ\,[k][x_k] \mid S \prec S'$ must be removed from Q
- the vast majority of state non-pushes and removals are consequent to checks involving ON
- IDs associated to states $\in Q$ and use of bitsets : advantages and drawbacks

DP Algorithm 2 (sample instance solution)

- DP algorithm 2 applied to the sample instance
  finds an optimal AGV route represented by this sequence of states :

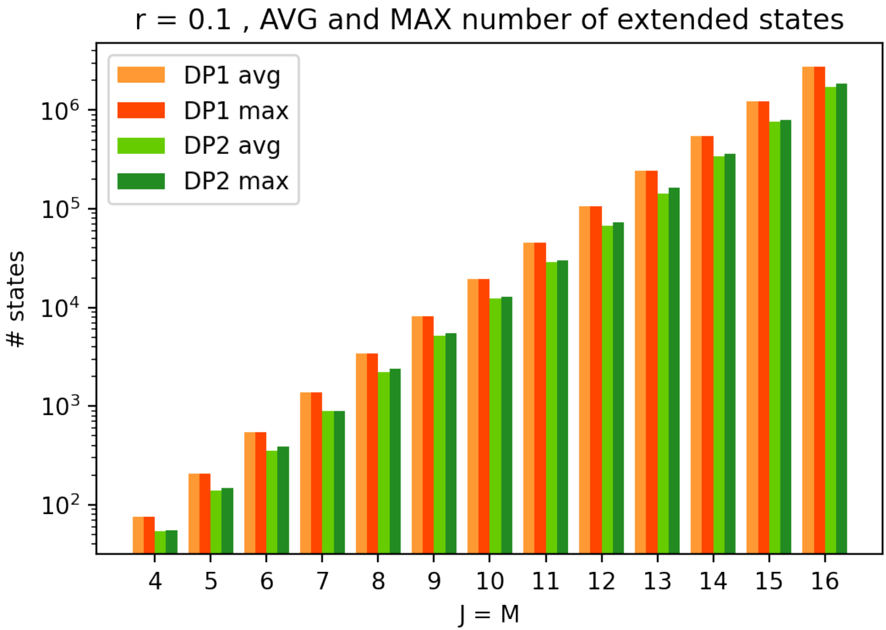
t	x	x_1	x_2	x_3	x_4	e_1	e_2	e_3	e_4
0	0	0	0	0	0	0	0	0	0
25	1	0	0	0	1	50	50	50	36
54	2	0	0	0	2	70	70	70	108
95	1	0	0	1	2	120	120	106	113
129	3	0	0	1	3	144	144	147	133
156	4	0	0	1	4	175	175	173	$\perp$
191	2	0	0	2	4	207	207	245	$\perp$
232	1	0	1	2	4	257	243	250	$\perp$
266	3	0	1	3	4	281	284	270	$\perp$
293	4	0	1	4	4	312	310	$\perp$	$\perp$
328	2	0	2	4	4	344	382	$\perp$	$\perp$
369	1	1	2	4	4	380	387	$\perp$	$\perp$
403	3	1	3	4	4	421	407	$\perp$	$\perp$
439	2	2	3	4	4	493	455	$\perp$	$\perp$
478	4	2	4	4	4	503	$\perp$	$\perp$	$\perp$
519	3	3	4	4	4	523	$\perp$	$\perp$	$\perp$
546	4	4	4	4	4	$\perp$	$\perp$	$\perp$	$\perp$

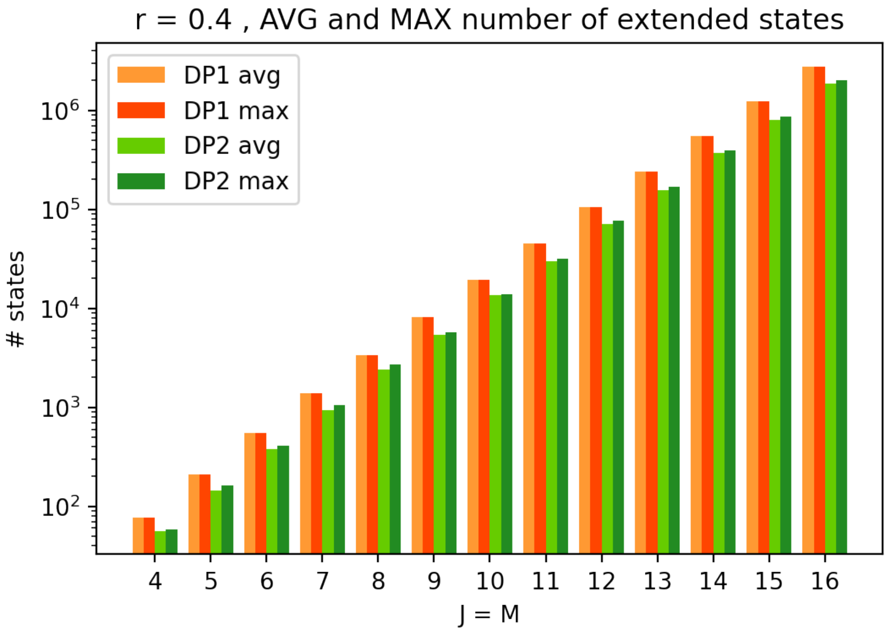
Instances generation

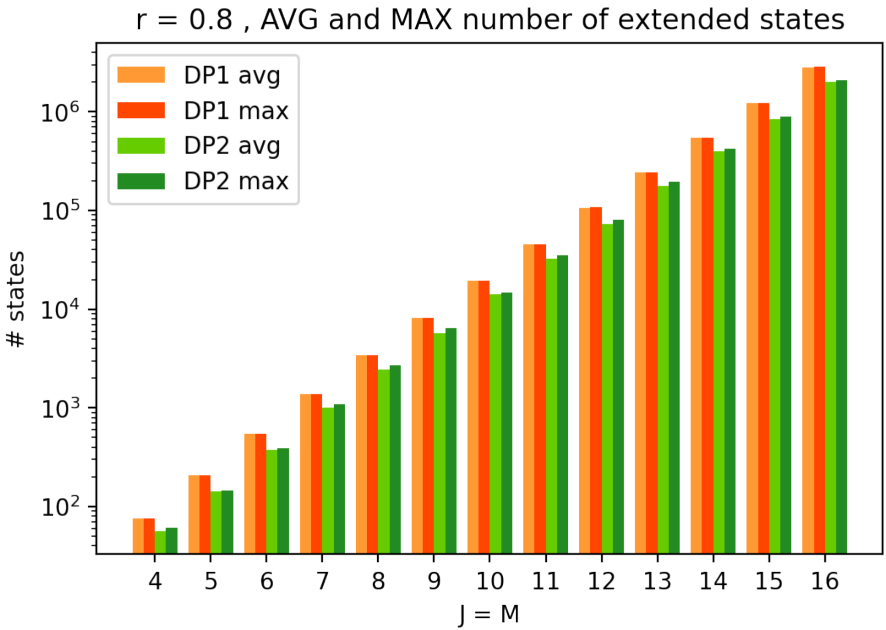
- very similar to what is explained in the reference article
- we vary J and M
- moving times t_{ij} are uniformly generated in an interval $[\, t_{MIN} \, , \, t_{MAX} \,]$
- we set $t_{MIN} = 15$ and $t_{MAX} = 25$
  - the triangular inequality automatically holds because $2 \cdot t_{MIN} \geq t_{MAX}$
  - average moving time is therefore 20
- we vary a parameter
  $r \in \{\, 0.1 \, , \, 0.4 \, , \, 0.8 \, , \, 1.1 \, , \, 1.4 \, , \, 1.8 \, , \, 2.1 \, , \, 2.5 \, , \, 3.0 \, , \, 3.5 \, , \, 4.0 \,\}$
- processing times p_i are uniformly generated in an interval $[\, 1 \, , \, r \cdot 40 \,]$
  - average processing time is therefore $r \cdot 20$
- 5 instances for each combination of J, M, r

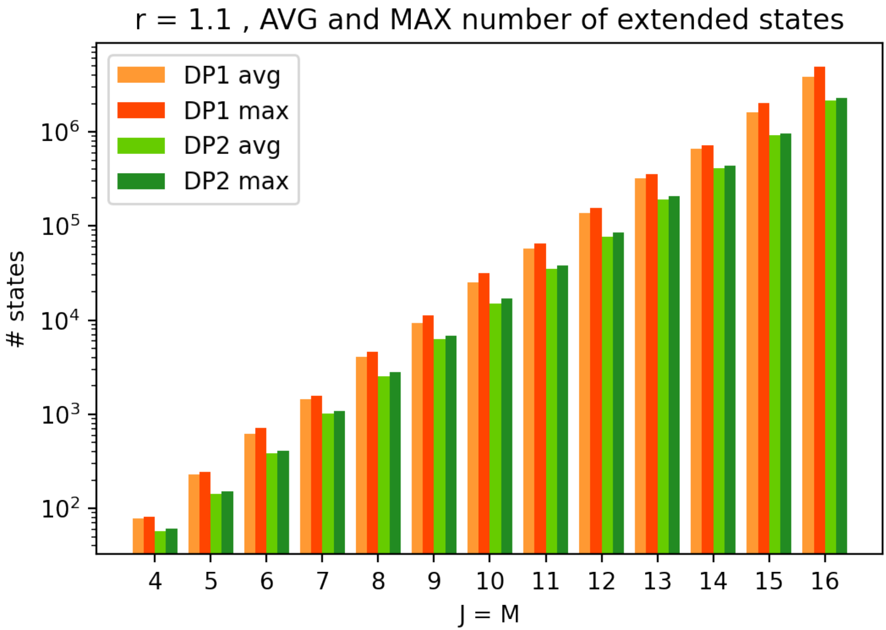
Computational results

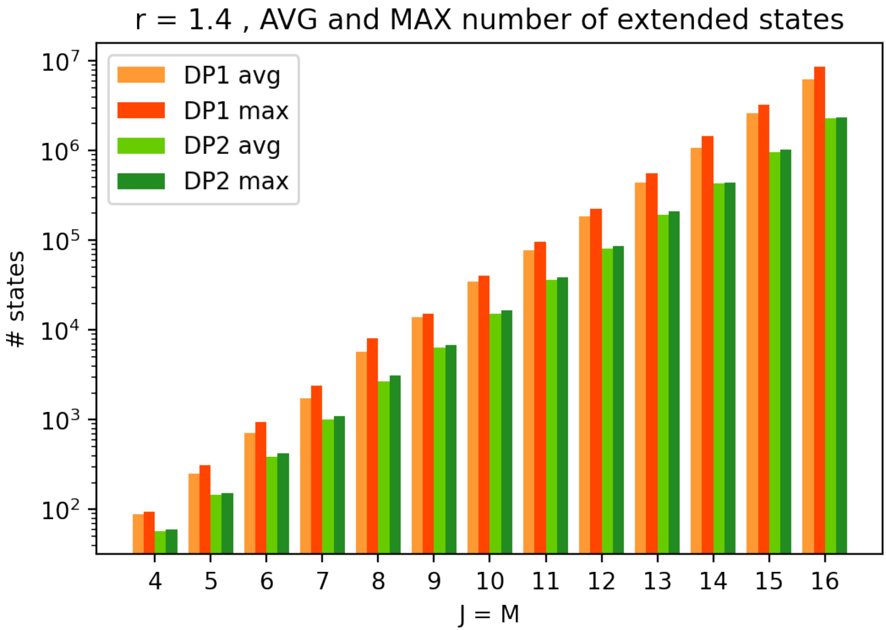
- AVG and MAX number of extended states
  - DP1 vs. DP2 for each r
  - DP2 percentual improvement wrt DP1
  - DP2 MAX number of states simultaneously in the pool
- AVG and MAX execution time
  - DP1 vs. DP2 for each r
  - DP2 percentual improvement wrt DP1
- AVG and MAX execution time
  - fixing $M = 10$ and varying $J \in \{4 \ldots 16\}$
  - fixing $J = 10$ and varying $M \in \{4 \ldots 16\}$
- the statistics for a given combination of J, M, r are only visible in the barplot if all 5 instances had an execution time < 15 minutes

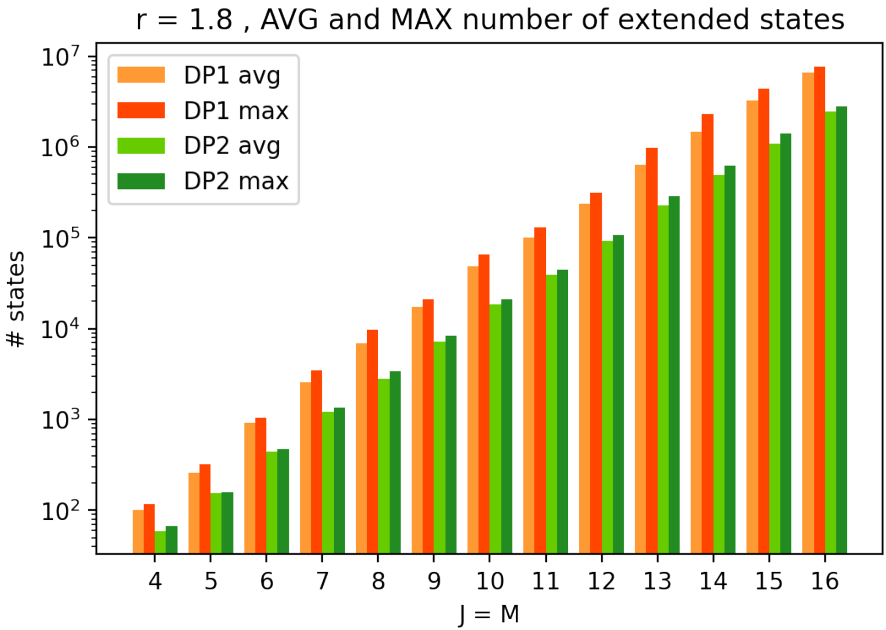

r = 0.1 , AVG and MAX number of extended states
DP1 avg
DP1 max
DP2 avg
DP2 max
states
10^2
10^3
10^4
10^5
10^6
4 5 6 7 8 9 10 11 12 13 14 15 16
J = M

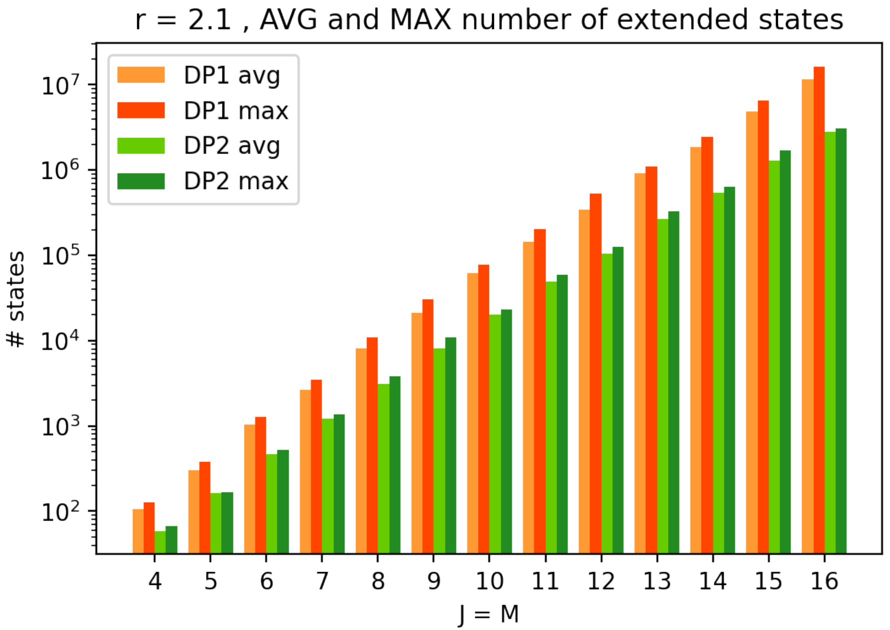

r = 0.4 , AVG and MAX number of extended states
DP1 avg
DP1 max
DP2 avg
DP2 max
states
10^6
10^5
10^4
10^3
10^2
4
5
6
7
8
9
10
11
12
13
14
15
16
J = M

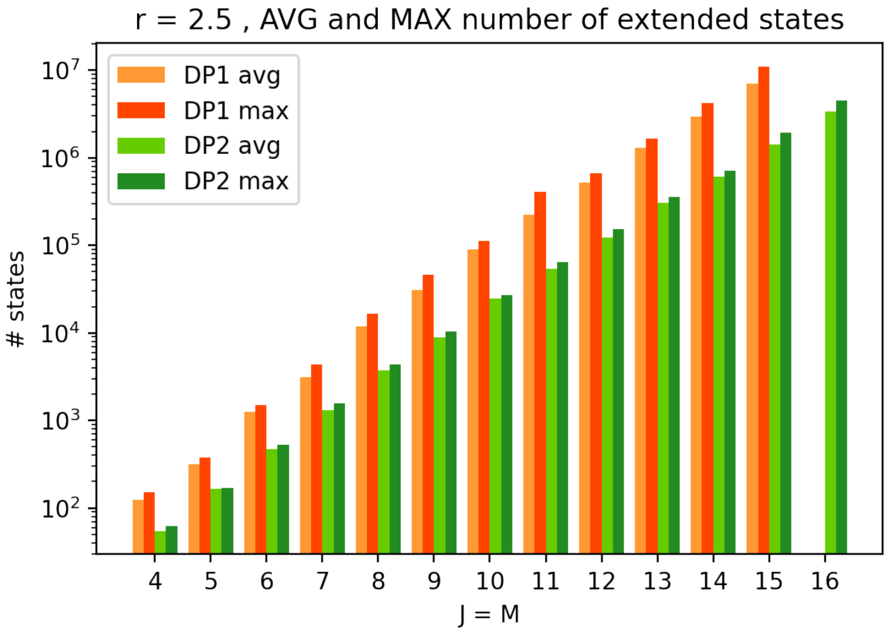

r = 0.8 , AVG and MAX number of extended states
DP1 avg
DP1 max
DP2 avg
DP2 max
states
10^2
10^3
10^4
10^5
10^6
4
5
6
7
8
9
10
11
12
13
14
15
16
J = M

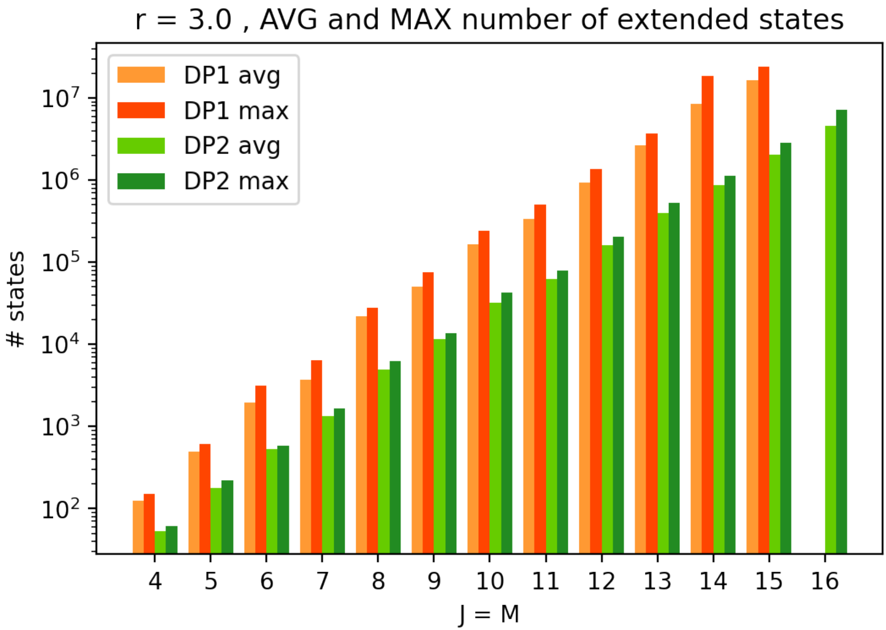

r = 1.1 , AVG and MAX number of extended states
DP1 avg
DP1 max
DP2 avg
DP2 max
states
10^2
10^3
10^4
10^5
10^6
4
5
6
7
8
9
10
11
12
13
14
15
16
J = M

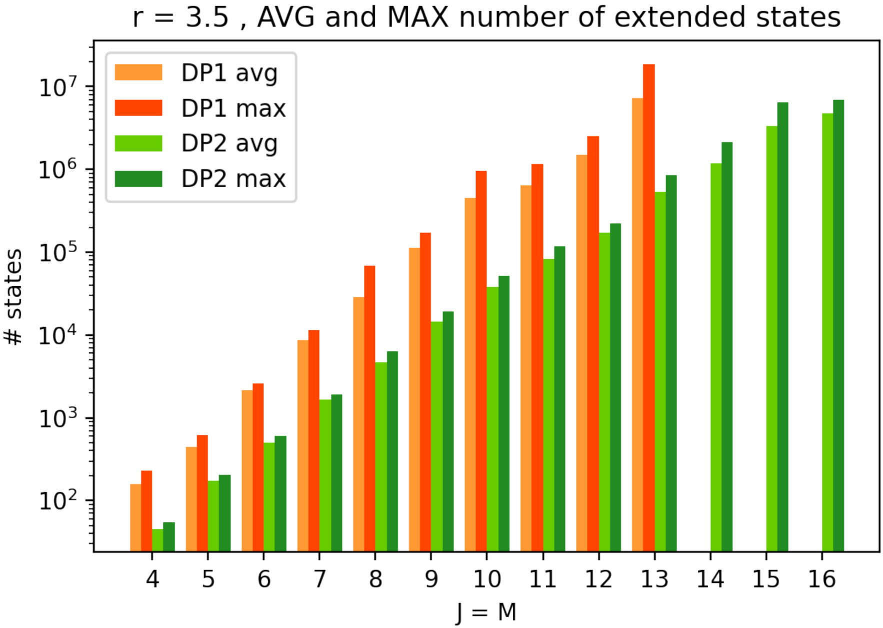

r = 1.4 , AVG and MAX number of extended states
DP1 avg
DP1 max
DP2 avg
DP2 max
states
10^7
10^6
10^5
10^4
10^3
10^2
4
5
6
7
8
9
10
11
12
13
14
15
16
J = M

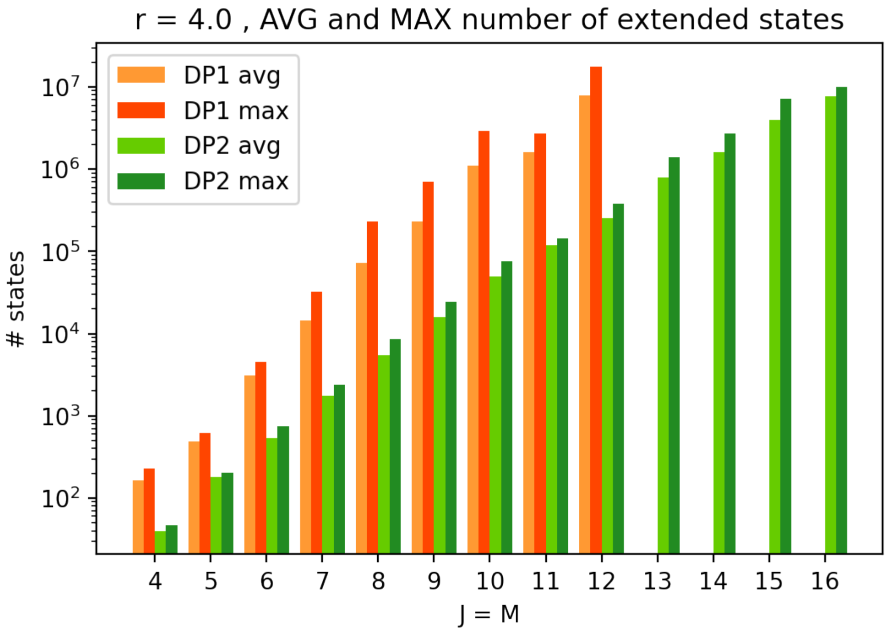

r = 1.8 , AVG and MAX number of extended states
DP1 avg
DP1 max
DP2 avg
DP2 max
states
10^2
10^3
10^4
10^5
10^6
10^7
4
5
6
7
8
9
10
11
12
13
14
15
16
J = M


r = 2.1 , AVG and MAX number of extended states
DP1 avg
DP1 max
DP2 avg
DP2 max
states
10^2
10^3
10^4
10^5
10^6
10^7
4
5
6
7
8
9
10
11
12
13
14
15
16
J = M


r = 2.5 , AVG and MAX number of extended states
DP1 avg
DP1 max
DP2 avg
DP2 max
states
10^7
10^6
10^5
10^4
10^3
10^2
4
5
6
7
8
9
10
11
12
13
14
15
16
J = M


r = 3.0 , AVG and MAX number of extended states
DP1 avg
DP1 max
DP2 avg
DP2 max
states
10^2
10^3
10^4
10^5
10^6
10^7
4 5 6 7 8 9 10 11 12 13 14 15 16
J = M


r = 3.5 , AVG and MAX number of extended states
DP1 avg
DP1 max
DP2 avg
DP2 max
states
10^2
10^3
10^4
10^5
10^6
10^7
4
5
6
7
8
9
10
11
12
13
14
15
16
J = M



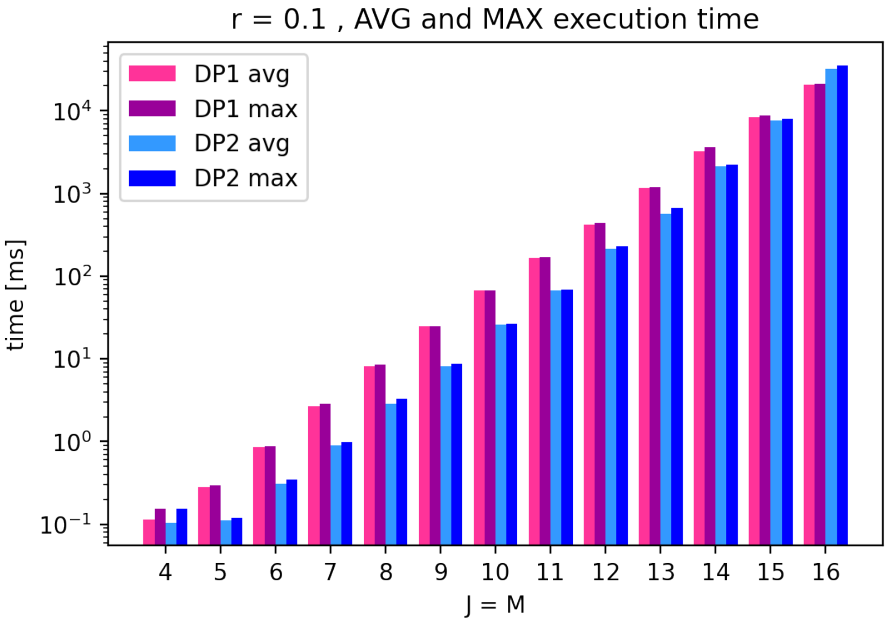
r = 4.0 , AVG and MAX number of extended states
DP1 avg
DP1 max
DP2 avg
DP2 max
states
10^2
10^3
10^4
10^5
10^6
10^7
4
5
6
7
8
9
10
11
12
13
14
15
16
J = M

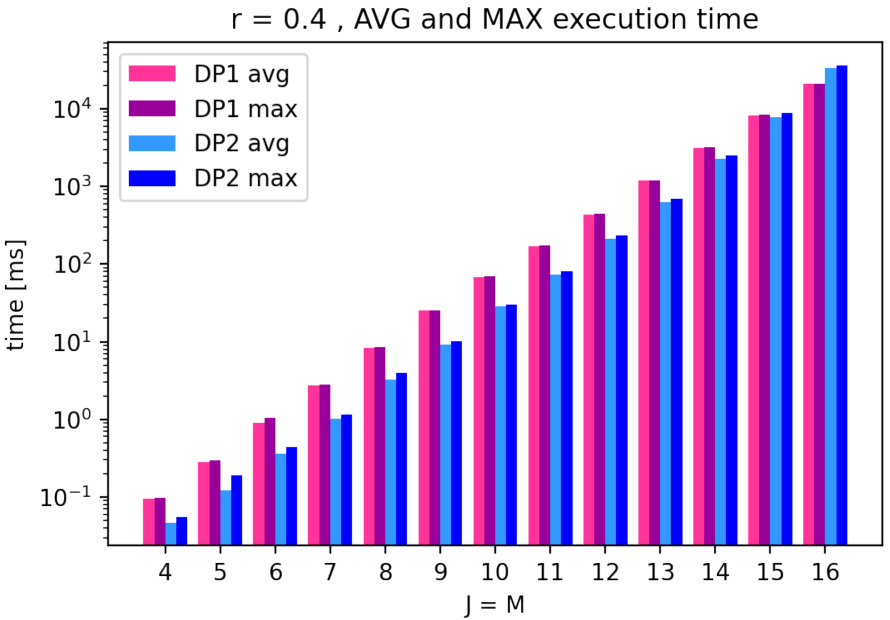
- DP2 percentual improvement wrt DP1
  in terms of MAX number of extended states

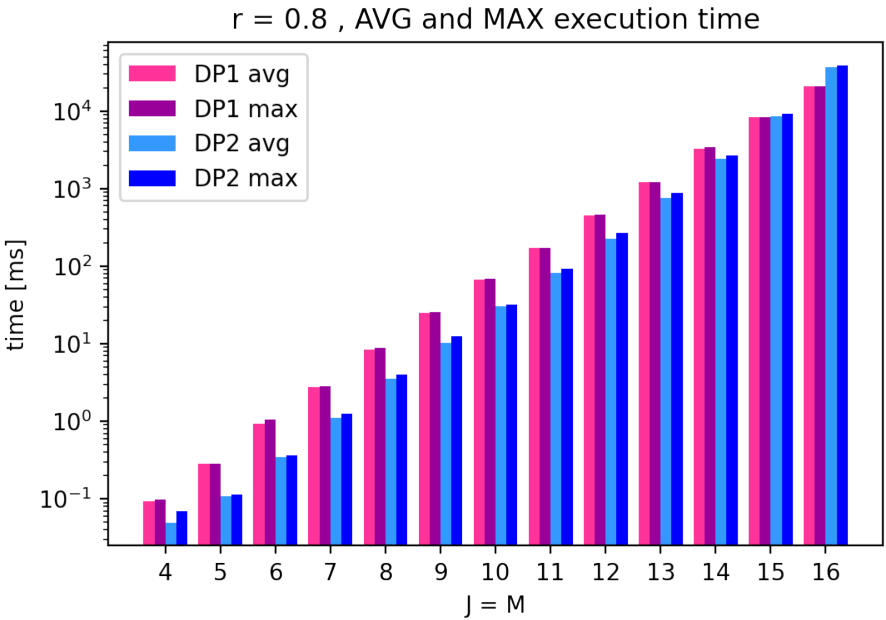
J = M	4	5	6	7	8	9	10	11	12	13	14	15	16
r = 0.1	0.28	0.28	0.28	0.35	0.29	0.33	0.35	0.34	0.31	0.32	0.34	0.36	0.33
r = 0.4	0.24	0.22	0.25	0.24	0.20	0.30	0.28	0.30	0.27	0.30	0.29	0.30	0.27
r = 0.8	0.20	0.30	0.29	0.21	0.20	0.21	0.24	0.22	0.25	0.18	0.23	0.28	0.27
r = 1.1	0.25	0.38	0.43	0.31	0.39	0.39	0.47	0.42	0.45	0.42	0.40	0.52	0.54
r = 1.4	0.37	0.50	0.55	0.54	0.62	0.55	0.59	0.60	0.61	0.62	0.70	0.68	0.73
r = 1.8	0.42	0.50	0.54	0.61	0.65	0.60	0.68	0.66	0.65	0.71	0.73	0.68	0.63
r = 2.1	0.47	0.55	0.59	0.61	0.65	0.64	0.71	0.71	0.76	0.70	0.74	0.74	0.81
r = 2.5	0.59	0.55	0.65	0.65	0.74	0.78	0.76	0.84	0.77	0.79	0.83	0.82	
r = 3.0	0.59	0.64	0.81	0.74	0.78	0.82	0.82	0.84	0.85	0.86	0.94	0.88	
r = 3.5	0.76	0.67	0.77	0.83	0.91	0.89	0.95	0.90	0.91	0.95			
r = 4.0	0.79	0.67	0.84	0.93	0.96	0.97	0.97	0.95	0.98	0.00			

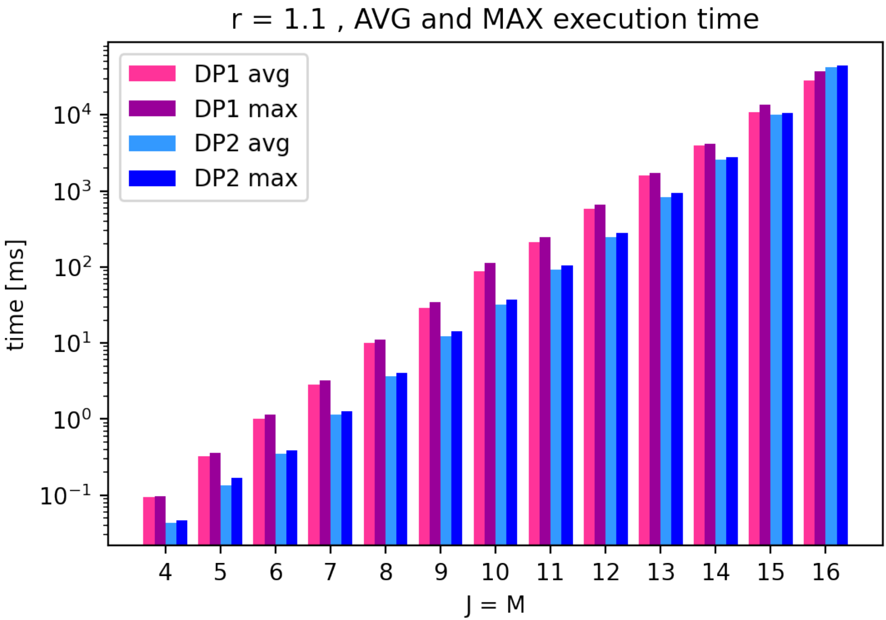
- DP2 MAX number of states simultaneously in the pool

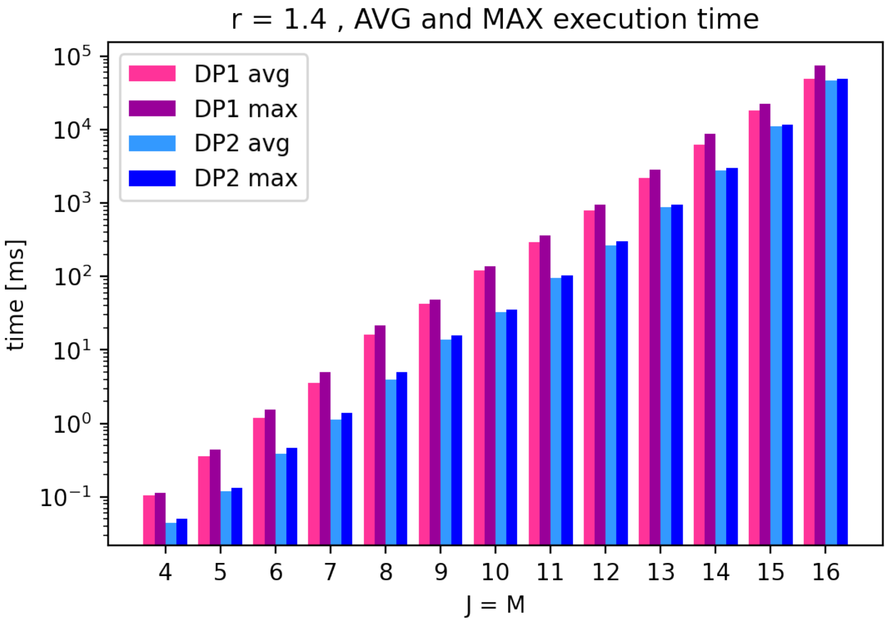
J = M	4	5	6	7	8	9	10	11	12	13	14	15	16
r = 0.1	6	12	21	38	78	146	279	553	1249	2228	4427	8829	17397
r = 0.4	7	13	22	42	77	134	264	507	1087	1996	4121	7996	15743
r = 0.8	7	12	22	44	80	151	284	528	1151	2168	4197	7755	15890
r = 1.1	8	12	25	43	84	161	314	598	1102	2290	4227	8545	17186
r = 1.4	8	13	24	44	97	165	308	628	1151	2316	4594	9116	17911
r = 1.8	8	13	28	59	109	203	415	678	1465	3230	6126	12455	21202
r = 2.1	8	18	32	59	119	277	442	893	1737	3695	6289	14974	24215
r = 2.5	9	18	33	67	145	261	524	1068	2037	3947	7449	18223	34396
r = 3.0	9	20	46	78	227	370	841	1325	2861	6250	11859	25580	57704
r = 3.5	7	20	57	95	228	570	1123	2030	3225	10140	21756	61341	52958
r = 4.0	8	25	57	124	335	777	1642	2756	5439	18024	28425	67892	76244

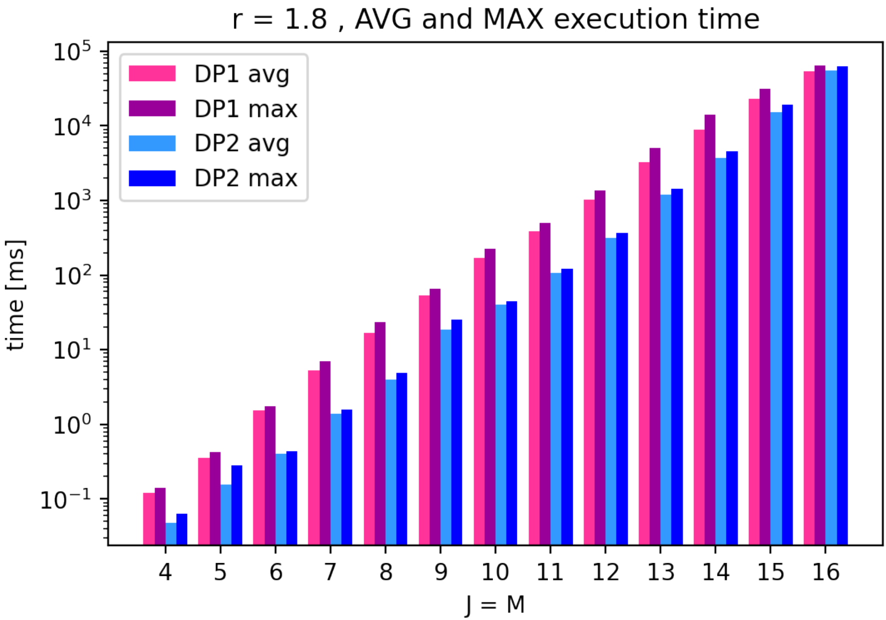

r = 0.1 , AVG and MAX execution time
DP1 avg
DP1 max
DP2 avg
DP2 max
time [ms]
10^{-1}
10^{0}
10^{1}
10^{2}
10^{3}
10^{4}
4
5
6
7
8
9
10
11
12
13
14
15
16
J = M

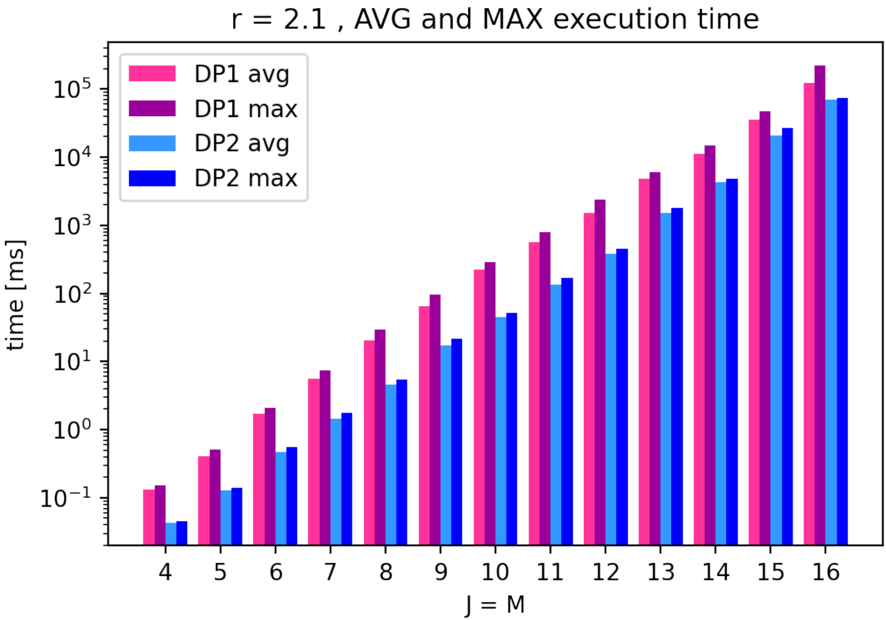

r = 0.4 , AVG and MAX execution time
DP1 avg
DP1 max
DP2 avg
DP2 max
10^4
10^3
10^2
10^1
10^0
10^{-1}
time [ms]
4
5
6
7
8
9
10
11
12
13
14
15
16
J = M

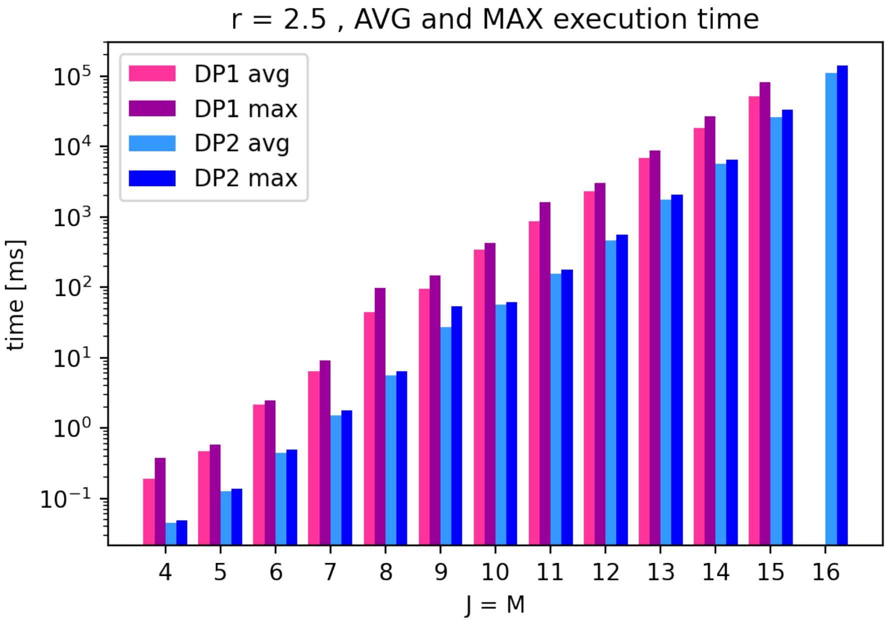

r = 0.8 , AVG and MAX execution time
DP1 avg
DP1 max
DP2 avg
DP2 max
time [ms]
10^{-1}
10^{0}
10^{1}
10^{2}
10^{3}
10^{4}
4
5
6
7
8
9
10
11
12
13
14
15
16
J = M

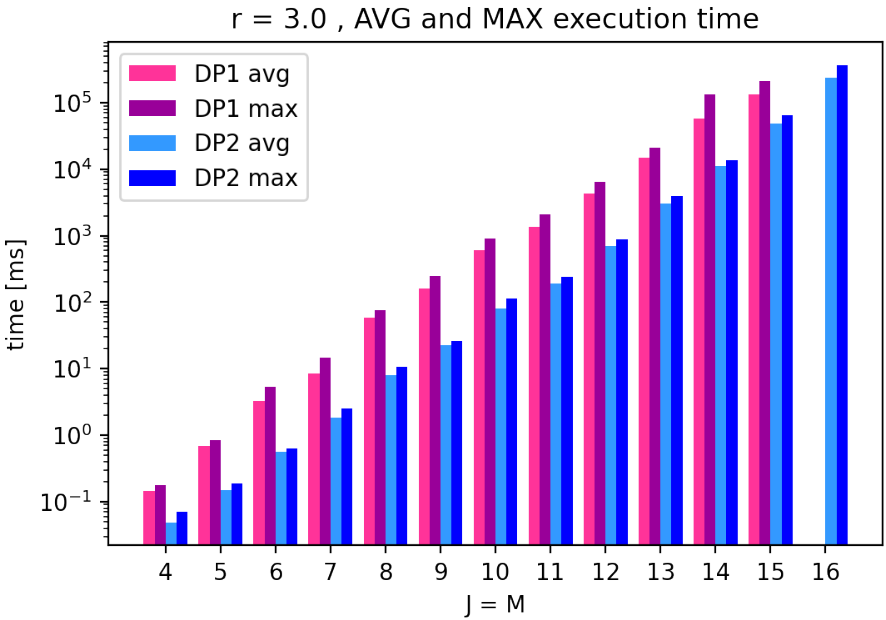

r = 1.1 , AVG and MAX execution time
DP1 avg
DP1 max
DP2 avg
DP2 max
time [ms]
10^{-1}
10^{0}
10^{1}
10^{2}
10^{3}
10^{4}
4
5
6
7
8
9
10
11
12
13
14
15
16
J = M

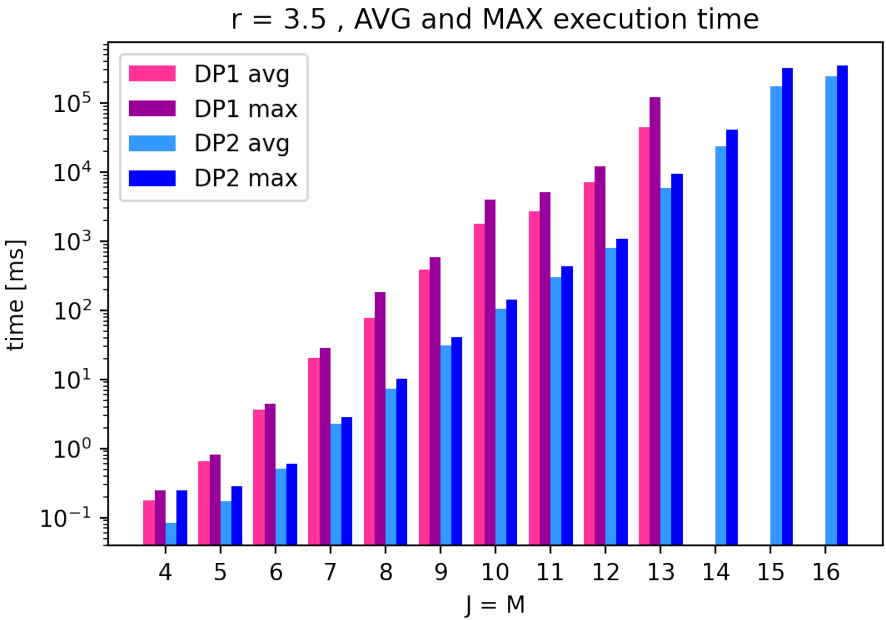

r = 1.4 , AVG and MAX execution time
DP1 avg
DP1 max
DP2 avg
DP2 max
time [ms]
10^5
10^4
10^3
10^2
10^1
10^0
10^{-1}
4
5
6
7
8
9
10
11
12
13
14
15
16
J = M

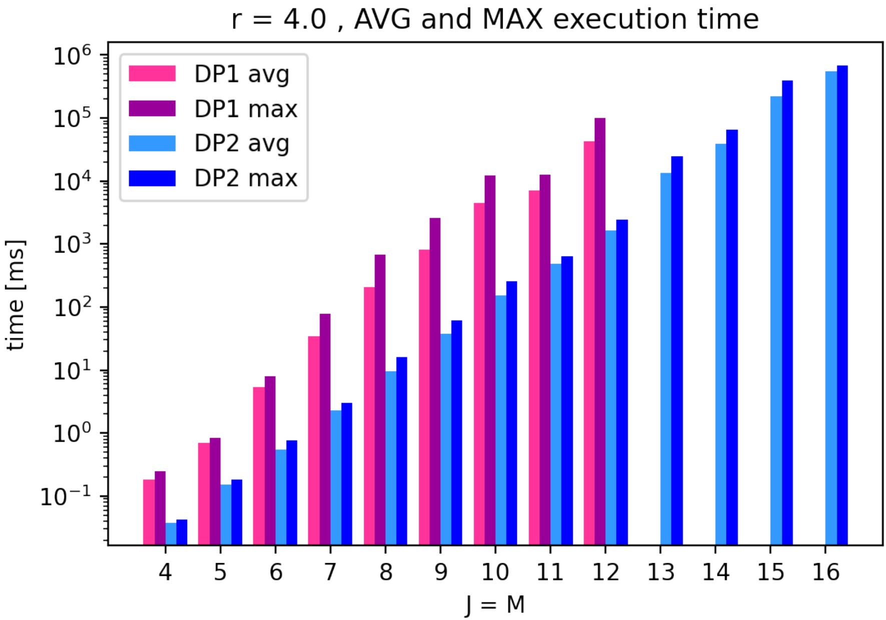

r = 1.8 , AVG and MAX execution time
DP1 avg
DP1 max
DP2 avg
DP2 max
time [ms]
10^5
10^4
10^3
10^2
10^1
10^0
10^{-1}
4 5 6 7 8 9 10 11 12 13 14 15 16
J = M


r = 2.1 , AVG and MAX execution time
DP1 avg
DP1 max
DP2 avg
DP2 max
time [ms]
10^5
10^4
10^3
10^2
10^1
10^0
10^-1
4
5
6
7
8
9
10
11
12
13
14
15
16
J = M


r = 2.5 , AVG and MAX execution time
DP1 avg
DP1 max
DP2 avg
DP2 max
time [ms]
10^{-1}
10^0
10^1
10^2
10^3
10^4
10^5
4
5
6
7
8
9
10
11
12
13
14
15
16
J = M

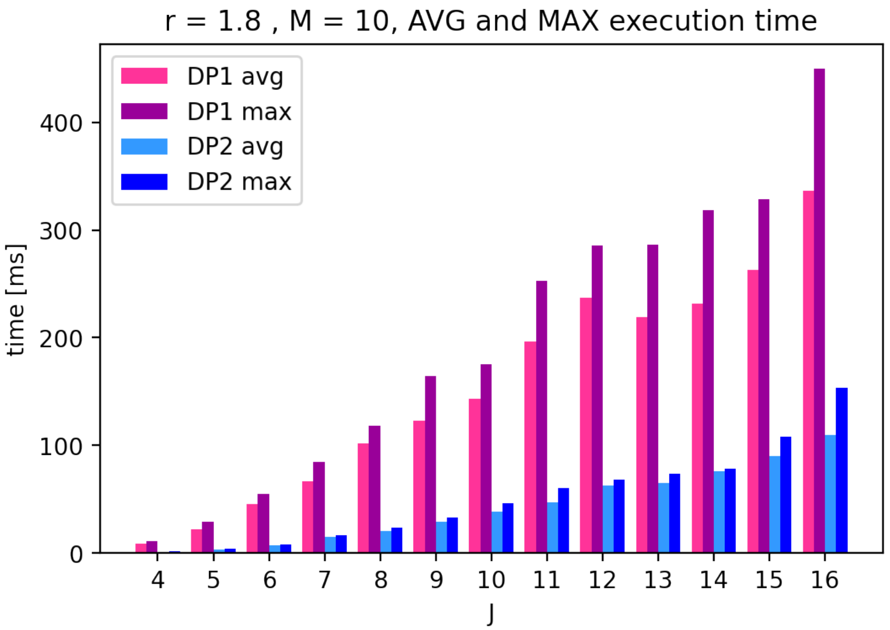

r = 3.0 , AVG and MAX execution time
DP1 avg
DP1 max
DP2 avg
DP2 max
time [ms]
10^5
10^4
10^3
10^2
10^1
10^0
10^{-1}
4
5
6
7
8
9
10
11
12
13
14
15
16
J = M

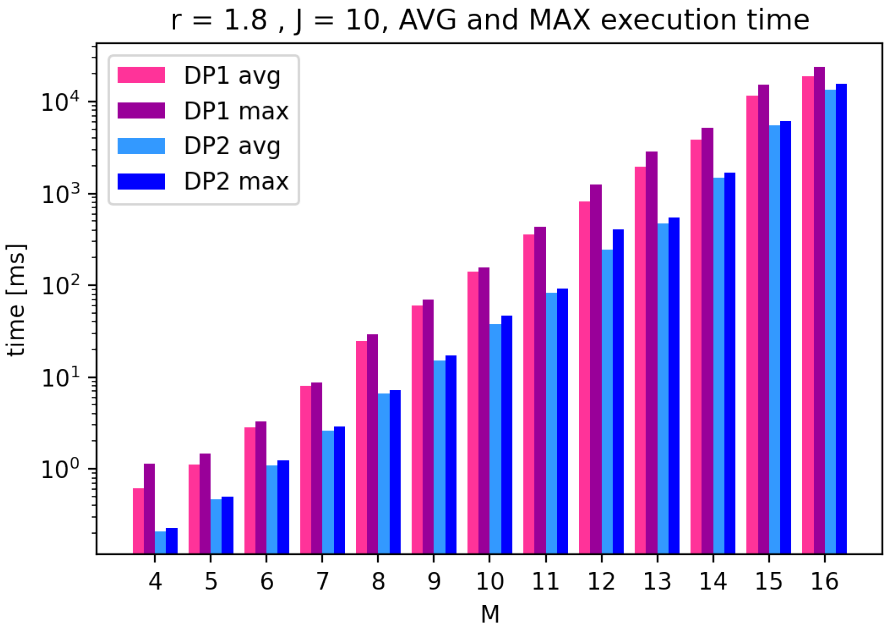

r = 3.5 , AVG and MAX execution time
DP1 avg
DP1 max
DP2 avg
DP2 max
time [ms]
10^{-1}
10^{0}
10^{1}
10^{2}
10^{3}
10^{4}
10^{5}
4
5
6
7
8
9
10
11
12
13
14
15
16
J = M


r = 4.0 , AVG and MAX execution time
DP1 avg
DP1 max
DP2 avg
DP2 max
time [ms]
J = M
10^{6}
10^{5}
10^{4}
10^{3}
10^{2}
10^{1}
10^{0}
10^{-1}
4 5 6 7 8 9 10 11 12 13 14 15 16

- DP2 percentual improvement wrt DP1 in terms of MAX execution time

J = M	4	5	6	7	8	9	10	11	12	13	14	15	16
r = 0.1	0.01	0.59	0.61	0.65	0.62	0.65	0.61	0.59	0.48	0.44	0.37	0.07	**-0.69**
r = 0.4	0.44	0.35	0.58	0.59	0.53	0.60	0.57	0.54	0.48	0.42	0.21	**-0.04**	**-0.71**
r = 0.8	0.29	0.60	0.65	0.56	0.55	0.51	0.53	0.47	0.42	0.28	0.21	**-0.10**	**-0.85**
r = 1.1	0.53	0.54	0.67	0.61	0.63	0.59	0.67	0.58	0.58	0.46	0.34	0.23	**-0.21**
r = 1.4	0.55	0.70	0.70	0.72	0.77	0.68	0.74	0.72	0.69	0.66	0.66	0.49	0.34
r = 1.8	0.55	0.33	0.75	0.77	0.79	0.61	0.80	0.76	0.73	0.72	0.68	0.38	0.03
r = 2.1	0.70	0.73	0.73	0.76	0.81	0.77	0.82	0.78	0.81	0.70	0.68	0.43	0.66
r = 2.5	0.87	0.76	0.80	0.81	0.93	0.64	0.85	0.89	0.81	0.76	0.75	0.59	
r = 3.0	0.60	0.78	0.88	0.82	0.86	0.89	0.88	0.88	0.86	0.81	0.90	0.69	
r = 3.5	0.02	0.66	0.87	0.90	0.95	0.93	0.96	0.91	0.91	0.92			
r = 4.0	0.83	0.78	0.90	0.96	0.98	0.98	0.98	0.95	0.98				


r = 1.8 , M = 10, AVG and MAX execution time
DP1 avg
DP1 max
DP2 avg
DP2 max
time [ms]
0
100
200
300
400
4
5
6
7
8
9
10
11
12
13
14
15
16
J


r = 1.8 , J = 10, AVG and MAX execution time
DP1 avg
DP1 max
DP2 avg
DP2 max
time [ms]
10^4
10^3
10^2
10^1
10^0
4
5
6
7
8
9
10
11
12
13
14
15
16
M