Optimal index policies for multi-product make-to-stock queues: if resources are more costly would we use less?

D. J. Hodge & K. D. Glazebrook

Lancaster University, UK

November 20th 2009

Replenishment models in make-to-stock queues

Divisible resources

- Replenishment control problem (holding costs and backorder penalties)
- Arrival controlled make-to-stock queue
- Many types of product, k, and numerous machines
- We allow for limited backorders, and limited inventory stockpiling
- Markovian, exponential distributions approach via Stochastic Dynamic Programming
- Objective: long-run cost minimization.

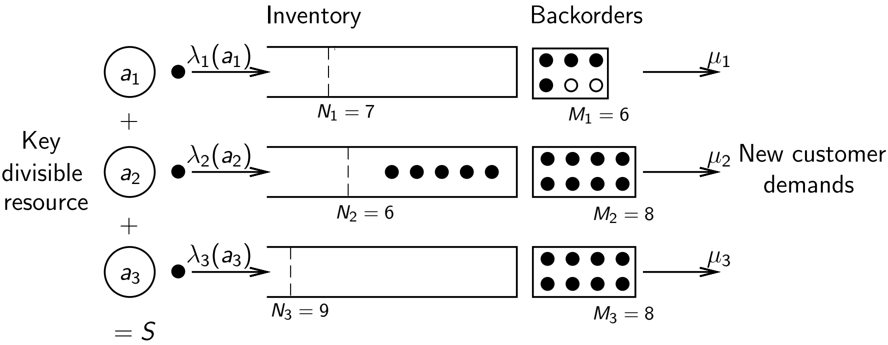
$$\{\text{Instantaneous cost rate in state } i\} = hi^+ + bi^- + D\mu\mathbb{I}(i = -M)$$

- Numerous natural applications

Results with non-divisible resources

- Ha (1997): hedging point and switching curve optimality, with two identical demand products
- de Vericourt, Karaesman, & Dallery (2000): distinct $b\mu$ values, hedging point optimality
- Zheng & Zipkin (1990) and Zipkin (1995): centralized policy better than local demands served FCFS
- Veatch & Wein (1996): Indices good for lost sales, not great for idling. Perez & Zipkin's myopic heuristic performed well. No indices for backorders.

The Model



System state $(-2, 5, 0)$

Simplification to a single product problem

- As formulated, the K product problem is very hard to solve
- Can be seen as a restless bandit, since all queues evolve while some are being replenished

Our approach: reduction to a single-product subproblem. Two natural approaches lead to same single-product problem:

1. Introduce a per unit time cost (W) for machine usage, or
2. Whittle's Lagrangian relaxation of the multi-product problem (discussed later)

DP equation and a change of variables

- Easy to form DP
- Optimal policy $\iff$ Satisfies DP

Uniformization $\Rightarrow$

$$\Delta^{\pi}_{i+1}(\lambda(\pi(i)) - \lambda(\pi(i)+1)) \leq W \leq \Delta^{\pi}_{i+1}(\lambda(\pi(i)-1) - \lambda(\pi(i)))$$

Key Idea

Introduce a scalar c on all inventory and backorder costs, and set $W = 1$. Clearly a problem with W is equivalent to $c = W^{-1}$. Why is it fruitful?

Now we can target our monotonicities ...

Objectives for the single product problem

State monotonicity

If we increase our inventory level do we desire to use fewer machines?

Cost monotonicity

If we increase the replenishment-costs do we use less replenishment? Yes, certainly on average.
What about state-wise? This is indexability.

Non-monotonicity in state

Take $N = M = 10$, $S = 15$, $h = 0$, $b = 1.5$, $D = 50$, $\mu = 0.6$. The production rate model is given by

$$\lambda(a) = \frac{0.8a}{1+a} + 0.05$$

The uniquely optimal policy $\bar{P}(c)$ in the stationary class when $c = 0.0613$ is given by

j	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	7	8	9	10
$\pi^*(j)$	1	1	2	2	2	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	0

- Why does this happen?

Monotonicity in state

Theorem (State monotonicity)

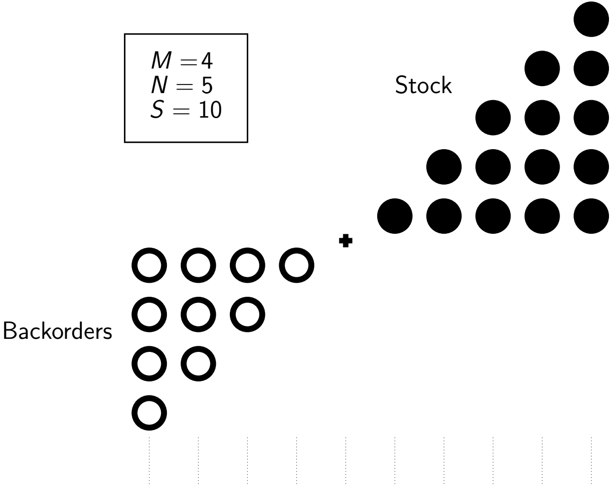
For $\lambda(S) > \mu$, fixed model parameters, then there exists D^* such that

$$D > D^* \implies \textit{State Monotonicity}$$

Furthermore

As we increase c the D we have looks relatively bigger and thus monotonicity occurs for all large enough c.

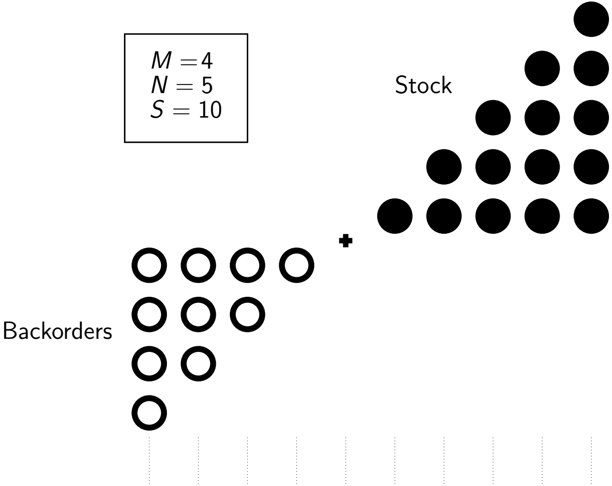
$M = 4$
$N = 5$
$S = 10$



Optimal policy	10	10	10	9	8	8	7	6	4	0

$c = 3.45$

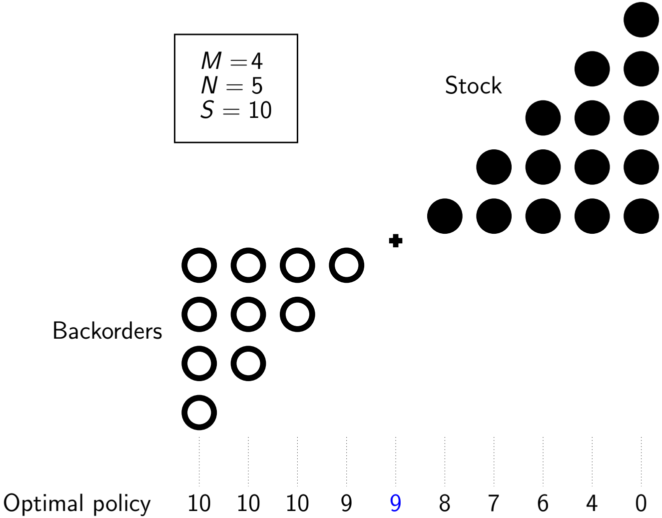
$M = 4$
$N = 5$
$S = 10$



Optimal policy	10	10	10	9	8	8	7	6	4	0
					9					

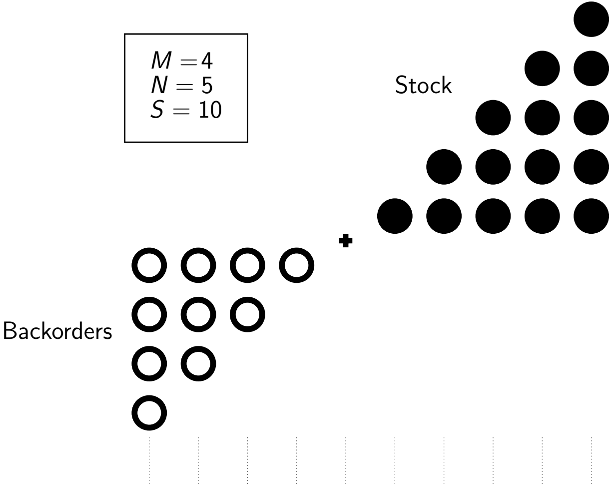
$$c = 3.4568$$

$M = 4$
$N = 5$
$S = 10$



$$c = 3.4569$$

$M = 4$
$N = 5$
$S = 10$

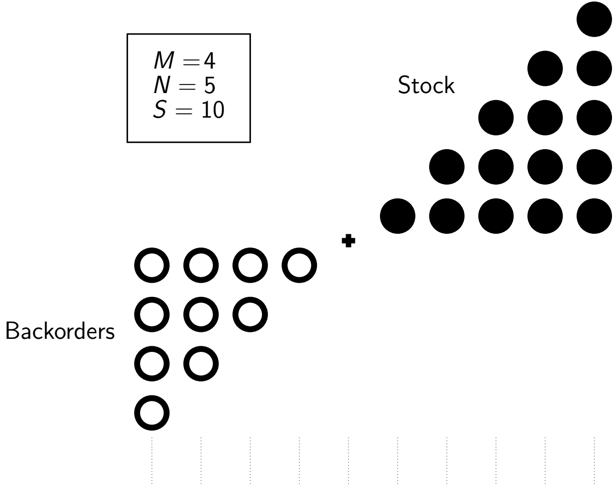


Optimal policy 10 10 10 9 9 8 7 6 4 0

7

$$c = 3.4713$$

$M = 4$
$N = 5$
$S = 10$



Optimal policy	10	10	10	9	9	8	7	7	4	0

$$c = 3.4714$$

Non-monotonicity in c

non-indexability

Take $S = 25$, $h = 0.05$, $b = 1.5$, $D = 50$, $\mu = 0.65$. Same convex form for $\lambda(a)$ (reciprocal).

The unique optimal stationary policy for $\bar{P}(c)$ is computed at four values of c as follows:

j	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	7	8	9	10	
$\pi^*(j)$	25	25	25	25	25	25	23	20	18	16	12	10	8	6	5	4	3	2	1	1	0	$c = 26.4644$
$\pi^*(j)$	25	25	25	25	25	25	23	21	18	16	12	10	8	6	5	4	3	2	1	1	0	$c = 26.6114$
$\pi^*(j)$	25	25	25	25	25	25	23	21	18	16	12	10	8	6	5	4	3	2	1	0	0	$c = 26.7760$
$\pi^*(j)$	25	25	25	25	25	25	23	21	18	16	13	10	8	6	5	4	3	2	1	0	0	$c = 27.0362$

Increasing c can make you use less resource! Why?

Monotonicity in c

Theorem (Cost monotonicity — i.e. Indexability)

For fixed system parameters, there exists $h^* > 0$ such that

$$0 \leq h < h^* \Longrightarrow \textit{Optimal Policy is state-wise increasing in } c$$

Furthermore, the provably sufficient h^* is larger than most reasonable values

Conclusion

For all small (reasonable) h, we have indexability

Application to lost sales

as a special case of backorders

- We can choose a maximal number of allowed backorders equal to 0
- Maximum backorder penalty $\mapsto$ Lost Sales penalty in state 0

Same results apply

Theorem (State Monotonicity)

For large enough Lost Sales penalty D, we have state monotonicity.

Theorem (Indexability)

For small holding costs h we have indexability

An algorithm for finding optimal policies

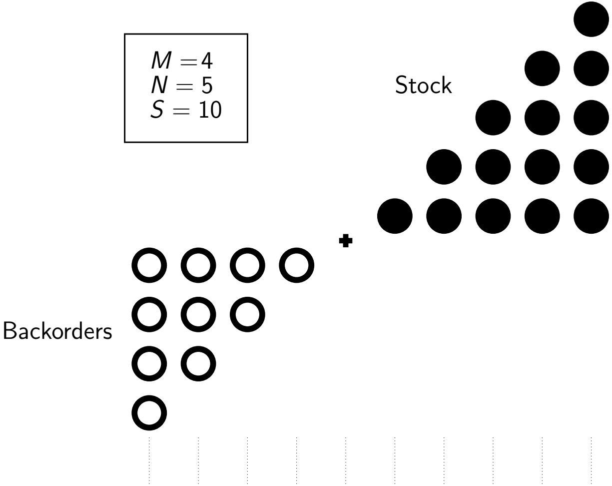
Our proof gives rise to easy to an implement algorithm

$$\Delta_{i+1}^{c}(\lambda(\pi(i)) - \lambda(\pi(i)+1)) \leq W \leq \Delta_{i+1}^{c}(\lambda(\pi(i)-1) - \lambda(\pi(i)))$$

Start from $c = 0$, incrementally find optimal policies for all c
Recall cost function has the form

$$c \times (\text{Backorder \& Inventory costs}) + \text{Machine costs}$$

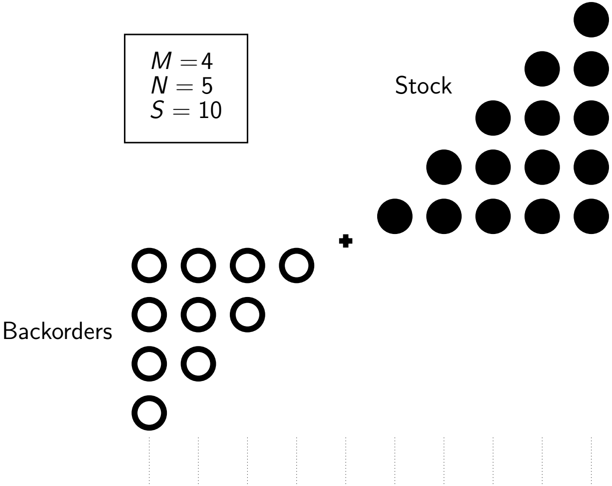
$M = 4$
$N = 5$
$S = 10$



Optimal policy 10 10 10 9 8 8 7 6 4 0

$c = 3.45$

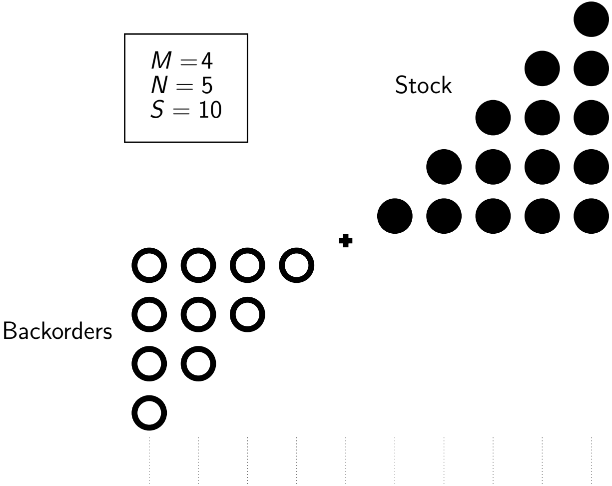
$M = 4$
$N = 5$
$S = 10$



Optimal policy	10	10	10	9	8	8	7	6	4	0
					9					

$$c = 3.4568$$

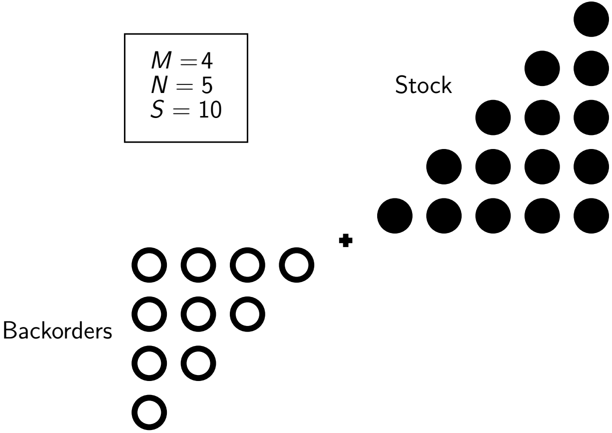
$M = 4$
$N = 5$
$S = 10$



Optimal policy	10	10	10	9	9	8	7	6	4	0

$c = 3.4569$

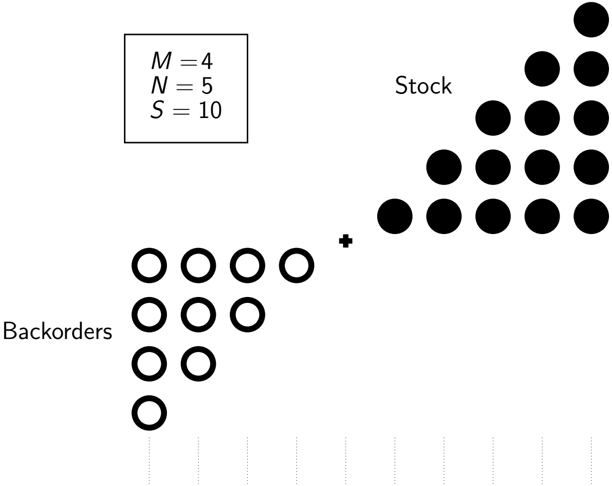
$M = 4$
$N = 5$
$S = 10$



Optimal policy	10	10	10	9	9	8	7	6	4	0
								7		

$c = 3.4713$

$$M = 4$$
$$N = 5$$
$$S = 10$$



Optimal policy	10	10	10	9	9	8	7	7	4	0

$$\boxed{c = 3.4714}$$

Back to the multi-product problem

- Restless bandit, generally very hard
- Our single product approach arises from Whittle's Lagrangian relaxation approach:
  - Relax the total number of machines
  - Introduce an average machine usage requirement instead
  - Lagrange multiplier W represents a cost per machine per unit time
  - Product-wise decomposition follows
- Other heuristics exist for $S = 1$: Zipkin & Perez, Wein & Veatch

Indices for products

The index heuristic

What are our Indices?

Values of c (or W) at which the optimal policy transitions. Equivalently, fair charges for the next unit of machinery in a state.

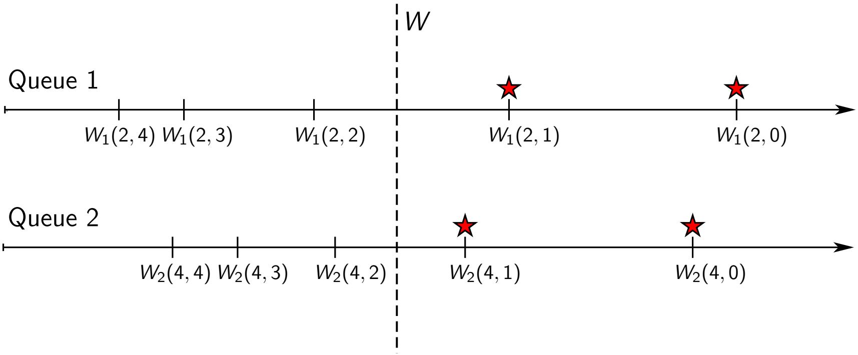
Greedy heuristic for the multi-product scenario

- Record the system state
- Allocate machines sequentially using index values (by the highest bidder) for each new machine
- Stop when all S machines allocated, or no queues will pay.

Performance?

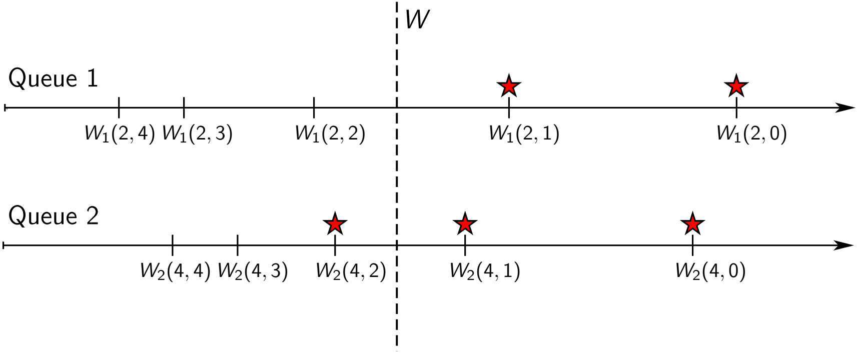
Why might indices perform well? Why might they fail?

Relaxed solution in action



$$\boxed{S = 5 \text{ and System state } (2,4)}$$

Index heuristic in action



$$\boxed{S = 5 \text{ and System state } (2,4)}$$

Numerical performance

Policies

- A simple myopic policy — maximize the rate of reduction of inventory costs performs very poorly
- Static policy — best fixed allocation of $\leq S$ servers between queues
- Greedy Index heuristic (see above)

We therefore look primarily at benefits of dynamic state-dependent allocations. Standard machine production rate

$$\lambda(a) = \lambda a(a+m)^{-1} + \epsilon$$

Fix $S = 25$, $M = 10$, $N = 10$, $D = 50$, $b = 1.5$, $h = 1/5000$, $K = 2$.

A backorder model

PARAMETERS						POLICIES			
						INDEX		STATIC	
λ_1	λ_2	μ_1	μ_2	M_1	M_2	MED	MAX	MED	MAX
1.5	1.5	(1,1.5)	(1,1.5)	(4,6)	(4,6)	**0.024**	**0.116**	**2.673**	**27.141**
1.5	3.0	(1,1.5)	(2,3.0)	(4,6)	(4,6)	**0.018**	**0.141**	**2.167**	**24.205**
1.5	4.5	(1,1.5)	(3,4.5)	(4,6)	(4,6)	**0.019**	**0.528**	**2.177**	**35.293**
1.2	1.2	(1,1.5)	(1,1.5)	(2,3)	(2,3)	**0.001**	**0.016**	**0.461**	**4.048**
1.2	2.4	(1,1.5)	(2,3.0)	(2,3)	(2,3)	**0.001**	**0.016**	**0.414**	**5.640**
1.2	3.6	(1,1.5)	(3,4.5)	(2,3)	(2,3)	**0.001**	**0.021**	**0.267**	**4.687**
1.5	1.2	(1,1.5)	(1,1.5)	(4,6)	(2,3)	**0.008**	**0.114**	**0.881**	**8.408**
3.0	1.2	(2,3.0)	(1,1.5)	(4,6)	(2,3)	**0.006**	**0.132**	**0.931**	**11.049**
1.5	2.4	(1,1.5)	(2,3.0)	(4,6)	(2,3)	**0.009**	**0.147**	**0.840**	**11.010**

Table: Overstretched production – large customer arrival rate

A lost sales model

POLICIES			
INDEX		STATIC	
MED	MAX	MED	MAX
0.118	0.218	22.094	37.981
0.224	0.437	23.785	35.024
0.366	0.846	22.552	37.493
0.014	0.031	8.244	12.976
0.024	0.066	8.092	12.701
0.036	0.086	7.195	11.675
0.046	0.106	13.937	21.039
0.098	0.214	11.409	18.744
0.057	0.128	14.894	23.272

Table: Moderate demands – medium to large customer arrival rate

Conclusion and extensions

Conclusions

- Have found conditions for indexability in the single-product problem
- Understanding of where indexability fails to hold, practical ways to cope with non-indexability
- Complexity reduction from index policies for large K are enormous

Further ideas

- Necessary conditions for indexability? Too hard?
- Other natural heuristics?
- Could we use dynamic indices for non-fixed customer arrival rates?