The Joint Replenishment Problem
Optimal Policy And Exact Evaluation Method

Stefan Creemers
Robert Boute
(2022 ISIR)

IÉSEG SCHOOL OF MANAGEMENT
VLERICK BUSINESS SCHOOL
KATHOLIEKE UNIVERSITEIT LEUVEN

Introduction

Introduction

- The Joint Replenishment Problem (JRP) is a well-known problem in the ORMS literature

Introduction

- The Joint Replenishment Problem (JRP) is a well-known problem in the ORMS literature
- 2,080 hits on Google Scholar

Introduction

- The Joint Replenishment Problem (JRP) is a well-known problem in the ORMS literature
- 2,080 hits on Google Scholar
- General idea:
  - You keep several Stock Keeping Units (SKUs) in inventory.
  - For each SKU i, you incur a holding cost h_i and face a Poisson demand with rate parameter λ_i.
  - You can replenish the inventory of an SKU by issuing an order that has major order cost K. For each SKU i included in the order, you incur minor order cost k_i.

Introduction

- The Joint Replenishment Problem (JRP) is a well-known problem in the ORMS literature
- 2,080 hits on Google Scholar
- General idea:
  - You keep several Stock Keeping Units (SKUs) in inventory.
  - For each SKU i, you incur a holding cost h_i and face a Poisson demand with rate parameter λ_i.
  - You can replenish the inventory of an SKU by issuing an order that has major order cost K. For each SKU i included in the order, you incur minor order cost k_i.
- Million-dollar question: how do we coordinate orders such that holding and order costs are minimized?

Introduction

Introduction

- Ignall, E. 1969. *Optimal continuous review policies for two product inventory systems with joint setup costs*. Management Sci.,15(5), 278–283.

Introduction

- Ignall, E. 1969. *Optimal continuous review policies for two product inventory systems with joint setup costs*. Management Sci.,15(5), 278–283.
- Finding the optimal control policy is intractable, even for problems with only two SKUs

Introduction

- Ignall, E. 1969. *Optimal continuous review policies for two product inventory systems with joint setup costs*. Management Sci.,15(5), 278–283.
- Finding the optimal control policy is intractable, even for problems with only two SKUs
- Examples of tractable policies:
  - Periodic policy; see e.g., Atkins & Iyogun (1987) and Viswanathan (1997 & 2007)
  - Can-order policy

Introduction

- Ignall, E. 1969. *Optimal continuous review policies for two product inventory systems with joint setup costs*. Management Sci.,15(5), 278–283.
- Finding the optimal control policy is intractable, even for problems with only two SKUs
- Examples of tractable policies:
  - Periodic policy; see e.g., Atkins & Iyogun (1987) and Viswanathan (1997 & 2007)
  - Can-order policy

Can-order policy

Can-order policy

- Introduced by Balintfly (1964)

Can-order policy

- Introduced by Balintfly (1964)
- For each SKU i there are three parameters:
  - Order-up-to level S_i
  - Can-order level c_i
  - Reorder point s_i

Can-order policy

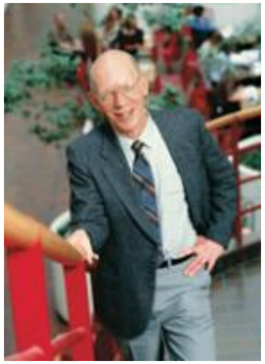
- Introduced by Balintfly (1964)
- For each SKU i there are three parameters:
  - Order-up-to level S_i
  - Can-order level c_i
  - Reorder point s_i
- If the inventory of one of the SKUs hits the reorder point, a replenishment order is triggered, and any other SKU that has inventory below/at the can-order level joins the order

Can-order policy

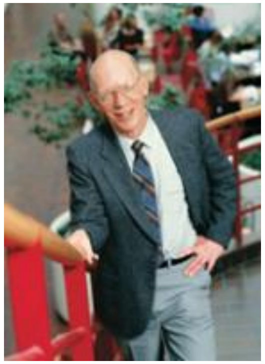
- Introduced by Balintfly (1964)
- For each SKU i there are three parameters:
  - Order-up-to level S_i
  - Can-order level c_i
  - Reorder point s_i
- If the inventory of one of the SKUs hits the reorder point, a replenishment order is triggered, and any other SKU that has inventory below/at the can-order level joins the order
- The exact cost of a can-order policy can be determined using a Continuous-Time Markov Chain (CTMC)

Can-order policy

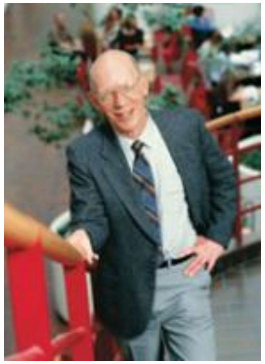
- Introduced by Balintfly (1964)
- For each SKU i there are three parameters:
  - Order-up-to level S_i
  - Can-order level c_i
  - Reorder point s_i
- If the inventory of one of the SKUs hits the reorder point, a replenishment order is triggered, and any other SKU that has inventory below/at the can-order level joins the order
- The exact cost of a can-order policy can be determined using a Continuous-Time Markov Chain (CTMC)
- However, for systems with more than a few SKUs, the CTMC becomes too big, and we can no longer determine the best can-order policy (curse of dimensionality!)



Decomposition approach

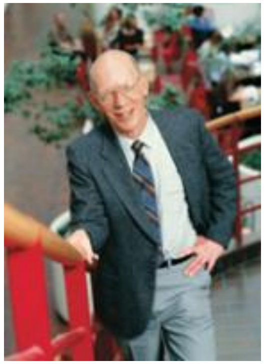


Decomposition approach



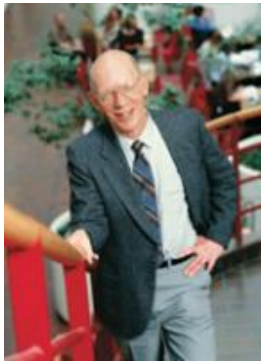
- Introduced by Silver E.A. (1974), and further refined by Federgruen et al. (1984)

Decomposition approach



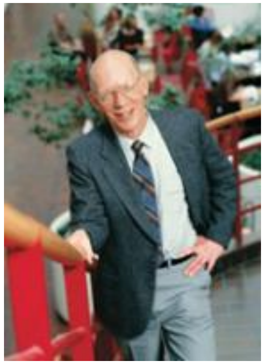
- Introduced by Silver E.A. (1974), and further refined by Federgruen et al. (1984)
- Can be used to obtain a "good" can-order policy, even for large problems with many SKUs

Decomposition approach



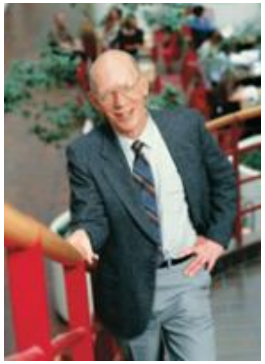
- Introduced by Silver E.A. (1974), and further refined by Federgruen et al. (1984)
- Can be used to obtain a "good" can-order policy, even for large problems with many SKUs
- The decomposition approach decomposes the JRP into single-item problems that are solved iteratively:

Decomposition approach



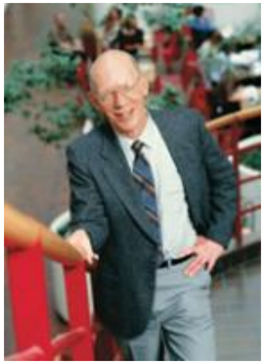
- Introduced by Silver E.A. (1974), and further refined by Federgruen et al. (1984)
- Can be used to obtain a “good” can-order policy, even for large problems with many SKUs
- The decomposition approach decomposes the JRP into single-item problems that are solved iteratively:
  - For each SKU i, find the best can-order parameters (S_i, c_i, and s_i) in a single-item system where the replenishment orders of other SKUs are captured using so-called “special replenishment opportunities” that arrive with rate μ_i.
  - Given the updated can-order policy for SKU i, determine the new rate of special replenishment opportunities μ_j for all other SKUs $j \neq i$.
  - Repeat this procedure for each SKU until the can-order policy itself convergences.

Decomposition approach



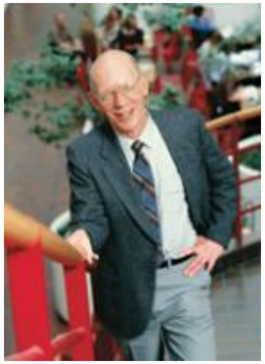
- Introduced by Silver E.A. (1974), and further refined by Federgruen et al. (1984)
- Can be used to obtain a “good” can-order policy, even for large problems with many SKUs
- The decomposition approach decomposes the JRP into single-item problems that are solved iteratively:
  - For each SKU i, find the best can-order parameters (S_i, c_i, and s_i) in a single-item system where the replenishment orders of other SKUs are captured using so-called “special replenishment opportunities” that arrive with rate μ_i.
  - Given the updated can-order policy for SKU i, determine the new rate of special replenishment opportunities μ_j for all other SKUs $j \neq i$.
  - Repeat this procedure for each SKU until the can-order policy itself convergences.
- Although the decomposition approach resolves the curse of dimensionality, there are some drawbacks:

Decomposition approach



- Introduced by Silver E.A. (1974), and further refined by Federgruen et al. (1984)
- Can be used to obtain a “good” can-order policy, even for large problems with many SKUs
- The decomposition approach decomposes the JRP into single-item problems that are solved iteratively:
  - For each SKU i, find the best can-order parameters (S_i, c_i, and s_i) in a single-item system where the replenishment orders of other SKUs are captured using so-called “special replenishment opportunities” that arrive with rate μ_i.
  - Given the updated can-order policy for SKU i, determine the new rate of special replenishment opportunities μ_j for all other SKUs $j \neq i$.
  - Repeat this procedure for each SKU until the can-order policy itself convergences.
- Although the decomposition approach resolves the curse of dimensionality, there are some drawbacks:
  - It is a heuristic procedure (the single-item problem for SKU i ignores the interaction of SKUs $j \neq i$; all interaction is captured by a single parameter μ_i).

Decomposition approach



- Introduced by Silver E.A. (1974), and further refined by Federgruen et al. (1984)
- Can be used to obtain a "good" can-order policy, even for large problems with many SKUs
- The decomposition approach decomposes the JRP into single-item problems that are solved iteratively:
  - For each SKU i, find the best can-order parameters (S_i, c_i, and s_i) in a single-item system where the replenishment orders of other SKUs are captured using so-called "special replenishment opportunities" that arrive with rate μ_i.
  - Given the updated can-order policy for SKU i, determine the new rate of special replenishment opportunities μ_j for all other SKUs $j \neq i$.
  - Repeat this procedure for each SKU until the can-order policy itself convergences.
- Although the decomposition approach resolves the curse of dimensionality, there are some drawbacks:
  - It is a heuristic procedure (the single-item problem for SKU i ignores the interaction of SKUs $j \neq i$; all interaction is captured by a single parameter μ_i).
  - It approximates the cost of a single-item system using a closed-form expression. As a result, we need to simulate the can-order policy in order to obtain its real cost. In addition, to determine whether one can-order policy is better than another, we base ourselves on approximate costs (that may differ substantially from the real cost).

Main contributions

Main contributions

- New, exact method to determine the cost of a JRP that partially solves the curse of dimensionality

New, exact method

New, exact method

- In a traditional CTMC approach, a state is defined as a tuple $(I_1, \ldots, I_N)$ (with I_i the inventory of SKU i, and N the number of SKUs). For a given can-order policy, the number of states is given by $\prod_{i=1}^{N}(S_i - s_i)$. Even for problems with only a few SKUs, the CTMC can no longer be analyzed.

New, exact method

- In a traditional CTMC approach, a state is defined as a tuple $(I_1, \dots, I_N)$ (with I_i the inventory of SKU i, and N the number of SKUs). For a given can-order policy, the number of states is given by $\prod_{i=1}^{N}(S_i - s_i)$. Even for problems with only a few SKUs, the CTMC can no longer be analyzed.
- We propose a new approach that uses a Discrete-Time Markov Chain (DTMC) that models transitions between so-called "initial states"; states in which we end up after an order has been triggered. By considering only initial states, we can reduce the number of states in our DTMC to $\sum_{i=1}^{N}\prod_{j\neq i}(S_i - c_i)$.

New, exact method

- In a traditional CTMC approach, a state is defined as a tuple $(I_1, \dots, I_N)$ (with I_i the inventory of SKU i, and N the number of SKUs). For a given can-order policy, the number of states is given by $\prod_{i=1}^{N}(S_i - s_i)$. Even for problems with only a few SKUs, the CTMC can no longer be analyzed.
- We propose a new approach that uses a Discrete-Time Markov Chain (DTMC) that models transitions between so-called "initial states"; states in which we end up after an order has been triggered. By considering only initial states, we can reduce the number of states in our DTMC to $\sum_{i=1}^{N}\prod_{j\neq i}(S_i - c_i)$.
- The reduction in the number of states can be significant:

Number of states required for analyzing the best can-order policy for the Federgruen instances			
Example problem	1	2	3
Traditional CTMC	34,848	34,848	18,000
New DTMC	256	300	853

New, exact method

- In a traditional CTMC approach, a state is defined as a tuple $(I_1, \dots, I_N)$ (with I_i the inventory of SKU i, and N the number of SKUs). For a given can-order policy, the number of states is given by $\prod_{i=1}^{N}(S_i - s_i)$. Even for problems with only a few SKUs, the CTMC can no longer be analyzed.
- We propose a new approach that uses a Discrete-Time Markov Chain (DTMC) that models transitions between so-called "initial states"; states in which we end up after an order has been triggered. By considering only initial states, we can reduce the number of states in our DTMC to $\sum_{i=1}^{N}\prod_{j\neq i}(S_i - c_i)$.
- The reduction in the number of states can be significant:

Number of states required for analyzing the best can-order policy for the Federgruen instances			
Example problem	1	2	3
Traditional CTMC	34,848	34,848	18,000
New DTMC	256	300	853

- This huge reduction in the number of states allows us to analyze JRP policies of larger problems with several SKUs.

New, exact method

- In a traditional CTMC approach, a state is defined as a tuple $(I_1, \dots, I_N)$ (with I_i the inventory of SKU i, and N the number of SKUs). For a given can-order policy, the number of states is given by $\prod_{i=1}^{N}(S_i - s_i)$. Even for problems with only a few SKUs, the CTMC can no longer be analyzed.
- We propose a new approach that uses a Discrete-Time Markov Chain (DTMC) that models transitions between so-called "initial states"; states in which we end up after an order has been triggered. By considering only initial states, we can reduce the number of states in our DTMC to $\sum_{i=1}^{N}\prod_{j\neq i}(S_i - c_i)$.
- The reduction in the number of states can be significant:

Number of states required for analyzing the best can-order policy for the Federgruen instances			
Example problem	1	2	3
Traditional CTMC	34,848	34,848	18,000
New DTMC	256	300	853

- This huge reduction in the number of states allows us to analyze JRP policies of larger problems with several SKUs.
- In addition, we can easily extend our method (compound Poisson demand, lead time, backlog, lost sales…) without increasing the number of states.

Main contributions

- New, exact method to determine the cost of a JRP that partially solves the curse of dimensionality

Main contributions

- New, exact method to determine the cost of a JRP that partially solves the curse of dimensionality
- Characterization of the optimal JRP policy

Optimal JRP policy

Optimal JRP policy

- The optimal policy has a can-order structure; if a set of SKUs joins the order triggered by SKU i, they will do so if their inventory level is at/below a given can-order level. The can-order level (and the order-up-to level) of the SKUs that join the order depends on the inventory levels of the SKUs that do not join the order.

Optimal JRP policy

- The optimal policy has a can-order structure; if a set of SKUs joins the order triggered by SKU i, they will do so if their inventory level is at/below a given can-order level. The can-order level (and the order-up-to level) of the SKUs that join the order depends on the inventory levels of the SKUs that do not join the order.
- Example

Example

- Problem with two SKUs:
  - $\lambda_1 = 12$ and $\lambda_2 = 16$
  - $h_1 = 12$ and $h_2 = 23$
  - $k_1 = 7$ and $k_2 = 21$
  - $K = 25$
- No lead time
- Best can-order policy:
  - $S_1 = 7$, $c_1 = 4$, and $s_1 = 0$
  - $S_2 = 8$, $c_2 = 2$, and $s_2 = 0$

Example

- Problem with two SKUs:
  - $\lambda_1 = 12$ and $\lambda_2 = 16$
  - $h_1 = 12$ and $h_2 = 23$
  - $k_1 = 7$ and $k_2 = 21$
  - $K = 25$
- No lead time
- Best can-order policy:
  - $S_1 = 7$, $c_1 = 4$, and $s_1 = 0$
  - $S_2 = 8$, $c_2 = 2$, and $s_2 = 0$

Can-order policy

(I_1, I_2)	→	(I_1, I_2)
(0,8)	→	(7,8)
(0,7)	→	(7,7)
(0,6)	→	(7,6)
(0,5)	→	(7,5)
(0,4)	→	(7,4)
(0,3)	→	(7,3)
(0,2)	→	(7,8)
(0,1)	→	(7,8)
(7,0)	→	(7,8)
(6,0)	→	(6,8)
(5,0)	→	(5,8)
(4,0)	→	(7,8)
(3,0)	→	(7,8)
(2,0)	→	(7,8)
(1,0)	→	(7,8)

Example

- Problem with two SKUs:
  - $\lambda_1 = 12$ and $\lambda_2 = 16$
  - $h_1 = 12$ and $h_2 = 23$
  - $k_1 = 7$ and $k_2 = 21$
  - $K = 25$
- No lead time
- Best can-order policy:
  - $S_1 = 7$, $c_1 = 4$, and $s_1 = 0$
  - $S_2 = 8$, $c_2 = 2$, and $s_2 = 0$

Can-order policy		Optimal policy	
(I_1, I_2) →	(I_1, I_2)	(I_1, I_2) →	(I_1, I_2)
(0,8) →	(7,8)	(0,8) →	(7,8)
(0,7) →	(7,7)	(0,7) →	(7,7)
(0,6) →	(7,6)	(0,6) →	(7,6)
(0,5) →	(7,5)	(0,5) →	(7,5)
(0,4) →	(7,4)	(0,4) →	(6,4)
(0,3) →	(7,3)	(0,3) →	(7,3)
(0,2) →	(7,8)	(0,2) →	(7,8)
(0,1) →	(7,8)	(0,1) →	(7,8)
(7,0) →	(7,8)	(7,0) →	(7,8)
(6,0) →	(6,8)	(6,0) →	(6,7)
(5,0) →	(5,8)	(5,0) →	(5,7)
(4,0) →	(7,8)	(4,0) →	(4,7)
(3,0) →	(7,8)	(3,0) →	(7,8)
(2,0) →	(7,8)	(2,0) →	(7,8)
(1,0) →	(7,8)	(1,0) →	(7,8)

Optimal JRP policy

- The optimal policy has a can-order structure; if a set of SKUs joins the order triggered by SKU i, they will do so if their inventory level is at/below a given can-order level. The can-order level (and the order-up-to level) of the SKUs that join the order depends on the inventory levels of the SKUs that do not join the order.
- Example
- Two important implications:

Optimal JRP policy

- The optimal policy has a can-order structure; if a set of SKUs joins the order triggered by SKU i, they will do so if their inventory level is at/below a given can-order level. The can-order level (and the order-up-to level) of the SKUs that join the order depends on the inventory levels of the SKUs that do not join the order.
- Example
- Two important implications:
  - The can-order policy is a logical heuristic; it adopts the structure of the optimal policy.

Optimal JRP policy

- The optimal policy has a can-order structure; if a set of SKUs joins the order triggered by SKU i, they will do so if their inventory level is at/below a given can-order level. The can-order level (and the order-up-to level) of the SKUs that join the order depends on the inventory levels of the SKUs that do not join the order.
- Example
- Two important implications:
  - The can-order policy is a logical heuristic; it adopts the structure of the optimal policy.
  - However, the can-order policy assumes a single can-order level for each SKU independent of the inventory levels of the SKUs that do not join the order ➔ if the number of SKUs increases, the optimality gap is expected to increase as well!

Main contributions

- New, exact method to determine the cost of a JRP that partially solves the curse of dimensionality
- Characterization of the optimal JRP policy

Main contributions

- New, exact method to determine the cost of a JRP that partially solves the curse of dimensionality
- Characterization of the optimal JRP policy
- Introduction of a new, generalized can-order policy

Generalized can-order policy

Generalized can-order policy

- Using the insights of the optimal JRP policy, the can-order policy can be generalized using a greedy procedure:
  - Start from the best can-order policy.
  - For each combination of inventory levels of SKUs that do not join the order, evaluate whether it is beneficial to alter the can-order level (and/or order-up-to level) of SKUs that do join the order.
  - Repeat until no further improvement can be found.

Generalized can-order policy

- Using the insights of the optimal JRP policy, the can-order policy can be generalized using a greedy procedure:
  - Start from the best can-order policy.
  - For each combination of inventory levels of SKUs that do not join the order, evaluate whether it is beneficial to alter the can-order level (and/or order-up-to level) of SKUs that do join the order.
  - Repeat until no further improvement can be found.
- After applying this to the Federgruen instances, we get:

Expected cost of can-order policy and generalized can-order policy			
Example problem	1	2	3
Best can-order policy	77.51	80.87	67.80
Generalized can-order policy	77.36	80.73	67.45

Generalized can-order policy

- Using the insights of the optimal JRP policy, the can-order policy can be generalized using a greedy procedure:
  - Start from the best can-order policy.
  - For each combination of inventory levels of SKUs that do not join the order, evaluate whether it is beneficial to alter the can-order level (and/or order-up-to level) of SKUs that do join the order.
  - Repeat until no further improvement can be found.
- After applying this to the Federgruen instances, we get:

Expected cost of can-order policy and generalized can-order policy			
Example problem	1	2	3
Best can-order policy	77.51	80.87	67.80
Generalized can-order policy	77.36	80.73	67.45

- The difference is not substantial, however, we expect the gap to increase if the number of SKUs increases!

Generalized can-order policy

- Using the insights of the optimal JRP policy, the can-order policy can be generalized using a greedy procedure:
  - Start from the best can-order policy.
  - For each combination of inventory levels of SKUs that do not join the order, evaluate whether it is beneficial to alter the can-order level (and/or order-up-to level) of SKUs that do join the order.
  - Repeat until no further improvement can be found.
- After applying this to the Federgruen instances, we get:

Expected cost of can-order policy and generalized can-order policy			
Example problem	1	2	3
Best can-order policy	77.51	80.87	67.80
Generalized can-order policy	77.36	80.73	67.45

- The difference is not substantial, however, we expect the gap to increase if the number of SKUs increases!

Main contributions

- New, exact method to determine the cost of a JRP that partially solves the curse of dimensionality
- Characterization of the optimal JRP policy
- Introduction of a new, generalized can-order policy

Main contributions

- New, exact method to determine the cost of a JRP that partially solves the curse of dimensionality
- Characterization of the optimal JRP policy
- Introduction of a new, generalized can-order policy
- Generalization of the decomposition approach

Generalized decomposition approach

Generalized decomposition approach

- Our exact method can analyze problems with several SKUs. Therefore, we can generalize the decomposition approach, and now also decompose the JRP into double-item and triple-item problems (next to single-item problems).

Generalized decomposition approach

- Our exact method can analyze problems with several SKUs. Therefore, we can generalize the decomposition approach, and now also decompose the JRP into double-item and triple-item problems (next to single-item problems).
- In addition, rather than using an approximate (closed-form) cost function, we use our method to analyze the exact cost of the single/double/triple-item problems.

Expected cost of different policies for the Federgruen instances			
Example problem	1	2	3
Decomposition approach (approximation)	88.71	89.98	71.53
Decomposition approach (exact)	81.03	83.62	68.52
Generalized decomposition (single item)	80.07	82.66	68.70
Generalized decomposition (double item)	78.10	82.16	68.04
Generalized decomposition (triple item)	77.97	81.27	67.96
Best can-order policy	77.51	80.87	67.80

Main contributions

- New, exact method to determine the cost of a JRP that partially solves the curse of dimensionality
- Characterization of the optimal JRP policy
- Introduction of a new, generalized can-order policy
- Generalization of the decomposition approach

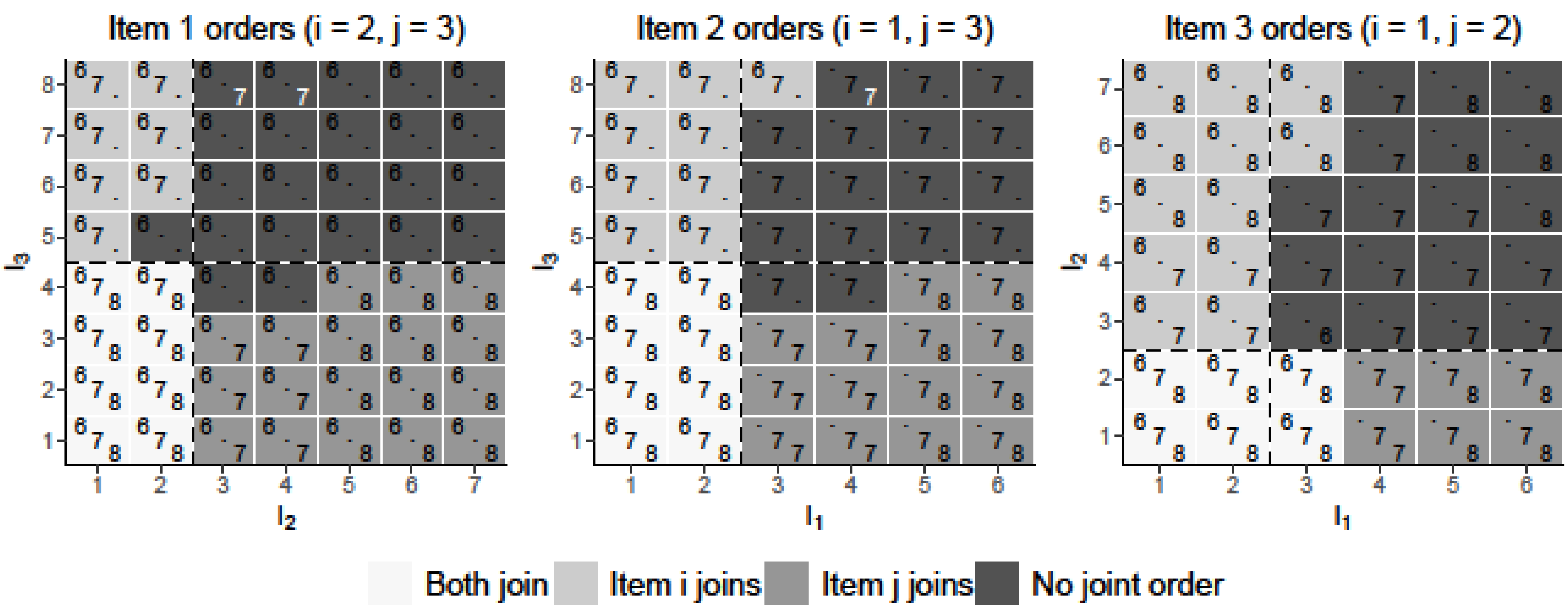
?

Generalized can-order policy (3 items)

- Best can-order policy:
  - $S_1 = 6$, $c_1 = 2$, and $s_1 = 0$
  - $S_2 = 7$, $c_2 = 2$, and $s_2 = 0$
  - $S_3 = 7$, $c_3 = 4$, and $s_3 = 0$

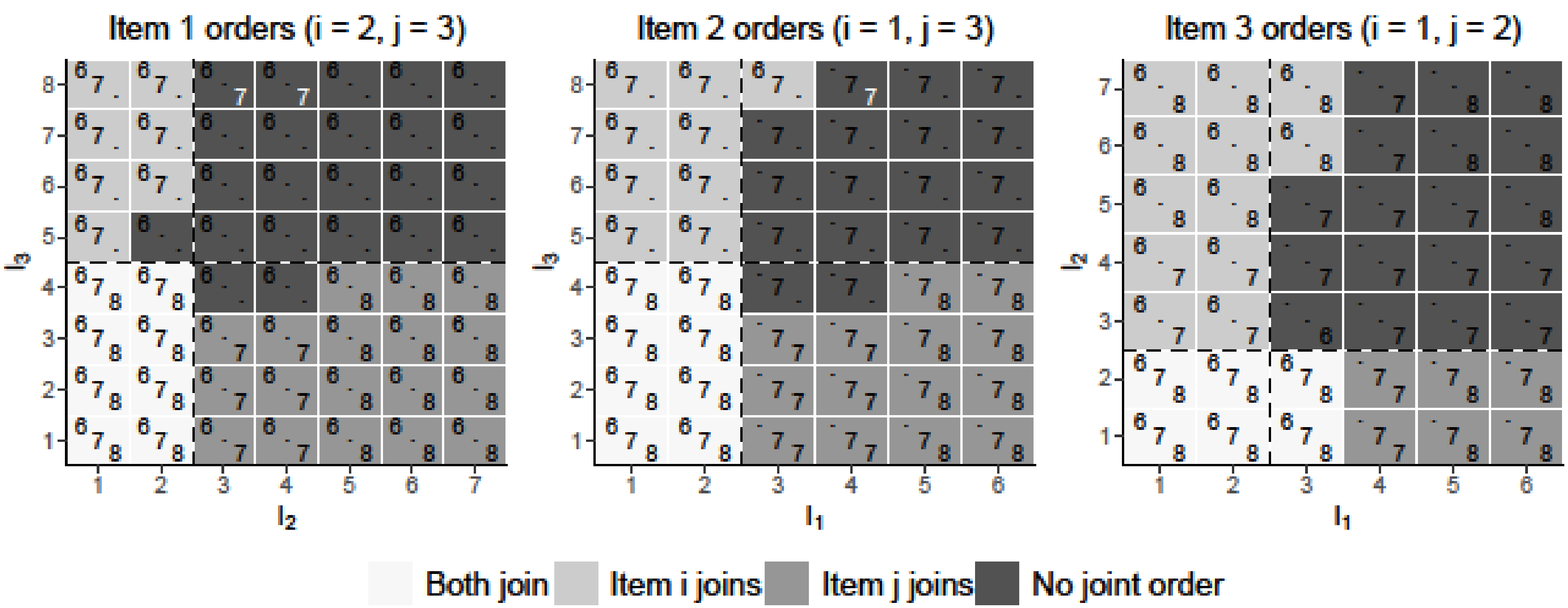
Generalized can-order policy (3 items)

- Best can-order policy:
  - $S_1 = 6$, $c_1 = 2$, and $s_1 = 0$
  - $S_2 = 7$, $c_2 = 2$, and $s_2 = 0$
  - $S_3 = 7$, $c_3 = 4$, and $s_3 = 0$



Generalized can-order policy (3 items)

- Best can-order policy:
  - $S_1 = 6$, $c_1 = 2$, and $s_1 = 0$
  - $S_2 = 7$, $c_2 = 2$, and $s_2 = 0$
  - $S_3 = 7$, $c_3 = 4$, and $s_3 = 0$
- It may be optimal to return inventory!



?