Lecture VI - Minimizing Split Orders in E-Commerce

Applied Optimization with Julia

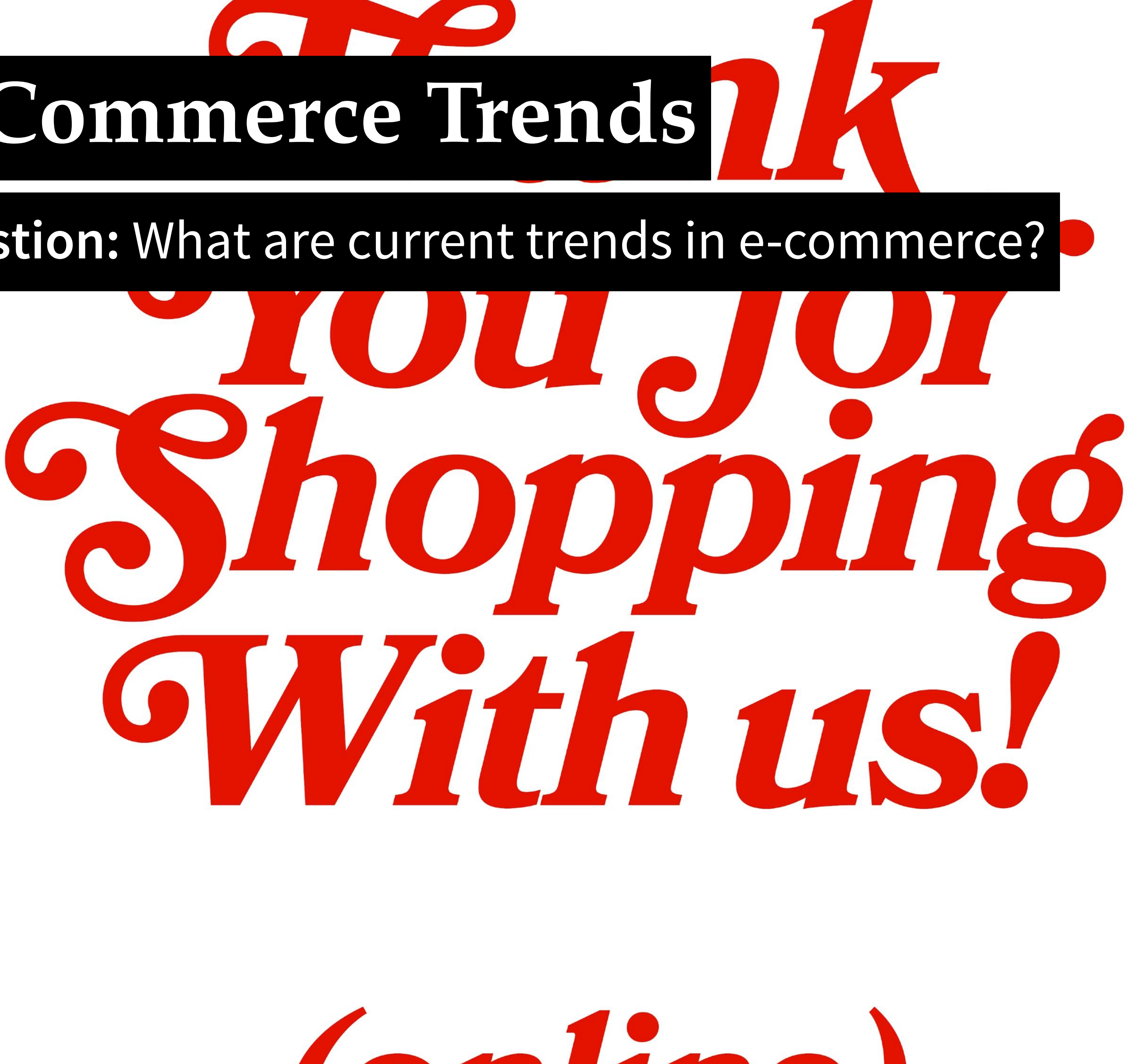
Dr. Tobias Vlćek

University of Hamburg - Fall 2024

Introduction

E-Commerce Trends

Question: What are current trends in e-commerce?



E-Commerce Sales

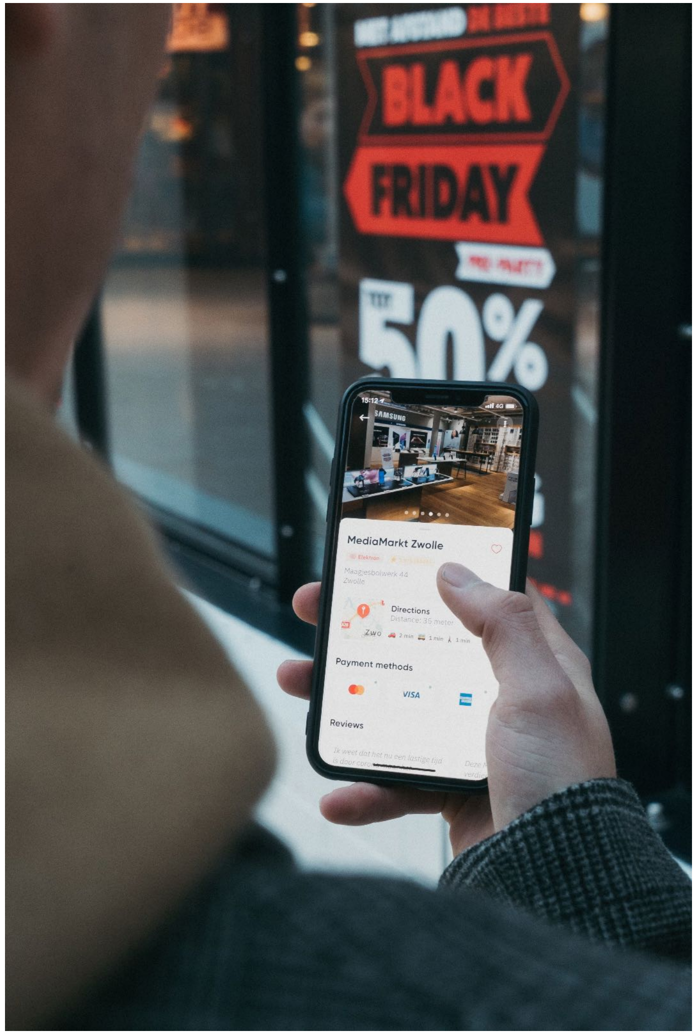
- E-Commerce sales **are growing fast**:
  - Products are **no longer bound between borders**
  - Product variety is **rising**
  - Consumer shopping patterns are **shifting**
  - Brick-and-mortar stores **loose customers to the internet**
  - Covid-19 **accelerated this trend even more**

Parcels Worldwide

- The number of parcels is rising:
  - **2014**: 44 billion parcels (Pitney Bowes Inc. 2017)
  - **2019**: 103 billion parcels (Pitney Bowes Inc. 2019)
  - **2026**: 220 – 262 billion parcels [1] (Pitney Bowes Inc. 2020)

1. Forecast, not actual number

Pressure on infrastructure



- Consumers nowadays expect **free and fast deliveries and returns**
- Existing warehouses have to store an **increasing range of products**
- Better customer service requires **faster deliveries**
- Incurred fulfillment costs **depend on the number of parcels**

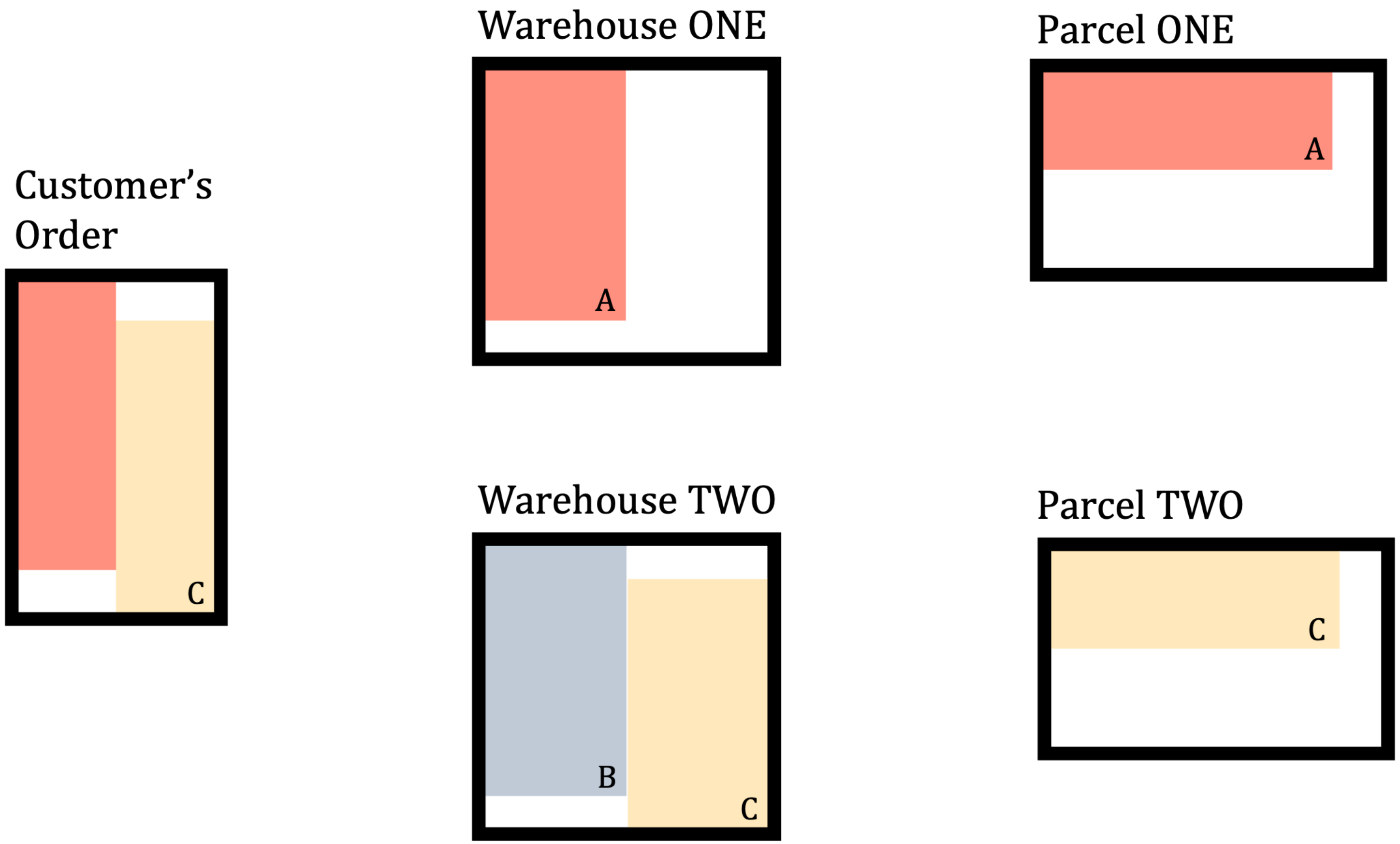
Pressure on the environment

- **Each parcel packaging** consumes resources during production
- Every dispatched parcel to the customer **causes CO_2 emissions**
- In case of returns, **more parcels cause more emissions**

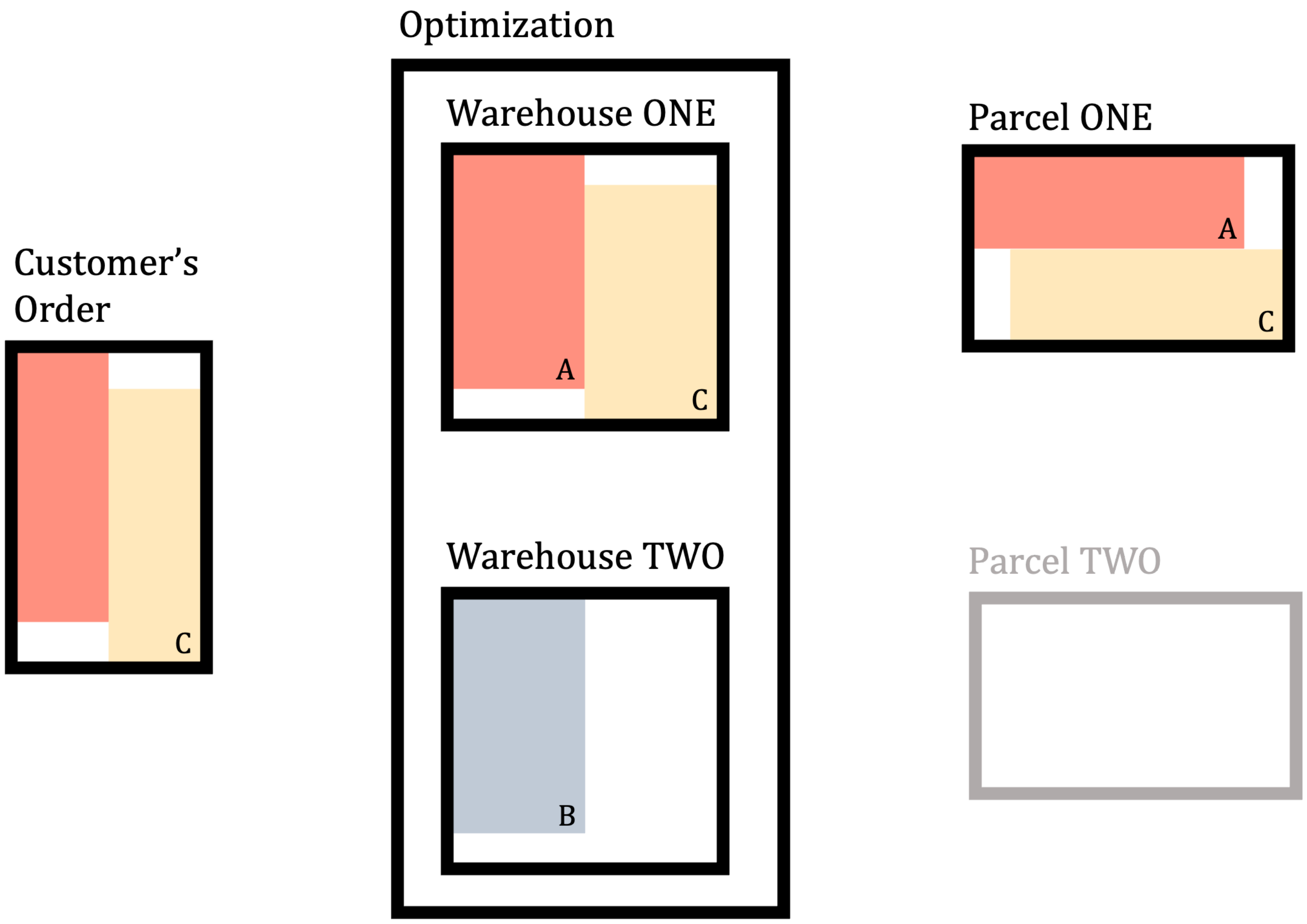
Problem Structure

Split Order

Question: What is a split order?



No Split Order



Reason for Split Orders

Question: Why might they occur?

- **Stock availability**: Some products are **out of stock** at a warehouse and need to be fulfilled from another warehouse
- **Capacity constraints**: Some products are stored at **different warehouses** and need to be shipped from elsewhere

Impact of Split Orders

Question: **What are the consequences?**

- **Higher** shipping costs
- **Increased** packaging material
- **More** CO_2 emissions
- **Higher operational complexity**
- **Lower** customer satisfaction

Mitigations?

Question: What are possible mitigations?

- **Consolidation**: Ship to a central warehouse before dispatch
- **Cross-docking**: Ship directly from supplier to customer
- **Transshipment**: Ship between warehouses before delivery
- **Co-allocation**: Predict **co-appearance of products** and allocate them to the **same warehouse**

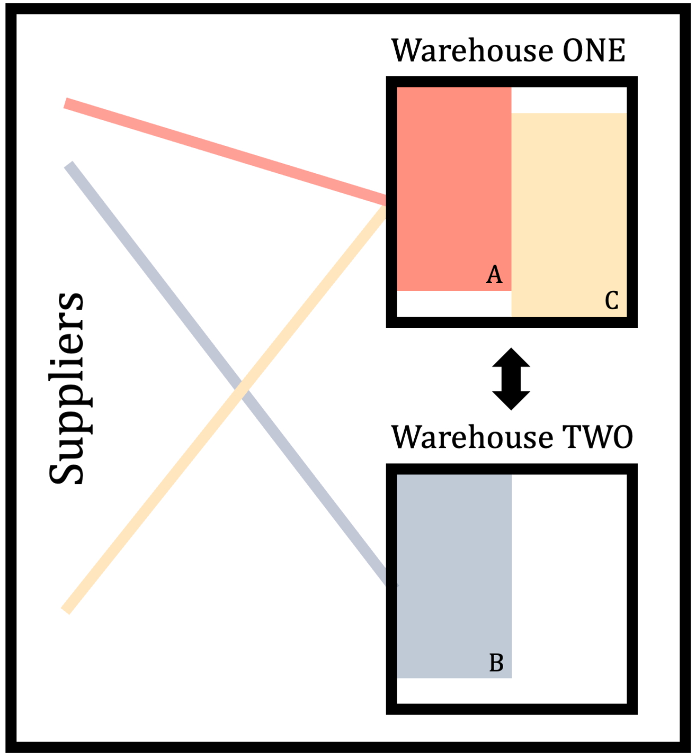
Case Study

Key information about the case:

- a large European e-commerce retailer
- the retailer has two warehouses
- product range cannot be stored in either warehouse
- product deliveries can be made to both warehouses
- products do not have to be stored exclusively

Problem Structure - Version 1

Optimizing Co-allocation



Question: **What could be our objective?**

We aim to improve the **SKU[1]-warehouse allocation** to minimize the number of split parcels resulting from **SKUs being stored in different warehouses.**

1. SKU: Stock Keeping Unit

Available Sets

Question: What could be the sets here?

- $\mathcal{I}$ - Set of products indexed by $i \in \{1, 2, \ldots, |\mathcal{I}|\}$
- $\mathcal{K}$ - Set of warehouses indexed by $k \in \{1, \ldots, |\mathcal{K}|\}$
- $\mathcal{M}$ - Set of customer orders $m \in \{1, 2, \ldots, |\mathcal{M}|\}$

Available Parameters

Question: What are possible parameters?

- c_k - Storage space of warehouse $k \in \{1, \dots, |\mathcal{K}|\}$
- $\boldsymbol{T} = (t_{m,i})$ - Past customer orders for SKUs

Question: What could the transactional data look like?

Transactional Data

Example of $\boldsymbol{T}$

$t_{m,i}$	A	B	C	D
1	1	1	1	0
2	1	1	1	0
3	1	1	0	0
4	1	0	0	1
5	1	0	0	1
6	1	0	0	1
7	1	0	0	1
8	0	0	1	1

Past vs. Future

- The **transactional data** T is based on **past orders**
- It is a **binary matrix** of customer orders and SKUs
- We use this data to **assume** future co-occurrence
  - **Past co-occurrence predicts future co-occurrence**

Question: What is your opinion on the assumption?

Split-Order Minimization

Question: What could be our decision variable/s?

> ⓘ We have the following sets:
>
> - $\mathcal{I}$ - Set of products indexed by $i \in \{1, 2, \ldots, |\mathcal{I}|\}$
> - $\mathcal{K}$ - Set of warehouses indexed by $k \in \{1, \ldots, |\mathcal{K}|\}$
> - $\mathcal{M}$ - Set of customer orders $m \in \{1, 2, \ldots, |\mathcal{M}|\}$

- $X_{i,k}$ - 1, if $i \in \mathcal{I}$ is stored in $k \in \mathcal{K}$, 0 otherwise
- $Y_{m,i,k}$ - 1, if SKU $i \in \mathcal{I}$ is shipped from warehouse $k \in \mathcal{K}$ for customer order $m \in \mathcal{M}$, 0 otherwise

Integer Programming Model

- Catalán and Fisher (2012) created an **integer model**
- Number of SKUs of E-Commerce retailers can easily be **between 10,000 - 100,000**
- Number of customer orders necessary for “stable” results have to be higher in the order of **100,000 - 10,000,000**

Question: Anybody an idea what this could mean?

Implementation Challenges

- Small instance with 10 SKUs and 1000 customer orders
- **CPLEX 20.1.0** needs 3100 seconds to solve the problem
- Computation times scales exponentially
- $\rightarrow$ **Not applicable** in real world applications!

Any idea what could be done?

Problem Structure - Version 2

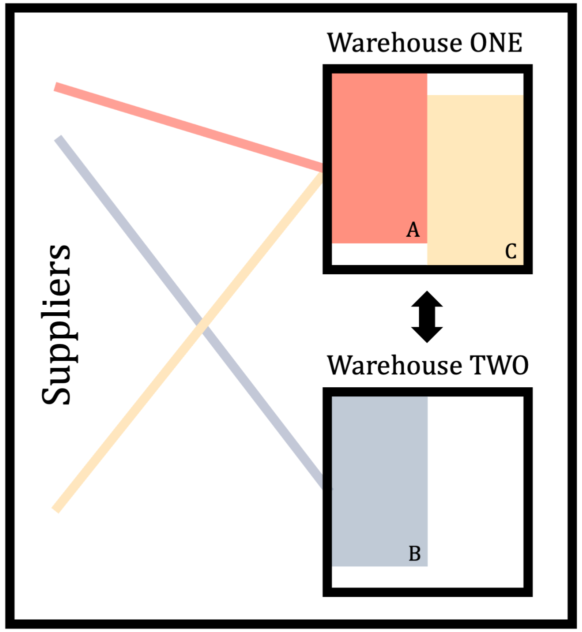
Heuristic Approach

- **Heuristic**: Fast, but not necessarily optimal
- **Approximation**: Not guaranteed to be optimal, but close
- **Computational Effort**: Reasonable even for large instances

Different view on the problem

Focus on the warehouses and the co-appearance of SKUs! Discard the exact information about the customer orders.

Objective



Question: What could be the objective?

Maximize the coappearance of products that are often **part of the same customer orders.**

Transaction Matrix

```julia
T = [
      1 1 1 0;
      1 1 1 0;
      1 1 0 0;
      1 0 0 1;
      1 0 0 1;
      1 0 0 1;
      1 0 0 1;
      0 0 1 1
]

# Create the coappearance matrix
Q = T' * T
println("Coappearance matrix Q:")
display(Q)
```

```
Coappearance matrix Q:

4×4 Matrix{Int64}:
 7  3  2  4
 3  3  2  0
 2  2  3  1
 4  0  1  5
```

Coappearance Matrix

- $\boldsymbol{Q}$ is a **symmetric binary matrix**
- Proposed by Catalán and Fisher (2012)
- $\boldsymbol{Q} = (\boldsymbol{T}^T \cdot \boldsymbol{T})$ where $\boldsymbol{Q} = (q_{ij})_{i \in \{1,...,\mathcal{I}\}, j \in \{1,...,\mathcal{I}\}}$
- q_{ij} shows how often i and j appear **in the same order**

Question: What do the principal diagonal values tell us?

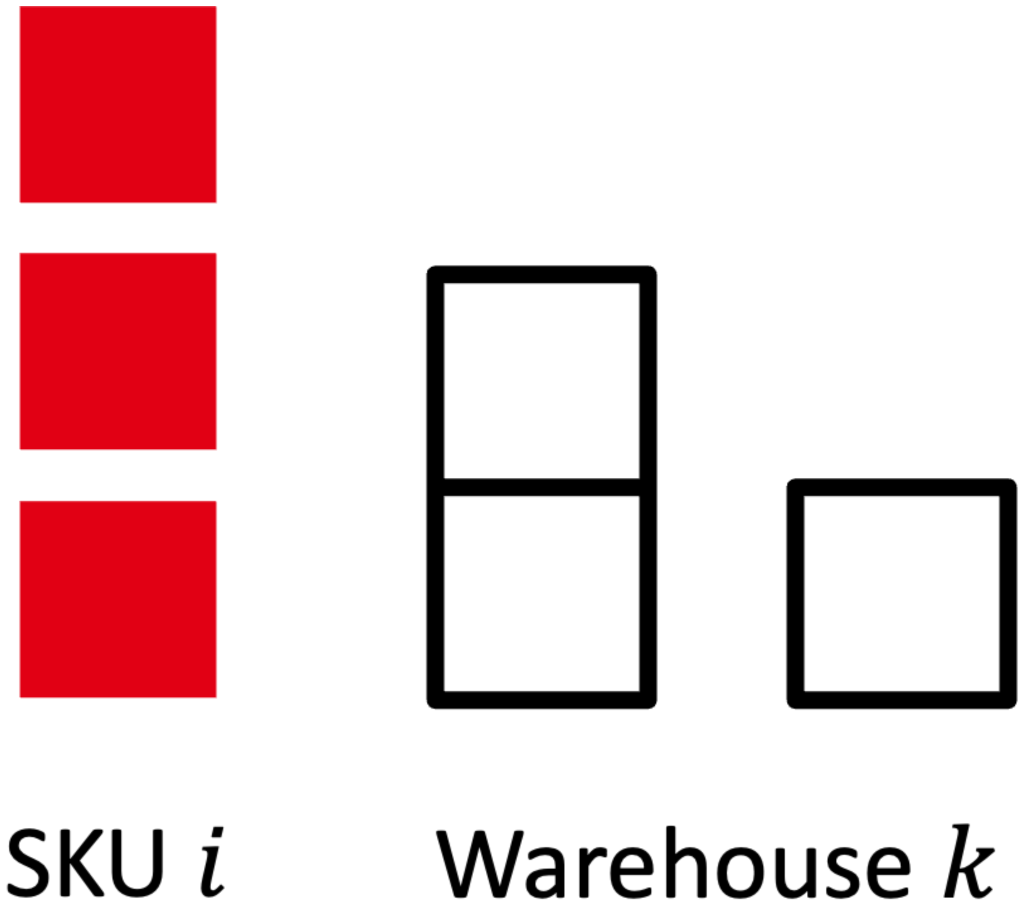
- How often each SKU appeared over all orders **(binary!)**

How to approach the problem?

- **Greedy Heuristic**[1]: Allocation based on matrix
- **Mathematical Model**[2]: Maximizes coappearance
- **GRASP**[3]: Good on small instances
- **New**: Max. coappearance with non-linear solver
- **New**: Heuristic based on Chi-Square Tests

1. Simple and very fast, Catalán and Fisher (2012)
2. Computationally intensive with CPLEX, Zhu et al. (2021)
3. Greedy Randomized Adaptive Search Procedure, Zhu et al. (2021)

Basic Setting



Available Data (Version 2)

Question: What could be the sets?

- $\mathcal{I}$ - Set of products indexed by $i \in \{1, 2, \ldots, |\mathcal{I}|\}$
- $\mathcal{K}$ - Set of warehouses indexed by $k \in \{1, \ldots, |\mathcal{K}|\}$

No customer order information is needed!

We can focus on the SKUs and the warehouses, making the problem **much smaller**!

Available Parameters

Question: What are possible parameters?

- c_k - Storage space of warehouse $k \in \{1, \ldots, |\mathcal{K}|\}$
- $\boldsymbol{Q} = (q_{ij})_{i \in \{1,\ldots,\mathcal{I}\}, j \in \{1,\ldots,\mathcal{I}\}}$ - Coappearance matrix

Transactional Data replaced

Instead of the transactional data, we just **use the coappearance matrix** in our model!

Model Formulation

Decision Variables?

We have the following sets:

- $\mathcal{I}$ - Set of products indexed by $i \in \{1, 2, \ldots, |\mathcal{I}|\}$
- $\mathcal{K}$ - Set of warehouses indexed by $k \in \{1, \ldots, |\mathcal{K}|\}$

Our objective is to:

Maximize the coappearance of products that are often part of the same customer orders. **In more mathematical terms:** Maximize the sum of all unique pair-wise values $q_{i,j}$ of all SKUs stored in the same warehouse.

Question: What could be our decision variable/s?

Decision Variables

- $X_{i,k}$ - 1, if SKU $i \in \mathcal{I}$ is stored in $k \in \mathcal{K}$, 0 otherwise

Only one variable per SKU and warehouse!

As we don't need the customer order information, we only need to make a decision for each SKU and warehouse pair!

Decision Variable in Julia

Question: How could we formulate the variable in Julia?

```julia
using JuMP, SCIP # SCIP is a non-commercial MIQCP solver

warehouses = ["Hamburg", "Berlin"] # Add warehouses as a vector
skus = ["Smartphone", "Socks", "Charger"] # Add SKUs as a vector

warehouse_model = Model(SCIP.Optimizer)
```

```julia
@variable(warehouse_model, X[i in skus, k in warehouses], Bin)
```

```
2-dimensional DenseAxisArray{JuMP.VariableRef,2,...} with index sets:
    Dimension 1, ["Smartphone", "Socks", "Charger"]
    Dimension 2, ["Hamburg", "Berlin"]
And data, a 3×2 Matrix{JuMP.VariableRef}:
 X[Smartphone,Hamburg]  X[Smartphone,Berlin]
 X[Socks,Hamburg]       X[Socks,Berlin]
 X[Charger,Hamburg]     X[Charger,Berlin]
```

Objective Function

We need the following:

- $X_{i,k}$ - 1, if SKU $i \in \mathcal{I}$ is stored in $k \in \mathcal{K}$, 0 otherwise
- q_{ij} - Coappearance of SKU $i \in \mathcal{I}$ and $j \in \mathcal{I}$

Our objective is to:

Maximize the sum of all unique pair-wise values $q_{i,j}$ of all SKUs stored in the same warehouse. Note, that this is a **quadratic objective function**!

Question: What could the objective function look like?

Quadratic Objective Function

$$\text{maximize} \quad \sum_{i=2}^{\mathcal{I}} \sum_{j=1}^{i-1} \sum_{k \in \mathcal{K}} X_{ik} \times X_{jk} \times q_{ij}$$

This is a quadratic objective function!

The quadratic terms are $X_{ik} \times X_{jk}$. This objective function is based on the **Quadratic Multiple Knapsack Problem (QMKP)**, formulated by Hiley and Julstrom (2006).

Objective Function in Julia

Question: How could we formulate this in Julia?

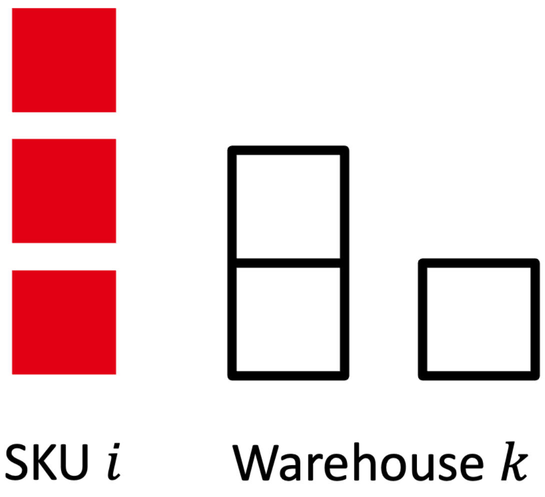
```julia
Q = [2 1 2; 1 2 1; 2 1 2]

@objective(warehouse_model,
    Max,
    sum(
        X[skus[i], warehouses[k]] * X[skus[j], warehouses[k]] * Q[i,j]
        for i in 2:length(skus)
        for j in 1:i-1
        for k in 1:length(warehouses)
    )
)
```

X[Socks,Hamburg]∗X[Smartphone,Hamburg] + X[Socks,Berlin]∗X[Smartphone,Berlin] + 2 X[Charger,Hamburg]∗X[Smartphone,Hamburg] + 2 X[Charger,Berlin]∗X[Smartphone,Berlin] + X[Charger,Hamburg]∗X[Socks,Hamburg] + X[Charger,Berlin]∗X[Socks,Berlin]

Constraints

What constraints?



Question: **What constraints?**

- Allocate each SKU **at least once**
- Warehouses have a **finite capacity**
- Capacity is **not exceeded**

Single Allocation Constraint?

The goal of this constraint is to:

Ensure that each SKU is allocated at least once.

We need the following variable:

- $X_{i,k}$ - 1, if SKU $i \in \mathcal{I}$ is stored in $k \in \mathcal{K}$, 0 otherwise

Question: What could the constraint look like?

Single Allocation Constraint

$$\sum_{k \in \mathcal{K}} X_{ik} \geq 1 \quad \forall i \in \mathcal{I}$$

Remember, this is the variable:

- $X_{i,k}$ - 1, if SKU $i \in \mathcal{I}$ is stored in $k \in \mathcal{K}$, 0 otherwise

Question: How could we change the constraint to ensure that each SKU is allocated only once?

Question: How could we add the constraint in Julia?

Single Allocation in Julia

```julia
@constraint(warehouse_model, single_allocation[i in skus],
    sum(X[i, k] for k in warehouses) >= 1
)
```

```
1-dimensional DenseAxisArray{JuMP.ConstraintRef{JuMP.Model,
MathOptInterface.ConstraintIndex{MathOptInterface.ScalarAffineFunction{Float64}
MathOptInterface.GreaterThan{Float64}}, JuMP.ScalarShape},1,...} with index
sets:
    Dimension 1, ["Smartphone", "Socks", "Charger"]
And data, a 3-element Vector{JuMP.ConstraintRef{JuMP.Model,
MathOptInterface.ConstraintIndex{MathOptInterface.ScalarAffineFunction{Float64}
MathOptInterface.GreaterThan{Float64}}, JuMP.ScalarShape}}:
 single_allocation[Smartphone] : X[Smartphone,Hamburg] + X[Smartphone,Berlin]
≥ 1
 single_allocation[Socks] : X[Socks,Hamburg] + X[Socks,Berlin] ≥ 1
 single_allocation[Charger] : X[Charger,Hamburg] + X[Charger,Berlin] ≥ 1
```

Capacity Constraints?

The goal of these constraints is to:

Ensure that the capacity of each warehouse is not exceeded.

We need the following variables and parameters:

- $X_{i,k}$ - 1, if SKU $i \in \mathcal{I}$ is stored in $k \in \mathcal{K}$, 0 otherwise
- c_k - Storage space of warehouse $k \in \mathcal{K}$

Question: What could the second constraint be?

Capacity Constraints

$$\sum_{i \in \mathcal{I}} X_{ik} \leq c_k \quad \forall k \in \mathcal{K}$$

And that's basically it!

Question: How could we add the second constraint in Julia?

Capacity Constraints in Julia

```
capacities = Dict("Hamburg" => 2, "Berlin" => 1) # Add capacities

@constraint(warehouse_model, capacity[k in warehouses],
    sum(X[i, k] for i in skus) <= capacities[k]
)
```

```
1-dimensional DenseAxisArray{JuMP.ConstraintRef{JuMP.Model,
MathOptInterface.ConstraintIndex{MathOptInterface.ScalarAffineFunction{Float64}
MathOptInterface.LessThan{Float64}}, JuMP.ScalarShape},1,...} with index sets:
    Dimension 1, ["Hamburg", "Berlin"]
And data, a 2-element Vector{JuMP.ConstraintRef{JuMP.Model,
MathOptInterface.ConstraintIndex{MathOptInterface.ScalarAffineFunction{Float64}
MathOptInterface.LessThan{Float64}}, JuMP.ScalarShape}}:
 capacity[Hamburg] : X[Smartphone,Hamburg] + X[Socks,Hamburg] +
X[Charger,Hamburg] ≤ 2
 capacity[Berlin] : X[Smartphone,Berlin] + X[Socks,Berlin] + X[Charger,Berlin]
≤ 1
```

QMK Model

$$\text{maximize} \quad \sum_{i=2}^{\mathcal{I}} \sum_{j=1}^{i-1} \sum_{k \in \mathcal{K}} X_{ik} \times X_{jk} \times q_{ij}$$

subject to:

$$\sum_{k \in \mathcal{K}} X_{ik} \geq 1 \qquad \forall i \in \mathcal{I}$$

$$\sum_{i \in \mathcal{I}} X_{ik} \leq c_k \qquad \forall k \in \mathcal{K}$$

$$X_{ik} \in \{0, 1\} \qquad \forall i \in \mathcal{I}, \forall k \in \mathcal{K}$$

QMK Model in Julia

```
set_attribute(warehouse_model, "display/verblevel", 0) # Hide solver output
optimize!(warehouse_model)

println("The optimal objective value is: ", objective_value(warehouse_model
println("The optimal solution is: ", value.(X))
```

```
The optimal objective value is: 2.0
The optimal solution is: 2-dimensional DenseAxisArray{Float64,2,...} with
index sets:
    Dimension 1, ["Smartphone", "Socks", "Charger"]
    Dimension 2, ["Hamburg", "Berlin"]
And data, a 3×2 Matrix{Float64}:
  1.0  0.0
 -0.0  1.0
  1.0  0.0
```

Model Characteristics

Characteristics

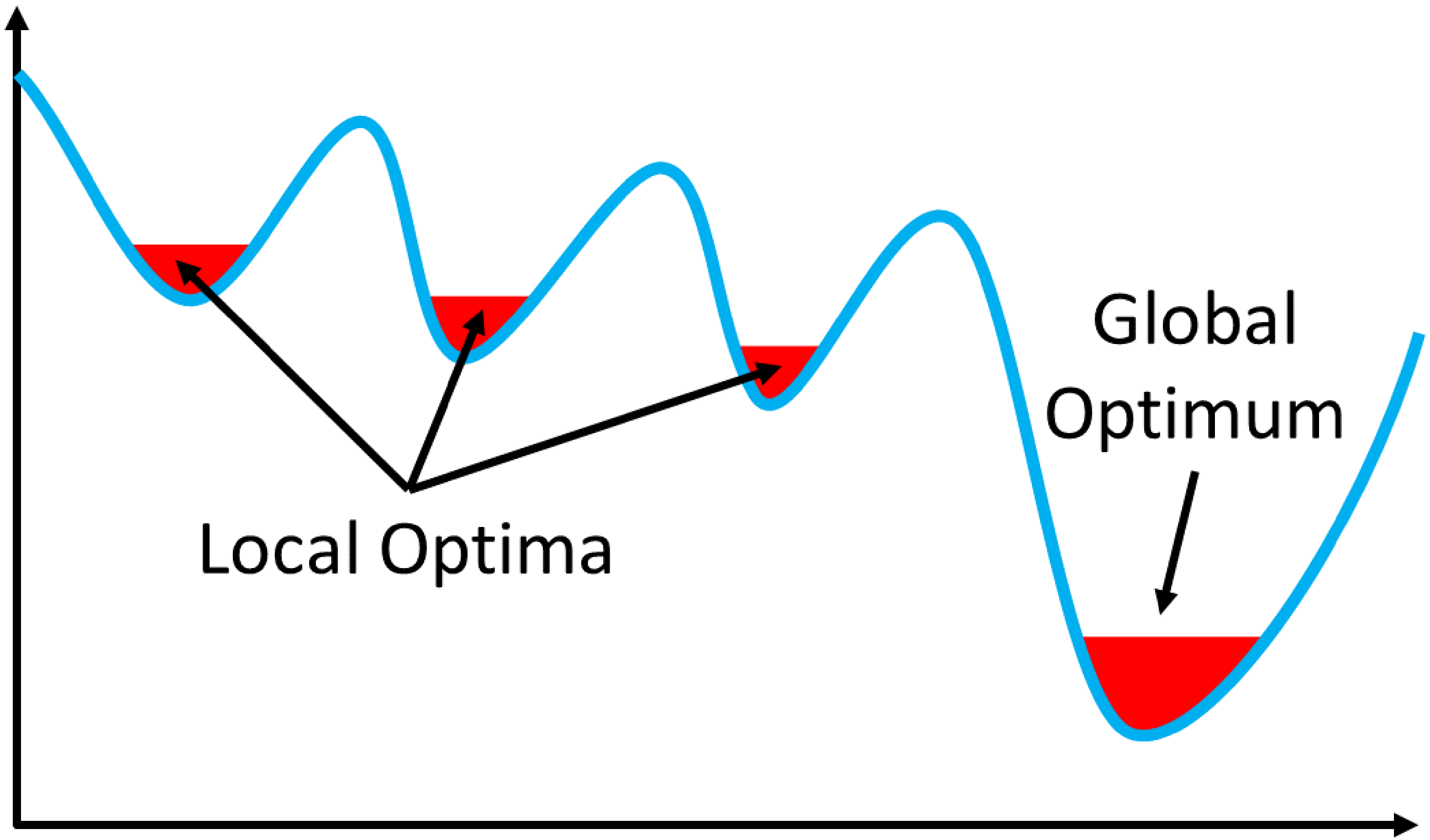
- Is the model formulation **linear/ non-linear?**
- What kind of **variable domain** do we have?
- Do we know the **split-orders** based on the **objective value?**
- Why **couldn’t we use HiGHS** as solver?

Choosing a solver

- Identify **problem structure**, e.g. LP, MIP, NLP, QCP, MIQCP, …
- What is the **size** of the problem?
- Is a **commercial** solver needed?

> **Commercial Solvers**
>
> Commercial solvers are **faster** and **more robust** as open source solvers but also **more expensive**. During your studies, you can use most of them for free though! Nonetheless, we will only use open source solvers in this course.

Global vs Local Optimality



Local vs Global Optimum by Christoph Roser

Model Assumptions

Questions: On model assumptions

- What assumptions have we made?
- Problem with allocating SKUs to multiple warehouses?
- What else might pose a problem in the real world?

Impact

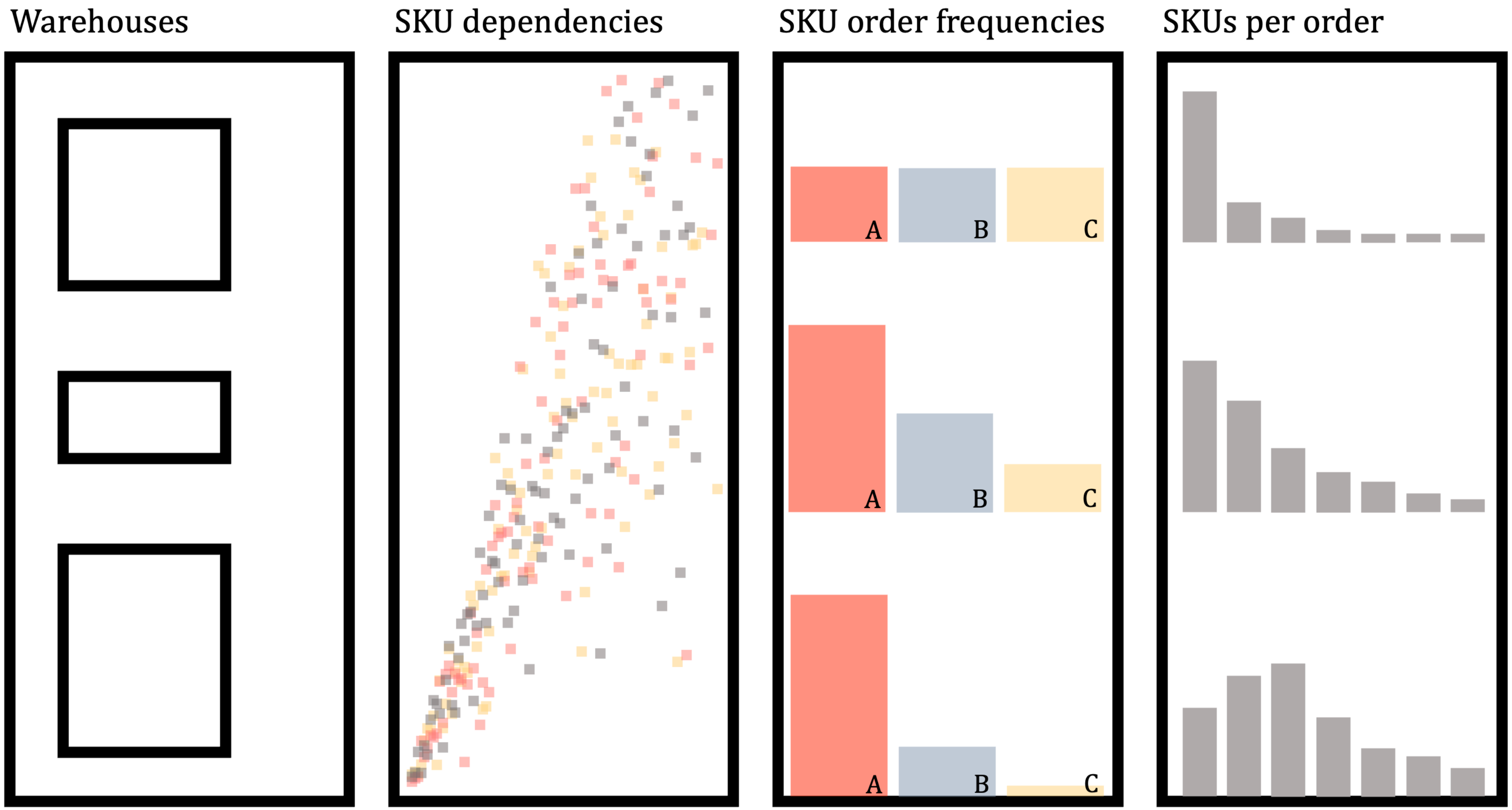
Can this be applied?

Problem Size is Crucial

- Up to 10,000 SKUs → **commercial solvers**
- More than 10,000 SKUs → **heuristics**
- For example, the **CHI** heuristic

> **CHI-Heuristic**
>
> Detect dependencies between products and allocate them accordingly, as products within orders can have dependencies and products are bought with different frequencies!

Potential Improvements

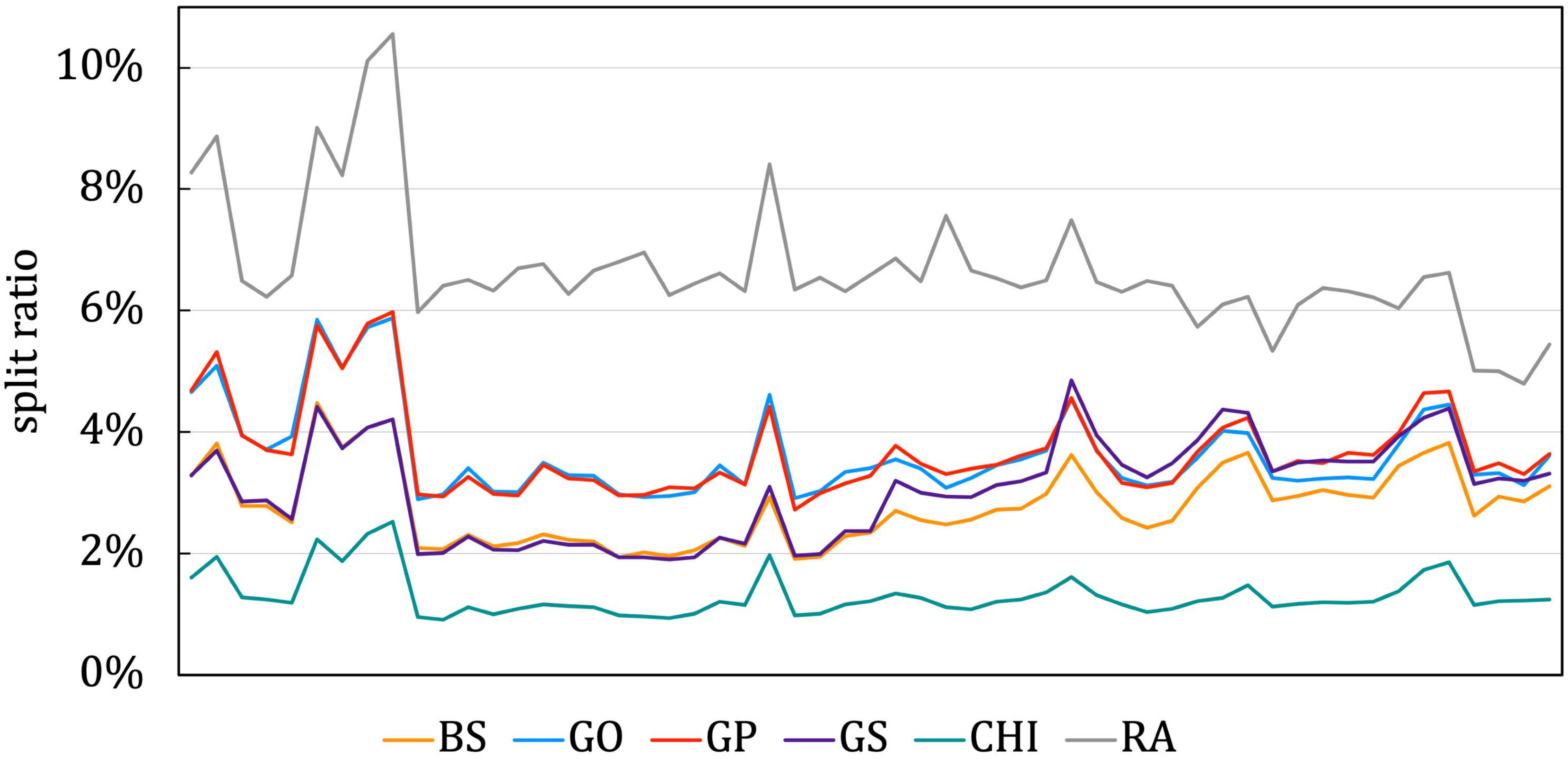


Case Study

- More than 100,000 SKUs and several millions of orders
- Comparison of **different heuristics**[1]
  - **CHI**: based on Chi-Square tests Vlćek and Voigt (2024)
  - **GP, GO, GS, BS**: based on greedy algorithms (Catalán and Fisher 2012)
  - **RA**: Random allocation of SKUs to warehouses

1. QMKP is not applicable for instance in case study

Real Data Set



Conclusion

- Splits are **of no benefit**, except **faster customer deliveries**
- **Increase workload, packaging and shipping costs**
- Mathematical Optimisation of **“full” problem not solvable**
- **CHI** Heuristic close to mathematical optimisation

And that’s it for todays lecture!

We now have covered the Quadratic Multiple Knapsack Problem and are ready to start solving some tasks in the upcoming tutorial.

Questions?

Literature

Literature I

For more interesting literature to learn more about Julia, take a look at the literature list of this course.

Catalán, Andrés, and Marshall Fisher. 2012. "Assortment Allocation to Distribution Centers to Minimize Split Customer Orders." *SSRN Electronic Journal*. https://doi.org/10.2139/ssrn.2166687.

Hiley, Amanda, and Bryant A. Julstrom. 2006. "The Quadratic Multiple Knapsack Problem and Three Heuristic Approaches to It." In *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*, edited by M. Keijzer, 547–52. New York, NY: Association for Computing Machinery. https://doi.org/10.1145/1143997.1144096.

Pitney Bowes Inc. 2017. "Pitney Bowes Parcel Shipping Index Reveals 48 Percent Growth in Parcel Volume since 2014." 2017. https://www.businesswire.com/news/home/20170830005628/en/Pitney-Bowes-Parcel-Shipping-Index-Reveals-48.