Training Material - UPRC

4. Local Optimization

4.2 Task-Specific Optimization: (IT)

4.2.2 Robot picking sequence optimization for pick-and-place tasks

Funded by
the European Union

Educational Goals of the Training

What will the user learn?

1. Understand how real-world robot pick-and-place problems map to the Bipartite Traveling Salesman Problem (BTSP)
2. Identify constraints, objective functions, and graph structures in the robot kitting task
3. Learn how to implement and compare various optimization methods:
   - Heuristics (e.g., Nearest Neighbor, 2-opt)
   - Reinforcement Learning methods (Q-learning)
   - Exact mathematical models using Integer Programming
4. Analyze how each method performs under:
   - Synthetic scenarios (equal and unequal GR-KH sets)
   - Real manufacturing data (e.g., from automotive production lines)
5. Evaluate trade-offs between solution quality and computational time
6. The input and output structure for robot task data

Contents

1.1 Introduction to Pick-and-Place (PnP) Operations

- **What are Pick-and-Place operations?**
  - Pick-and-Place (PnP) tasks involve a robotic agent moving components from a storage location (e.g., bins or racks) to a target location (e.g., assembly points or kit holders).
  - These operations are central to modern production lines, especially in kitting, packaging, and assembly.
- **Why are they important?**
  - Reduce manual labor and cycle times
  - Improve consistency, accuracy, and throughput
  - Ensure better utilization of robotic systems in constrained environments
- **Characteristics**
  - Each task involves both a picking and placing action.
  - Movements follow structured paths, often within linear tracks or robotic cells.
  - Time efficiency is critical due to repetitive and high-frequency nature of PnP

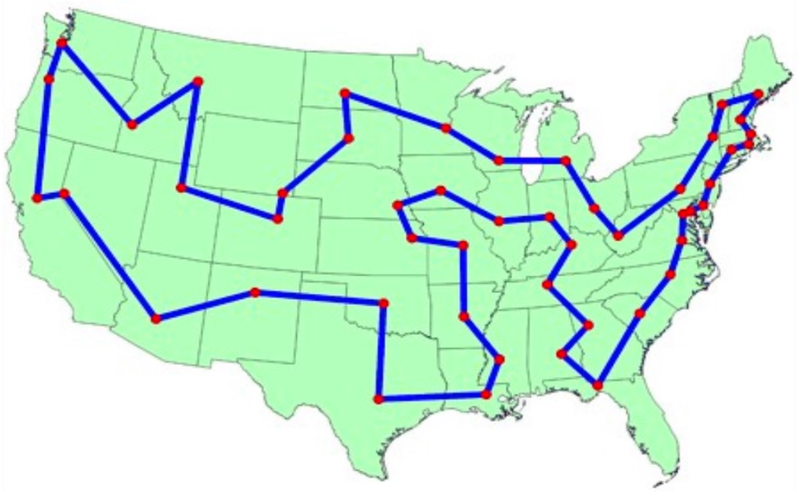
1.2 Problem Description: From TSP to BTSP
Traveling Salesman Problem (TSP):

- **What is the TSP?**
  - The **Traveling Salesman Problem (TSP)** is one of the most fundamental and well-studied problems in combinatorial optimization.
  - A salesman needs to visit a number of cities **exactly once** and return to the starting city.
  - The **goal** is to determine the **shortest possible route** that visits each city once and returns to the origin.
  - It’s a **minimization problem** over all possible permutations of the cities.

1.2 Problem Description: From TSP to BTSP

Traveling Salesman Problem (TSP):

- Given the costs required for travelling between every node pair, the standard Traveling Salesman Problem (TSP) is aimed at generating the route (node sequence) that:
  - The total cost of the route is minimized
  - The route is a cycle (the route begins from the central location and
  - returns to the central station)
  - The route contains all nodes just once



1.2 Problem Description: From TSP to BTSP

Traveling Salesman Problem (TSP):

- The Traveling Salesman Problem (TSP) is defined on a weighted complete graph $G=(V,E)$, where:
- V is a set of vertices (cities/locations)
- E is a set of edges between all pairs of vertices
- Each edge $(i,j)\in E$ has a non-negative weight d(i,j), representing the cost (distance, time, etc.) to travel between city i and city j.

1.2 Problem Description: From TSP to BTSP
Further Reading on the TSP

- **The Traveling Salesman: Computational Solutions for TSP Applications**
  Gerhard Reinelt (2003)
  - Offers both theory and practical methods.
  - Includes benchmark instances used in research and competitions.
  - Great for understanding **experimental evaluation** and **implementation aspects**.
- **Traveling Salesman Problem: Theory and Applications**
  Donald Davendra (Ed.) (2010)
  - A collection of research works combining **theory**, **applications**, and **novel approaches**.
  - Discusses **metaheuristics**, **multi-objective variants**, and **real-world industrial scenarios**.
  - A good read for those interested in **diverse formulations and adaptations** of the TSP.
- **The Traveling Salesman Problem and Related Problems**
  Gavish & Graves (1978)
  - Foundational paper linking TSP with logistics and manufacturing.
- **Literature Review on Travelling Salesman Problem**
  Chetna Dahiya & Shabnam Sangwan (2018)
  - A broad **survey** of TSP variants, algorithms, and **modern trends**.
  - Useful for quick **overview of solution techniques** and applications.

1.2 Problem Description: From TSP to BTSP
Q&A: TSP Understanding (1/2)

1. **What is the main objective of the TSP?**
   A. Visit the same city multiple times
   B. Minimize the number of routes
   C. Minimize the travel distance and return to the origin
   D. Visit only half of the cities
2. **Which algorithm is exact for solving small TSPs?**
   A. Nearest Neighbor
   B. Simulated Annealing
   C. Integer Programming
   D. None of the above
3. **What makes TSP difficult to solve for large instances?**
   A. It's a linear problem
   B. It requires multiple start points
   C. Number of possible tours grows factorially
   D. It doesn’t return to the starting city
4. **Which of the following is an application of the TSP?**
   A. Weather prediction
   B. Circuit board drilling
   C. Password encryption
   D. All of the above

1.2 Problem Description: From TSP to BTSP
Q&A: TSP Understanding (2/2)

- Answers:
  - 1. C
  - 2. C
  - 3. C
  - 4. B

1.2 Problem Description: From TSP to BTSP Bipartite Traveling Salesman Problem (BTSP):

- **What is the BTSP?**
  - The **Bipartite Traveling Salesman Problem (BTSP)** is a variation of the classical TSP.
  - In BTSP, the nodes are divided into two disjoint sets, commonly referred to as:
    - **Set A** (e.g., Gravity Racks or Pick Locations)
    - **Set B** (e.g., Kit Holders or Place Locations)
  - A valid tour **must alternate** between nodes in Set A and Set B.
- **Why is this important?**
  - BTSP models real-world industrial tasks, especially in **Pick-and-Place operations** where a robot picks items from one type of location and places them in another.
  - This alternation reflects real constraints in manufacturing, such as:
    - A robot **cannot pick two items in a row** or **place two items in a row** without violating physical/logical rules.

1.2 Problem Description: From TSP to BTSP

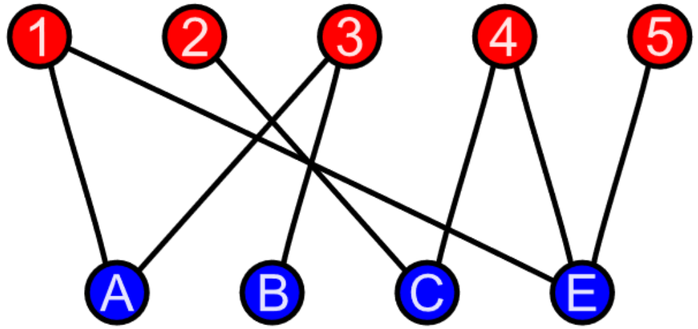
Bipartite Traveling Salesman Problem (BTSP):

- Find the **shortest possible alternating tour** that:
  - Starts and ends at specific nodes (e.g., depot or rest position)
  - Visits each node **once**
  - Alternates between **picking and placing**
- A valid tour must **alternate** between nodes in Set A and Set B.
- The graph is **bipartite**, meaning edges only connect nodes from A to B and vice versa, not within the same set.
- Objective: **Minimize total cost** (e.g., time or distance) while visiting each node **exactly once**, respecting the alternation rule.
- The problem is **NP-hard**, like the classical TSP.
- The solution methods are the same with the classical TSP. The solution methods will be covered in depth in the course.

1.2 Problem Description: From TSP to BTSP

Bipartite Traveling Salesman Problem (BTSP):

- **Definition**:
  A **bipartite graph** is a graph where the set of vertices V can be divided into two **disjoint sets**, V_1 and V_2, such that **every edge** connects a vertex in V_1 to one in V_2. – *Skiena (1990)*



- **Special Case – Complete Bipartite Graph**:
  If **every** vertex in V_1 is connected to **every** vertex in V_2, the graph is called a **complete bipartite graph**.

1.2 Problem Description: From TSP to BTSP
Further Reading on the BTSP

- **A Note on the Polytope of Bipartite TSP**
  Kovács, G., Tuza, Z., Vizvári, B., Jabbari, H. K. (2018)
  - Explores polyhedral differences between BTSP and classical TSP.
  - Proves that **comb inequalities** are not facet-defining for BTSP.
  - Essential for researchers interested in LP/IP formulations and cutting-plane techniques.
- **The Bipartite Travelling Salesman Problem: A Pyramidally Solvable Case**
  Deineko, Klinz & Woeginger (2024)
  - Identifies special cases of BTSP that are solvable in polynomial time.
  - Enhances theoretical understanding and guides practical algorithm design.
- **Solution for a Bipartite Euclidean Traveling-Salesman Problem in One Dimension**
  Caracciolo, S., Di Gioacchino, A., Gherardi, M., Malatesta, E. M. (2018)
  - Presents closed-form solution for **1D bipartite Euclidean TSP**.
  - Connects statistical physics with BTSP optimization theory.
  - Provides a theoretical baseline for evaluating algorithm performance.

1.2 Problem Description: From TSP to BTSP
Q&A: BTSP Understanding (1/2)

1. **What is a unique constraint in BTSP?**
   A. You can only travel between nodes in the same set
   B. You can revisit nodes
   C. You must alternate between two sets of nodes
   D. None of the above
2. **Which classic method can be adapted to solve BTSP?**
   A. Brute Force
   B. Integer Programming
   C. Nearest Neighbor
   D. All of the above
3. **What happens if a BTSP tour does not alternate between sets?**
   A. The result is optimal
   B. The solution is invalid
   C. It improves execution time
   D. It simplifies the problem

1.2 Problem Description: From TSP to BTSP
Q&A: BTSP Understanding (2/2)

- Answers:
  - 1. C
  - 2. D
  - 4. B

1.2 Problem Description: From TSP to BTSP
TSP vs BTSP:

Limitations of TSP in PnP context:

- TSP assumes a single set of nodes (locations).
- PnP involves two types of operations: picking and placing, which naturally map to two different sets of locations.

Why BTSP is ideal for robotics:

- Reflects the real sequence of tasks (pick → place).
- Can encode time costs per move (e.g., pick/place time, horizontal travel time).
- Supports custom constraints like start/end locations, container pairing, or skip rules

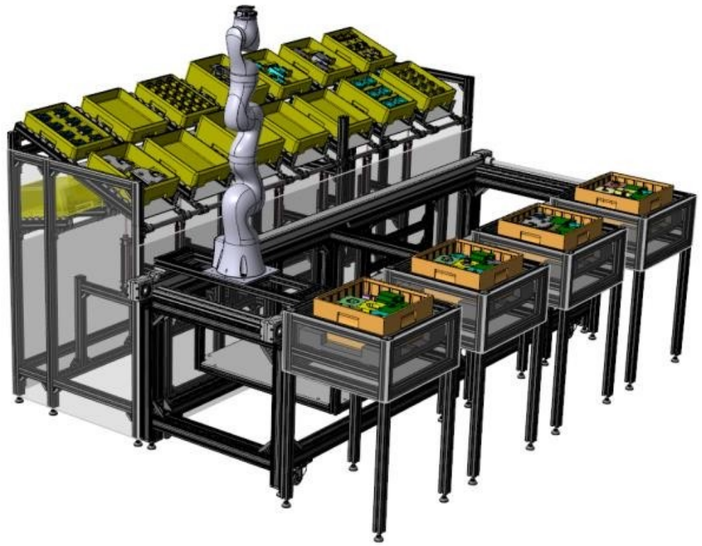
1.3 Problem Modeling and Mathematical Formulation
Problem Structure

- We solve a Pick-and-Place problem modeled as a Bipartite TSP.

- Nodes are divided into two disjoint sets:
  - **Gravity Rack (GR)**: Holds components to be picked
  - **Kit Holders (KH)**: Designated blocks where items are placed

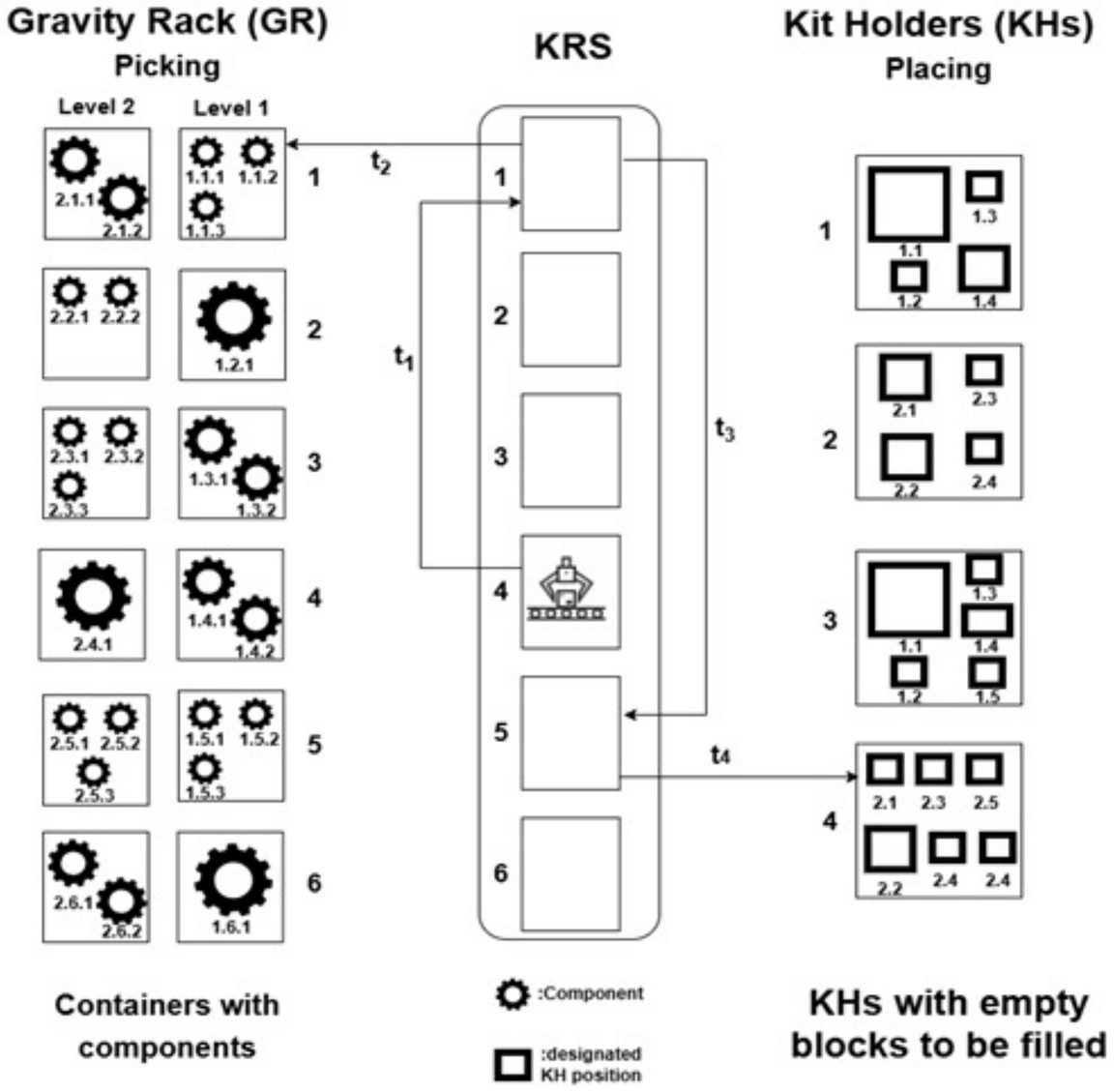
- The transfer agent (robot) must alternate: pick a part ➔ place it ➔ pick another part ➔ place it, and so on.

- Objective: Minimize total time/cost while fulfilling all pick-and-place tasks.

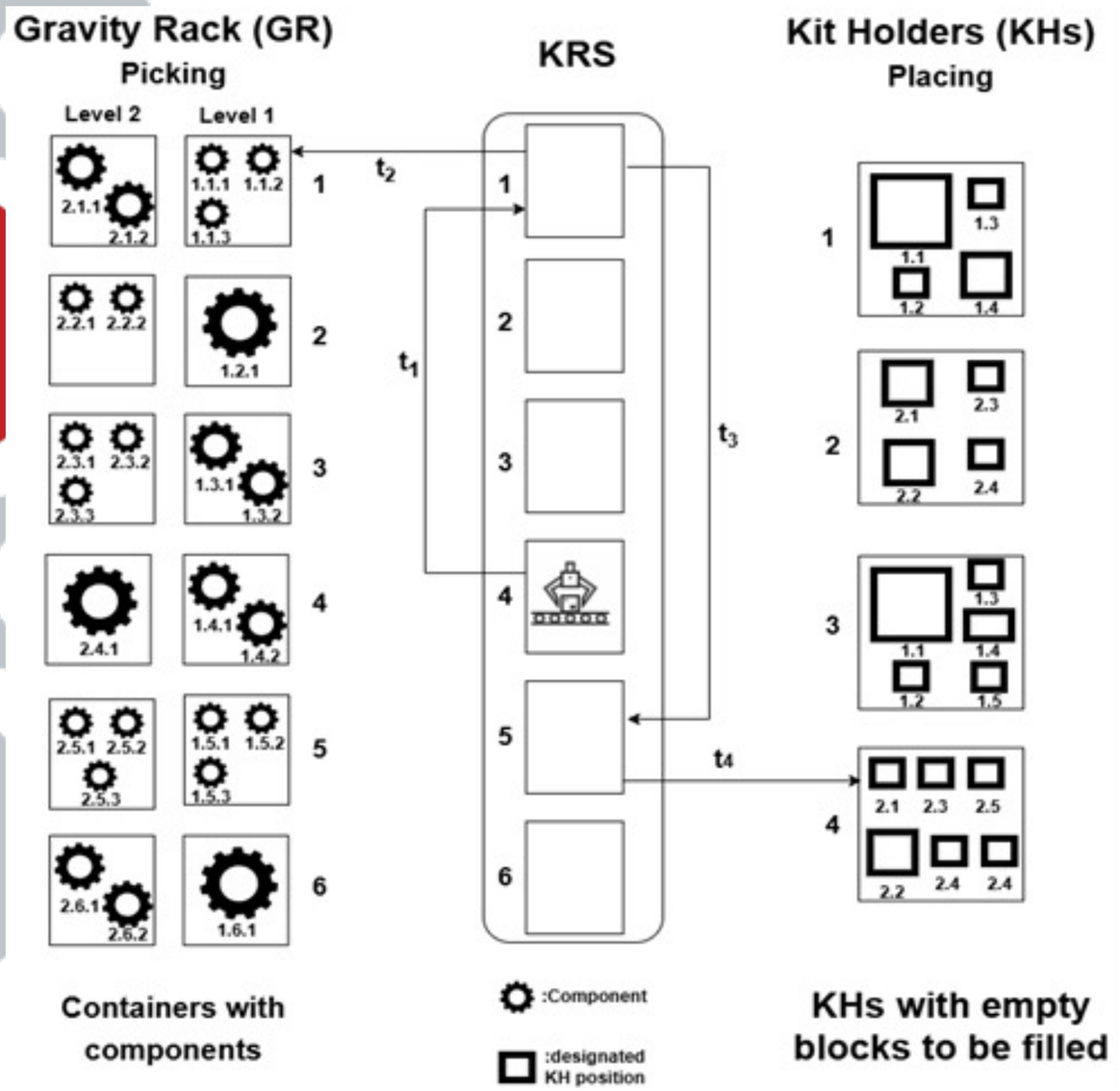


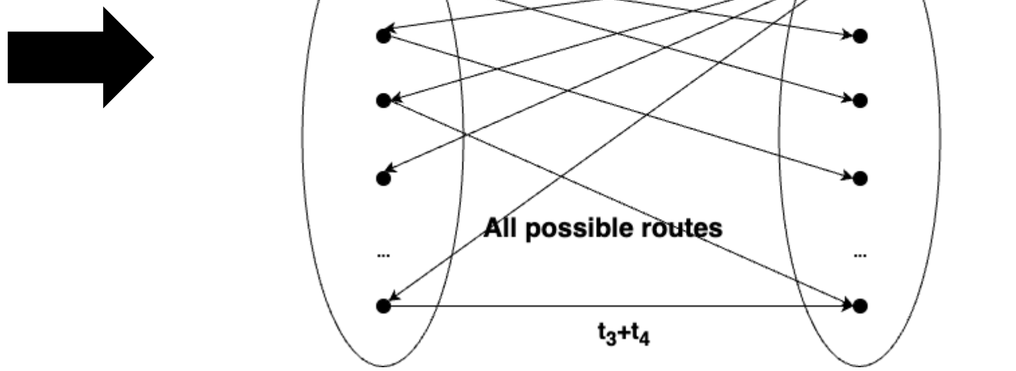
1.3 Problem Modeling and Mathematical Formulation
Modeling of BTSP

- Left: Gravity Rack where parts are picked.
- Middle: KRS moves parts between locations.
- Right: Kit Holders where components must be placed.
- Every pick must be immediately followed by a corresponding place.
- The final tour starts and ends at the transfer agent's depot position.
- Each pick location = node in Set 1
- Each place location = node in Set 2



1.3 Problem Modeling and Mathematical Formulation
Formulation of the Problem

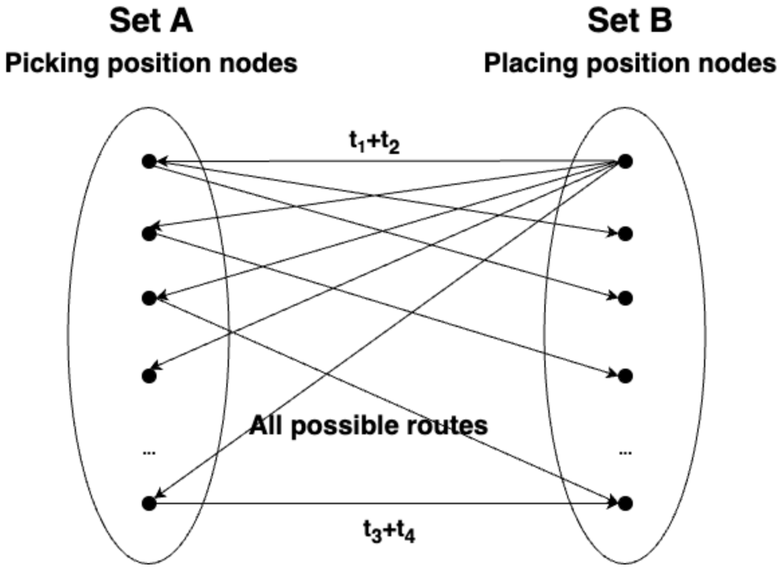




- **Computing edge weights**
- **Picking time breakdown**
- t_1: time to move to the picking safe position
- t_2: time to pick and return to safe position
- **Placing time breakdown**
- t_3: time to move to placing safe position
- t_4: time to place and return to safe position

1.3 Problem Modeling and Mathematical Formulation Properties of the BTSP

- Strict alternating sequence between two sets (no two consecutive picks or places).
- Bipartite graph representation: edges only connect nodes from different sets.
- Shares NP-hard complexity with the classical TSP but has specialized structure that affects solution methods.



1.3 Problem Modeling and Mathematical Formulation Mathematical Formulation (Integer Programming)

- Objective: Minimize total travel time while alternating between pick and place locations.
- We solve an IP model.
- Elements:
- Nodes are divided into two sets: Picking (A) and Placing (B).
- Edges connect nodes from different sets only.
- Binary decision variables: x_ij= 1 if edge (i,j) is used, 0 otherwise.

$$\min \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij} \tag{1}$$

s.t.

$$\sum_{i=1}^{|A|} x_{ij} \leq 1, \quad \forall j \in B \tag{2}$$

$$\sum_{i=1}^{|B|} x_{ij} = 1, \quad \forall j \in A \tag{3}$$

$$\sum_{i=1}^{m} x_{ik} = \sum_{j=1}^{n} x_{kj}, \quad \forall k \in A \cup B \tag{4}$$

$$x_{ij} \in \{0,1\} \quad \forall (i,j) \in E_G \tag{5}$$

$$\sum_{i \in Q} \sum_{j \neq i, j \in Q} x_{ij} \leq |Q| - 1, \quad \forall Q \subseteq A \cup B, |Q| \geq 2. \tag{6}$$

1.4 Optimization Approaches
Introduction to Optimization

What is Optimization in Industrial Robotics?

Optimization is the process of finding the *best possible solution* from a set of feasible alternatives—typically by **minimizing cost**, **time**, or **energy** under specific constraints. In robotic pick-and-place operations, optimization means planning the robot's movements in a way that:

- Reduces total operation time (e.g., travel, pick, and place durations)
- Fulfills all required tasks (e.g., filling all Kit Holder positions)
- Respects physical and operational constraints (e.g., movement patterns, start/end locations)

1.4 Optimization Approaches
Introduction to Optimization

- **Why Do We Optimize?**
  - To **improve productivity** and reduce cycle time on the shop floor
  - To **minimize costs** associated with robot operations and idle time
  - To **streamline logistics** and avoid bottlenecks in assembly preparation
  - To **support digital twins** and real-time planning tools with optimal sequences
- **How Is the Optimization Done?**

  We apply **algorithmic methods** to evaluate and choose the best movement sequence based on:
  - **Graph traversal models** (e.g., TSP, BTSP)
  - **Cost functions** (e.g., time per move, distance, task priority)
  - **Search and learning techniques** (e.g., heuristics, reinforcement learning, integer programming)

1.4 Optimization Approaches
Methods to Solve the Robot Picking Sequence Problem

We apply five distinct approaches to solve the BTSP:

- **Method 1: Nearest Neighbor (NN) Heuristic**
  - A fast greedy method that builds a tour by selecting the closest unvisited node at each step.
- **Method 2: 2-opt Local Optimization**
  - A refinement technique that improves an existing tour by swapping node pairs to reduce total cost.
- **Method 3: Q-learning (Reinforcement Learning)**
  - A learning-based method that trains the robot to discover efficient sequences through reward feedback.
- **Method 4: Exact Method (Integer Programming)**
  - A mathematically precise solution based on graph modeling and iterative subtour elimination.
- **Method 5: Linear Method (Baseline Logic)**
  - A rule-based method currently used in real production systems, optimized for feasibility and simplicity.

1.4 Optimization Approaches
Method 1: Nearest Neighbor (NN) Heuristic

- **Goal:** Quickly construct a tour by visiting the nearest unvisited node.
- **Adaptation for BTSP:** Alternate between picking (set A) and placing (set B).

How the Nearest Neighbor Algorithm Works

- At each step:
  - If at a picking node (A), select the closest unvisited placing node (B).
  - If at a placing node (B), select the closest unvisited picking node (A).

- Special Handling: After visiting all placing nodes, insert the pseudonode representing "return to start."

- Greedy Nature: Chooses the nearest node without backtracking or looking ahead.

1.4 Optimization Approaches
Method 1: Nearest Neighbor (NN) Heuristic

Nearest Neighbor TSP for BTSP

Pseudocode Summary Comments:

- **Initialization (Lines 1–3):**
  - Start from a fixed node (usually 0.0) and mark it visited. Set up tracking for visited nodes and alternate groups.
- **First Move to Set A (Line 4):**

  From the starting position, select the nearest unvisited node in set_A (picking nodes).
- **Main Loop (Lines 9–23):**

  Alternate between set_A and set_B:
  - If at a picking node (set_A), select nearest unvisited placing node (set_B)
  - If at a placing node (set_B), select nearest unvisited picking node (set_A)
  - Stop when all required KH positions are visited.
- **Greedy Design:**

  Always picks the closest neighbor, doesn't backtrack. Fast but not always optimal.
- **Finalization (Line 20):**

  Insert the end pseudonode (0.0.0) only after all KHs are filled to complete the tour

Algorithm 1 Nearest Neighbor TSP

Require: Graph G, start node *start*, end node *end*, sets *set_B*, *set_A*, threshold *large_value*

Ensure: A tour visiting all nodes and returning to *start*

```
1: Initialize visit[node] ← False for all nodes
2: tour ← [start], visit[start] ← True, current ← start
3: kit_holders ← nodes in set_B based on criteria
4: Find nearest next_node in set_A to start
5: if next_node = None then
6:     raise error
7: end if
8: add next_node to tour, visit[next_node] ← True, current ← next_node
9: while not all nodes visited do
10:    if current ∈ set_A then
11:        next_node ← find nearest unvisited neighbor in set_B
12:    else
13:        next_node ← find nearest unvisited neighbor in set_A
14:    end if
15:    if next_node = None then
16:        break
17:    end if
18:    add next_node to tour, visit[next_node] ← True, current ← next_node
19:    if all kit_holders visited then
20:        add end node to tour
21:        break
22:    end if
23: end while
24: return tour
```

1.4 Optimization Approaches
Further Reading on Heuristic

How to Solve It: Modern Heuristics
Michalewicz & Fogel (2004)

- A broad and engaging guide to modern heuristic strategies
- Covers greedy methods, constructive heuristics, and introduces randomness and hybrid methods
- Very practical, includes real-world problem examples

Heuristics and Metaheuristics for Combinatorial Optimization
Blum & Roli (2003)

- Very accessible and widely cited overview paper
- Explains the role of heuristics in complex search spaces
- Includes classification and use cases

Heuristic Research: Design, Methodology, and Applications”
Clark Moustakas (1990)

- Focuses on heuristics as a qualitative research approach
- Explores problem-solving as discovery, useful for conceptual thinking
- Valuable for understanding human-centered and exploratory heuristics

1.4 Optimization Approaches
Q&A: Heuristic Methods (1/2)

1. **Why do industries often prefer heuristics for large-scale problems?**
   A. Because they guarantee optimal solutions
   B. Because they are mathematically elegant
   C. Because they offer fast, good-enough solutions
   D. Because they require no data
2. **Which of the following is a common feature of heuristic methods?**
   A. Always produce the global optimum
   B. Use rule-of-thumb strategies to explore solutions
   C. Work only with exact data
   D. Require advanced solvers like Gurobi
3. **What makes the Nearest Neighbor heuristic useful in logistics?**
   A. It greedily selects the next closest unvisited location
   B. It evaluates all possible paths
   C. It finds the longest path possible
   D. It simulates every route variation
4. **What is one limitation of heuristics?**
   A. They are too slow to use in industry
   B. They often need AI to run
   C. They can get stuck in suboptimal solutions
   D. They are illegal in regulated sectors

1.4 Optimization Approaches
Q&A: Heuristic Methods (2/2)

- Answers:
  - 1. C
  - 2. B
  - 3. A
  - 4. C

1.4 Optimization Approaches
Method 2: 2-opt Local Optimization

- **Goal:** Improve an initial tour by swapping edges to reduce total travel cost.
- **Optimization Logic:**
  - Evaluate all possible 2-edge swaps.
  - If a swap reduces the total tour cost, apply it.

How the 2-opt Algorithm Works

- Starting Point: Tour generated by the NN algorithm.
- Iteratively examine pairs of edges.
- Swap two edges and reconnect the tour to see if the total distance decreases.
- Accept the swap only if it improves the total cost.
- Repeat until no better solution is found (local minimum).

1.4 Optimization Approaches
Method 2: 2-opt Local Optimization

2-Opt Optimization Algorithm Pseudocode

Summary Comments:

- **Core Idea (Lines 1–15):**

  Try all pairs of non-adjacent edges in the tour and reverse them (swap) if it reduces total cost and maintains valid connections.

- **Edge Validation (Line 8):**

  Only perform swap if both new edges exist (e.g., obey GR↔KH alternation constraints).

- **Apply Move (Lines 16–19):**

  Once a valid swap is identified, it's applied to create a new tour.

- **Optimization Loop (Lines 20–34):**

  Keep applying best possible 2-opt moves until no further improvement is found.

- **Fallback Handling (Lines 21–23):**

  If no initial tour is provided, it generates one using Nearest Neighbor.

- **Bipartite-Aware:**

  This 2-opt is adapted for bipartite constraints—it doesn't allow swaps that break GR→KH or KH→GR alternation.

Algorithm 2 2-Opt Optimization Algorithm

Require: Tour *tour*, Graph *graph*, sets *set_B*, *set_A*
Ensure: An optimized tour using 2-opt moves

```
1:  function FIND_BEST_TWO_OPT_MOVE(tour, graph, set_B, set_A)
2:      best_move_cost ← 0, best_tour ← tour
3:      for i ← 0 to len(tour) − 2 do
4:          for j ← i + 2 to len(tour) − 1 do
5:              Calculate current and new costs for edges (i, i + 1) and (j, j + 1)
6:              Calculate new cost after swapping edges
7:              move_cost ← new_cost − current_cost
8:              if move_cost < best_move_cost and both new edges exist then
9:                  best_move_cost ← move_cost
10:                 best_tour ← APPLY_TWO_OPT_MOVE(tour, i, j)
11:             end if
12:         end for
13:     end for
14:     return best_tour, best_move_cost
15: end function
16: function APPLY_TWO_OPT_MOVE(tour, i, k)
17:     new_tour ← the part of tour before i + 1, the reversed segment between
    i + 1 and k, and the part after k
18:     return new_tour
19: end function
20: function OPT2(tour, graph, set_B, set_A, start, end)
21:     if tour = None and start ≠ None and end ≠ None then
22:         tour ← Nearest_TSP(graph, start, end, set_B, set_A)
23:     end if
24:     best_tour ← tour, best_cost ← total_cost(graph, best_tour)
25:     while improved is True do
26:         improved ← False
27:         new_tour, move_cost ← find_best_two_opt_move(best_tour, graph,
    set_B, set_A)
28:         if move_cost < 0 then
29:             best_tour ← new_tour, best_cost ← best_cost + move_cost
30:             improved ← True
31:         end if
32:     end while
33:     return best_tour
34: end function
```

1.4 Optimization Approaches
Further Reading on Metaheuristics

Essentials of Metaheuristics (Free PDF)
Sean Luke (2016)

- Open-access, beginner-friendly with pseudocode and use cases

Heuristics and Metaheuristics for Combinatorial Optimization

Blum & Roli (2003)

- Very accessible and widely cited overview paper
- Explains the role of heuristics in complex search spaces
- Includes classification and use cases

Heuristic Research: Design, Methodology, and Applications”

Clark Moustakas (1990)

- Focuses on heuristics as a qualitative research approach
- Explores problem-solving as discovery, useful for conceptual thinking
- Valuable for understanding human-centered and exploratory heuristics

1.4 Optimization Approaches
Q&A: Metaheuristics (1/2)

1. **What is the main goal of a metaheuristic algorithm?**
   A. To test every possible solution
   B. To approximate a good solution efficiently
   C. To find the exact optimal solution every time
   D. To simulate human behavior only
2. **Which of the following is a feature of Genetic Algorithms (GA)?**
   A. Uses temperature to accept worse solutions
   B. Applies pheromone trails for guidance
   C. Uses crossover and mutation operations
   D. Prevents movement with a memory list
3. **Why are metaheuristics suitable for large-scale industrial problems?**
   A. They can guarantee global optimality
   B. They require very few parameters
   C. They adapt well and explore solution spaces efficiently
   D. They only apply to small instances
4. **What does “exploitation” mean in a metaheuristic context?**
   A. Randomly testing as many options as possible
   B. Ignoring the best areas of the solution space
   C. Focusing search around high-quality known solutions
   D. Repeating the same solution until optimal

1.4 Optimization Approaches
Q&A: Metaheuristics(2/2)

- Answers:
  - 1. B
  - 2. C
  - 3. C
  - 4. C

1.4 Optimization Approaches
Method 3: Q-learning (Reinforcement Learning)

Goal: Learn a near-optimal picking and placing sequence by interacting with the problem environment instead of hardcoding logic.

What is Q-learning?

- Q-learning is a reinforcement learning algorithm used to find the best sequence of actions by **learning from trial and error.** It does not require a model of the environment and is well-suited for **sequential decision-making** in industrial operations.

How It Works in Pick-and-Place BTSP

State (s): Current node (e.g., GR or KH location)

Action (a): Move to a connected, valid next node

Reward (r): Negative of the time cost (lower time = higher reward)

Q(s,a): Expected value of taking action a from state s

1.4 Optimization Approaches
Method 3: Q-learning (Reinforcement Learning)

1. **Initialize Q-values**:

For every node and each of its neighbors, initialize the Q-value to zero. Updates follow the Bellman equation (Sutton, 2018):

$$Q_{new}(s,a) \leftarrow (1-\alpha) * Q(s,a) + \alpha * [r + \gamma * max_{a'} Q(s',a')]$$

- **s:** the current state (node)
- **a:** the action taken (i.e., the transition to a neighbor node)
- **r:** the immediate reward received for the action (typically, the negative of the travel time)
- **s':** the next state after taking action a
- $max_{a'} Q(s',a')$**:** the maximum estimated future reward for all possible actions a' from state s'
- **α:** the learning rate (weight for new information)
- **γ:** the discount factor (importance of future rewards)

1.4 Optimization Approaches
Method 3: Q-learning (Reinforcement Learning)

2. **Start each episode from the initial node**:

   Begin at the predefined start node (e.g., the robot's resting position), and initialize an empty tour.

3. **Alternate between node sets**:

   To preserve the bipartite nature of the problem, the algorithm alternates between gravity rack nodes and kit holder nodes.

4. **Select the next node**:
   - With probability ε, select a random valid neighbor (**exploration**),
   - Otherwise, select the neighbor with the highest Q-value (**exploitation**).

5. **Compute the reward**:

   Calculate the immediate cost of transition (e.g., the time required to move and pick/place) and derive the reward accordingly.

6. **Update the Q-value**:

   Use the Bellman equation to revise the Q-value for the selected transition, incorporating both the immediate reward and the best future value.

7. **Repeat until all required nodes are visited**:

- The process continues until all kit holder and gravity rack positions are visited according to the constraints of the problem. Once training is completed, the final tour is extracted by greedily following the learned Q-values.

1.4 Optimization Approaches
Further Reading on RL

Reinforcement Learning: An Introduction

Richard S. Sutton & Andrew G. Barto (2015)

- The foundational book on RL theory and algorithms
- Covers key concepts like Q-learning, policy gradients, and exploration

A Survey on Reinforcement Learning for Industry 4.0

Ahmad Farooq, Kamran Iqbal (2025)

- Covers real-world RL applications in smart factories, energy systems, and logistics
- Highlights challenges, opportunities, and emerging trends in Industry 4.0 environments

Algorithms for Reinforcement Learning

Csaba Szepesvári (2010)

- Compact and mathematically grounded introduction to RL
- Focuses on value functions, exploration, and algorithm convergence

1.4 Optimization Approaches
Q&A: Reinforcement Learning (1/2)

1. **What does a reinforcement learning agent try to maximize?**
   A. The number of actions it takes
   B. The complexity of the environment
   C. The total accumulated reward over time
   D. The number of available states
2. **In a warehouse, what could be considered the “environment” in RL terms?**
   A. The item list
   B. The warehouse layout, shelves, and item locations
   C. The worker’s salary
   D. The optimization software
3. **Why is reward design important in RL?**
   A. It determines how fast the computer runs
   B. It limits the number of states
   C. It guides the agent's learning behavior
   D. It tells the agent which sensor to use
4. **What makes Deep RL different from basic RL?**
   A. It uses multiple agents
   B. It includes hardcoded policies
   C. It applies neural networks to handle complex inputs
   D. It always finds the exact optimal solution

1.4 Optimization Approaches
Q&A: Reinforcement Learning (2/2)

- Answers:
  - 1. C
  - 2. B
  - 3. C
  - 4. C

1.4 Optimization Approaches
Method 4: Exact Method (Integer Programming)

Goal: Solve the Bipartite Traveling Salesman Problem (BTSP) **exactly**, producing the **mathematically optimal pick-and-place sequence**.

What is the Exact Method?

This method formulates the BTSP as a **Mixed Integer Programming (MIP)** problem, using a set of binary variables and constraints to describe valid robot movements.

- The solution is obtained by solving this model using solvers like **CBC**, **CPLEX**, or **Gurobi**, with **subtour elimination** to ensure a valid cyclic tour.

Explanation of the Mathematical Model:

- Objective function (1): Minimize the sum of travel time
- Constraints (2) and (3): Ensure each node in A and B is visited exactly once.
- Flow conservation (4): If a tour enters a node, it must also exit.
- Binary variables (5): Each connection is either selected or not.
- Subtour elimination (6): Prevent small loops that do not visit all nodes.
- Special condition: $|A| \geq |B|$ to ensure all placements are fulfilled.

$$\min \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij} \tag{1}$$

s.t.

$$\sum_{i=1}^{|A|} x_{ij} \leq 1, \quad \forall j \in B \tag{2}$$

$$\sum_{i=1}^{|B|} x_{ij} = 1, \quad \forall j \in A \tag{3}$$

$$\sum_{i=1}^{m} x_{ik} = \sum_{j=1}^{n} x_{kj}, \quad \forall k \in A \cup B \tag{4}$$

$$x_{ij} \in \{0,1\} \quad \forall (i,j) \in E_G \tag{5}$$

$$\sum_{i \in Q} \sum_{j \neq i, j \in Q} x_{ij} \leq |Q| - 1, \quad \forall Q \subseteq A \cup B, |Q| \geq 2. \tag{6}$$

1.4 Optimization Approaches
Method 4: Exact Method (Integer Programming)

Solving the BTSP Exactly:

- Solve the initial IP model (1)-(5).
- **If subtours exist**:
  1. Detect subtours using graph traversal (e.g., DFS).
  2. Add subtour elimination constraints (6).
  3. Re-solve the updated model.
- Repeat until a single valid tour is found covering all pick and place locations.
- Solver used: **CBC (Coin-or Branch and Cut)**.

$$\min \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij} \tag{1}$$

s.t.

$$\sum_{i=1}^{|A|} x_{ij} \leq 1, \quad \forall j \in B \tag{2}$$

$$\sum_{i=1}^{|B|} x_{ij} = 1, \quad \forall j \in A \tag{3}$$

$$\sum_{i=1}^{m} x_{ik} = \sum_{j=1}^{n} x_{kj}, \quad \forall k \in A \cup B \tag{4}$$

$$x_{ij} \in \{0,1\} \quad \forall (i,j) \in E_G \tag{5}$$

$$\sum_{i \in Q} \sum_{j \neq i, j \in Q} x_{ij} \leq |Q| - 1, \quad \forall Q \subseteq A \cup B, |Q| \geq 2. \tag{6}$$

1.4 Optimization Approaches
Further Reading in Integer Programming

Model Building in Mathematical Programming

H. Paul Williams (5th Edition, 2013)

- One of the most practical and readable texts on LP, MILP, and modeling techniques
- Focuses on real-world industrial problems, including production, blending, scheduling, and logistics
- Includes model interpretation, dual values, and many industry-based examples

Linear Programming: Foundations and Extensions

Robert J. Vanderbei

- A theoretical yet accessible textbook for LP and extensions like duality, simplex, and interior-point methods
- Covers both foundational math and practical modeling logic
- Suitable for learners who want to go beyond applications into method mechanics

1.4 Optimization Approaches
Q&A: Integer Programming (1/2)

1. **What kind of real-world problems are addressed in H. Paul Williams' book?**
   A. Data mining and neural networks
   B. Component kitting and robot programming
   C. Production, blending, scheduling, and logistics
   D. Compiler optimization and code generation

2. **What is a strength of Robert Vanderbei's book *Linear Programming: Foundations and Extensions*?**
   A. Balances theoretical LP concepts with practical modeling
   B. Purely algorithmic focus without theory
   C. Focus on quantum optimization
   D. Avoids topics like simplex or duality for simplicity

3. **Why is Integer Programming often preferred in industrial settings like robot pick-and-place optimization?**
   A. Because it is the fastest method for real-time control
   B. Because it allows modeling of discrete decisions, like sequencing and assignments
   C. Because it does not require a solver
   D. Because it guarantees suboptimal but fast solutions

4. **Why is solving Integer Programming (IP) generally harder than Linear Programming (LP)?**
   A. Because IP uses more memory
   B. Because IP must explore many discrete combinations
   C. Because LP is not real math
   D. Because IP has no objective function

5. **In the context of the BTSP robot optimization case, why do we use Integer Programming with subtour elimination?**
   A. To ensure the robot ignores the kit holders
   B. To remove continuous variables and improve performance
   C. To simplify the model for heuristic application
   D. To guarantee that all nodes are visited in a valid alternating sequence

1.4 Optimization Approaches
Q&A: Integer Programming (2/2)

- Answers:
  - 1. C
  - 2. A
  - 3. B
  - 4. B
  - 5. D

1.4 Optimization Approaches
Method 5: Linear Method (Baseline Logic)

Goal: Simulate the **real-world logic** currently used on the manufacturing floor—prioritizing simplicity and execution feasibility.

How the Linear Method Works

- Follows a **fixed sequence** of Kit Holders (KHs), such as:
  - KH1 → KH2 → KH3 → KH4
- For each KH:
  - Find the **nearest GR location** (that has the needed component)
  - Move robot to that GR location → **pick component**
  - Move to current KH position → **place component**
- Repeat for all KHs in order.
- Finally, **return to predefined end node** (e.g., robot home position)

Limitations

- **No global optimization**: may miss better overall routes
- **Inflexible**: does not adapt to component location changes
- **Does not account** for total cost minimization or alternate paths

1.5 Input and Output Structure
Input Data

- Route:

The URL or path to the optimization service endpoint used to request or retrieve results

- uuid:

A unique identifier for this specific solution instance — useful for querying past results

- generated_at:

The timestamp indicating when the result was generated, formatted in ISO 8601 standard

- Method:

The optimization strategy to be applied (e.g., "exact-linear" in this case)

- KH Sequences:

List of kit holder (KH) setups the robot must fill in sequence (e.g., ["KH003", "KH002", "KH001", "KH001"]


```
{
    "route": "robot-pick-seq-opt",
    "uuid": "20575aa3-a69a-46e1-83b3-7e4ee2d68601",
    "generated_at": 1736414971,
    "data": {
        "method": "exact",
        "kh_sequences": [
            ["KH001", "KH003", "KH001", "KH002"]
        ],
        "current_config": {⟺},
        "containers_template": {⟺},
        "kit_holders_template": {⟺},
        "distance_matrix": [⟺]
    }
}
```

1.5 Input and Output Structure
Input Data

- Current Configuration

Describes available containers and their gravity rack (GR) positions.

Each container contains multiple GR positions and component types


```
{
    "route": "robot-pick-seq-opt",
    "uuid": "20575aa3-a69a-46e1-83b3-7e4ee2d68601",
    "generated_at": 1736414971,
    "data": {
        "method": "exact",
        "kh_sequences": [...],
        "current_config": {
    "containers": {
        "Container_1": {
            "gr_position": "1.1",
            "contents": [
                {
                    "position": "1.1.1",
                    "type": "Component_1"
                },
                {
                    "position": "1.1.2",
                    "type": "Component_1"
                },
                {
                    "position": "1.1.3",
                    "type": "Component_1"
                },
                {
                    "position": "1.1.4",
                    "type": "Component_1"
                }
```

1.5 Input and Output Structure
Input Data

- Containers Template:

Defines which components are stored in each container (GR location).


```
{
    "route": "robot-pick-seq-opt",
    "uuid": "20575aa3-a69a-46e1-83b3-7e4ee2d68601",
    "generated_at": 1736414971,
    "data": {
        "method": "exact",
        "kh_sequences": [⟺],
        "current_config": {⟺},
        "containers_template": {
    "Container_1": {"gr_position": "1.1","contents" : [{"position":"1.1.1","type":"Component_1"},
                                                       {"position":"1.1.2","type":"Component_1"},
                                                       {"position":"1.1.3","type":"Component_1"},
                                                       {"position":"1.1.4","type":"Component_1"}]},
    "Container_2": {"gr_position": "1.2","contents" : [{"position":"1.2.1","type":"Component_2"},
                                                       {"position":"1.2.2","type":"Component_2"},
                                                       {"position":"1.2.3","type":"Component_2"},
                                                       {"position":"1.2.4","type":"Component_2"}]},
    "Container_3": {"gr_position": "1.3", "contents": [{"position": "1.3.1", "type": "Component_3"},
                                                       {"position": "1.3.2", "type": "Component_3"},
                                                       {"position": "1.3.3", "type": "Component_3"},
                                                       {"position": "1.3.4", "type": "Component_3"}]},
```

1.5 Input and Output Structure
Input Data

- Distance Matrix:

All feasible distances between GR, KH, start (0.0), and return (0.0.0) positions

Used as the cost function for all optimization methods


```
{
    "route": "robot-pick-seq-opt",
    "uuid": "20575aa3-a69a-46e1-83b3-7e4ee2d68601",
    "generated_at": 1736414971,
    "data": {
        "method": "exact",
        "kh_sequences": [⟺],
        "current_config": {⟺},
        "containers_template": {⟺},
        "kit_holders_template": {⟺},
        "distance_matrix": [
        {
            "edge": "(0.0, 1.1.1)",
            "distance": 7515
        },
        {
            "edge": "(0.0, 1.1.2)",
            "distance": 7515
        }
```

1.5 Input and Output Structure for Simulation Output Data

- Phases:

Each optimization phase (e.g., exact, linear) contains

- Cost:

Total cost of the selected tour (e.g., 799584ms)

- Improvement:

Percentage improvement compared to a baseline method (if available)

- Solution Time:

Time in milliseconds to compute the solution

- Total Time:

Time in milliseconds to complete the optimization


```
{
    "uuid": "20575aa3-a69a-46e1-83b3-7e4ee2d68601",
    "produced_at": 1746481293481,
    "data": {
        "phases": [
            {
                "exact": {
                    "cost": 799584,
                    "time_details": [⟺]
                },
                "improvement_percentage": 19.7052
            }
        ],
        "solutionTime": 4149,
        "totalTime": 4151
    }
}
```

1.5 Input and Output Structure for Simulation Output Data

- Time Details:

Tour Sequence: Ordered list of transitions showing:

  - From → To nodes
  - Distance of each move


```
{
    "uuid": "20575aa3-a69a-46e1-83b3-7e4ee2d68601",
    "produced_at": 1746481293481,
    "data": {
        "phases": [
            {
                "exact": {
                    "cost": 799584,
                    "time_details": [
                        {
                            "from": "0.0",
                            "to": "2.2.3",
                            "distance": 13784
                        },
                        {
                            "from": "2.2.3",
                            "to": "4.3",
                            "distance": 34116
                        },
                        {
                            "from": "4.3",
                            "to": "2.7.2",
                            "distance": 10945
                        },
```

1.6 Tools & Solvers

What is a Solver?

- A solver is a software engine that takes a mathematical model (objective + constraints) and computes the optimal solution, or determines infeasibility.

Where They're Used:

- **Gurobi & CPLEX**: ERP systems, production schedulers, real-time logistics
- **CBC, GLPK, HiGHS**: Prototyping, education, low-cost solutions
- **Integrated in Tools**: Pyomo, PuLP, AMPL, JuMP, MATLAB, OR-Tools

Popular Solvers in Industry & Research

Solver	Type	Key Strengths	License
Gurobi	LP / MILP	Fastest for large-scale models, industry-grade	Commercial
CPLEX	LP / MILP	Mature, highly reliable, IBM ecosystem	Commercial
CBC	MILP	Open-source, ideal for academic and prototypes	Open-source
GLPK	LP / MILP	Lightweight, open-source, less scalable	Open-source

1.7 Simulation Framework

- The co-simulation methodology is designed to explore how different Gravity Rack (GR) configurations affect robot performance in pick-and-place operations.
  It supports evaluation under various realistic production modes, aiming to:
- Minimize total execution time
- Optimize layout and scheduling strategies
- Replicate real-world robot behavior
- This framework allows flexible testing of input scenarios while preserving operational constraints

1.7 Simulation Framework
GR Reconfiguration Concept

Content:

- Each GR layout can store components in various positions.
- **Reconfiguration:** random reassignment of component types within fixed container positions.
- **Purpose:** simulate how rearranging GR content affects pick-and-place efficiency.

Example:

Component_1 originally at GR position 1.1 → moved to 1.14
Component_16 swapped to 1.1

Result:

Different GR mappings = different robot travel paths = different execution times.

1.7 Simulation Framework
Simulation Modes

Mode	What it Does	Why it Matters
1. Single KH Sequence + One GR	Tests one fixed kit-filling task with one GR setup.	Quick validation of robot path efficiency under a known layout. Ideal for spot-checking designs.
2. Multiple KH Sequences + One GR	Runs several sequences using the same GR configuration.	Assesses performance consistency across various production runs using a shared layout.
3. Multiple KH Sequences + GR per Sequence	Assigns a unique GR layout to each KH sequence.	Simulates realistic scheduling changes and layout flexibility during production. Good for evaluating dynamic systems.
4. One KH Sequence + Many Random GRs	Keeps the task fixed but tests various GR layouts.	Helps discover the best layout for a critical sequence. Useful for GR layout optimization.
5. Many KH Sequences + Many Random GRs	Combines sequences and layout variations.	Full-scale performance testing to find robust solutions. Ideal for large-scale production planning.

1.7 Simulation Framework
Simulation Logic

Simulation Execution Steps

1. Read initial KH sequence(s) and GR layout
2. Calculate baseline total execution time
3. Generate random GR configurations
4. Simulate robot performance under each configuration
5. Compare outcomes and recommend best layout

Note:

KHs are dynamically configured per sequence, and the robot applies a **fixed 2000ms repositioning penalty** between sequences to simulate real KRS behavior

1.7 Simulation Framework
Detailed Robot Path Simulation

Simulated Robot Path & Logging

- Simulation engine uses a full distance matrix:
  - Start → GR
  - GR → KH
  - KH → End
  - End → Start (cycle reset)
- Each robot action is logged:
  - Horizontal movement
  - Pick operation
  - Place operation
- This allows full transparency and traceability of every robotic action in the simulation.

1.7 Simulation Framework
Optimal Configuration Selection

Selecting the Best GR Configuration

- Each GR configuration is simulated using the same KH sequences
- Baseline method: Linear
- Comparing method: Method 5 (Exact)
- Objective value is measured and compared to the baseline
- The layout yielding the **lowest total operation time** is identified as optimal
- Recommendation: implement most efficient GR mapping in production

1.7 Input and Output Structure for Simulation Input Data

- Route:

The URL or path to the optimization service endpoint used to request or retrieve results

- uuid:

A unique identifier for this specific solution instance — useful for querying past results

- generated_at:

The timestamp indicating when the result was generated, formatted in ISO 8601 standard

- Method:

Indicates the type of service applied.
"simulation" means the job is handled by the simulation service.

- num_random_gr_configs:

Specifies the number of random GR configurations to simulate.

- KH Sequences:

List of kit holder (KH) setups the robot must fill in sequence (e.g., ["KH003", "KH002", "KH001", "KH001"]


```
{
    "route": "robot-pick-seq-opt",
    "uuid": "20575aa3-a69a-46e1-83b3-7e4ee2d68601",
    "generated_at": 1736414971,
    "data": {
        "method": "simulation",
        "kh_sequences": [
            ["KH002", "KH002", "KH003", "KH001"]
        ],
        "num_random_gr_configs": 10,
        "containers_template": {⟺},
        "kit_holders_template": {⟺},
        "current_config": {
    "containers": {⟺}
    },
        "distance_matrix": [⟺]
    }
}
```

1.7 Input and Output Structure for Simulation Input Data

- Current Configuration

Describes available containers and their gravity rack (GR) positions.

Each container contains multiple GR positions and component types


```
{
    "route": "robot-pick-seq-opt",
    "uuid": "20575aa3-a69a-46e1-83b3-7e4ee2d68601",
    "generated_at": 1736414971,
    "data": {
        "method": "simulation",
        "kh_sequences": [
            ["KH002", "KH002", "KH003", "KH001"]
        ],
        "num_random_gr_configs": 10,
        "containers_template": {⟺},
        "kit_holders_template": {⟺},
        "current_config": {
    "containers": {
        "Container_1": {
            "gr_position": "1.1",
            "contents": [
                {
                    "position": "1.1.1",
                    "type": "Component_1"
                },
                {
                    "position": "1.1.2",
                    "type": "Component_1"
                },
                {
                    "position": "1.1.3",
                    "type": "Component_1"
                },
                {
                    "position": "1.1.4",
                    "type": "Component_1"
                }
            ]
        }
```

1.7 Input and Output Structure for Simulation Input Data

- Containers Template:

Defines which components are stored in each container (GR location).

- Kit Holders Template:

Defines which components are stored in each KH.


```
{
    "route": "robot-pick-seq-opt",
    "uuid": "20575aa3-a69a-46e1-83b3-7e4ee2d68601",
    "generated_at": 1736414971,
    "data": {
        "method": "simulation",
        "kh_sequences": [
            ["KH002", "KH002", "KH003", "KH001"]
        ],
        "num_random_gr_configs": 10,
        "containers_template": {
"Container_1": {"gr_position": "1.1","contents" : [{"position":"1.1.1","type":"Component_1"},
                                                   {"position":"1.1.2","type":"Component_1"},
                                                   {"position":"1.1.3","type":"Component_1"},
                                                   {"position":"1.1.4","type":"Component_1"}]},
```

```
{
    "route": "robot-pick-seq-opt",
    "uuid": "20575aa3-a69a-46e1-83b3-7e4ee2d68601",
    "generated_at": 1736414971,
    "data": {
        "method": "simulation",
        "kh_sequences": [
            ["KH002", "KH002", "KH003", "KH001"]
        ],
        "num_random_gr_configs": 10,
        "containers_template": {...},
        "kit_holders_template": {
"KH001": {"contents": [{"position": "", "type": "Component_5"},
                       {"position": "", "type": "Component_15"},
                       {"position": "", "type": "Component_3"},
                       {"position": "", "type": "Component_16"},
                       {"position": "", "type": "Component_14"},
                       {"position": "", "type": "Component_11"}]},
"KH002": {"contents": [{"position": "", "type": "Component_4"},
                       {"position": "", "type": "Component_2"},
                       {"position": "", "type": "Component_12"},
                       {"position": "", "type": "Component_10"}]},
"KH003": {"contents": [{"position": "", "type": "Component_9"},
                       {"position": "", "type": "Component_13"},
                       {"position": "", "type": "Component_1"},
                       {"position": "", "type": "Component_7"},
                       {"position": "", "type": "Component_10"}]}
```

1.7 Input and Output Structure for Simulation Input Data

- Distance Matrix:

All feasible distances between GR, KH, start (0.0), and return (0.0.0) positions

Used as the cost function for all optimization methods


```
{
    "route": "robot-pick-seq-opt",
    "uuid": "20575aa3-a69a-46e1-83b3-7e4ee2d68601",
    "generated_at": 1736414971,
    "data": {
        "method": "simulation",
        "kh_sequences": [
            ["KH002", "KH002", "KH003", "KH001"]
        ],
        "num_random_gr_configs": 10,
        "containers_template": {⟺},
        "kit_holders_template": {⟺},
        "current_config": {⟺},
        "distance_matrix": [
    {
        "edge": "(0.0, 1.1.1)",
        "distance": 7515
    },
    {
        "edge": "(0.0, 1.1.2)",
        "distance": 7515
    },
```

1.7 Input and Output Structure for Simulation Output Data

Baseline Performance

Shows performance of the initial configuration (the system before optimization):

- Exact baseline cost: "2349895"
- Linear baseline cost: "2904229”

Phase-based Evaluation

- exact_cost: "2367348"
  Cost when using the exact solver on this new GR configuration.
- improvement_exact: -0.7427%
  Indicates the new GR config actually performs worse (higher cost) than the baseline exact solution.
- improvement_linear: 18.4862%
  However, compared to the baseline linear method, this configuration is 18.49% better.
- gr_configuration.key
  Shows how components are rearranged in GR. Each key-value pair maps a GR slot to a component.


```
{
    "uuid": "20575aa3-a69a-46e1-83b3-7e4ee2d68601",
    "produced_at": 1746698974137,
    "data": {
        "baseline": {
            "exact": {
                "cost": "2349895"
            },
            "linear": {
                "cost": "2904229"
            },
            "baseline configuration": {…}
        },
        "phases": [
            {
                "phase": 1,
                "exact_cost": "2367348",
                "improvement_exact": -0.7427,
                "improvement_linear": 18.4862,
                "gr_configuration": {
                    "key": "1.1:Component_4-1.1:Component_4
```

1.7 Input and Output Structure for Simulation Output Data

- best_total_value: "2301429"
  The lowest cost achieved by any GR configuration tested—this is better than both baseline exact and linear.
- solutionTime & totalTime: 76770 ms (≈ 76.77 seconds)
  Time spent solving the problem. Since solution time equals total time, this was a focused simulation run.


```
{
    "uuid": "20575aa3-a69a-46e1-83b3-7e4ee2d68601",
    "produced_at": 1746698974137,
    "data": {
        "baseline": {
            "exact": {
                "cost": "2349895"
            },
            "linear": {
                "cost": "2904229"
            },
            "baseline configuration": {…}
        },
        "phases": […],
        "best_total_value": "2301429",
        "solutionTime": 76770,
        "totalTime": 76770
    }
```

1.7 Simulation Framework
Q&A (1/2)

1. **What is one main benefit of using simulation in robotic production systems?**
   A. It eliminates the need for real robots
   B. It guarantees optimal layout design
   C. It enables virtual testing of layouts and sequences
   D. It increases hardware cost
2. **How does simulation help improve production scheduling?**
   A. By manually recording each movement
   B. By introducing delays
   C. By generating real-time animations
   D. By modeling system behavior to anticipate bottlenecks
3. **Why is simulation preferred over manual trials?**
   A. Manual trials are more precise
   B. Simulation is slower but more accurate
   C. Simulation is faster, cheaper, and less disruptive
   D. Simulation avoids software use

1.7 Simulation Framework Q&A (2/2)

- Answers:
  - 1. C
  - 2. D
  - 3. C

CONSORTIUM

meet the team

MOSAPTO

ATC
ATHENS TECHNOLOGY CENTER

Fraunhofer
IOSB

UNIVERSITÉ
DE LORRAINE

UNIVERSITY OF PIRAEUS
RESEARCH CENTER

THE E-BUSINESS RESEARCH CENTER
ELTRUN
ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS

EKS
InTec GmbH

BOC
www.boc-group.com

AEGIS
IT RESEARCH

LIC
LIETUVOS
INOVACIJŲ
CENTRAS

FFT

SEW
USOCOME

CRF

ILTAR
ITALBOX

Funded by
the European Union

CONTACT us

MODAPTO

modapto.eu

info@modapto.eu

MODAPTO Horizon Europe Project

@MODAPTO_eu

MODAPTO Horizon Europe Project

Funded by
the European Union