Karlsruhe Institute of Technology

Institute for Material Handling and Logistics
Prof. Dr.-Ing. Kai Furmans

Seminar Paper

Implementation of “Algorithms for on-line order batching in an order picking warehouse” using Python

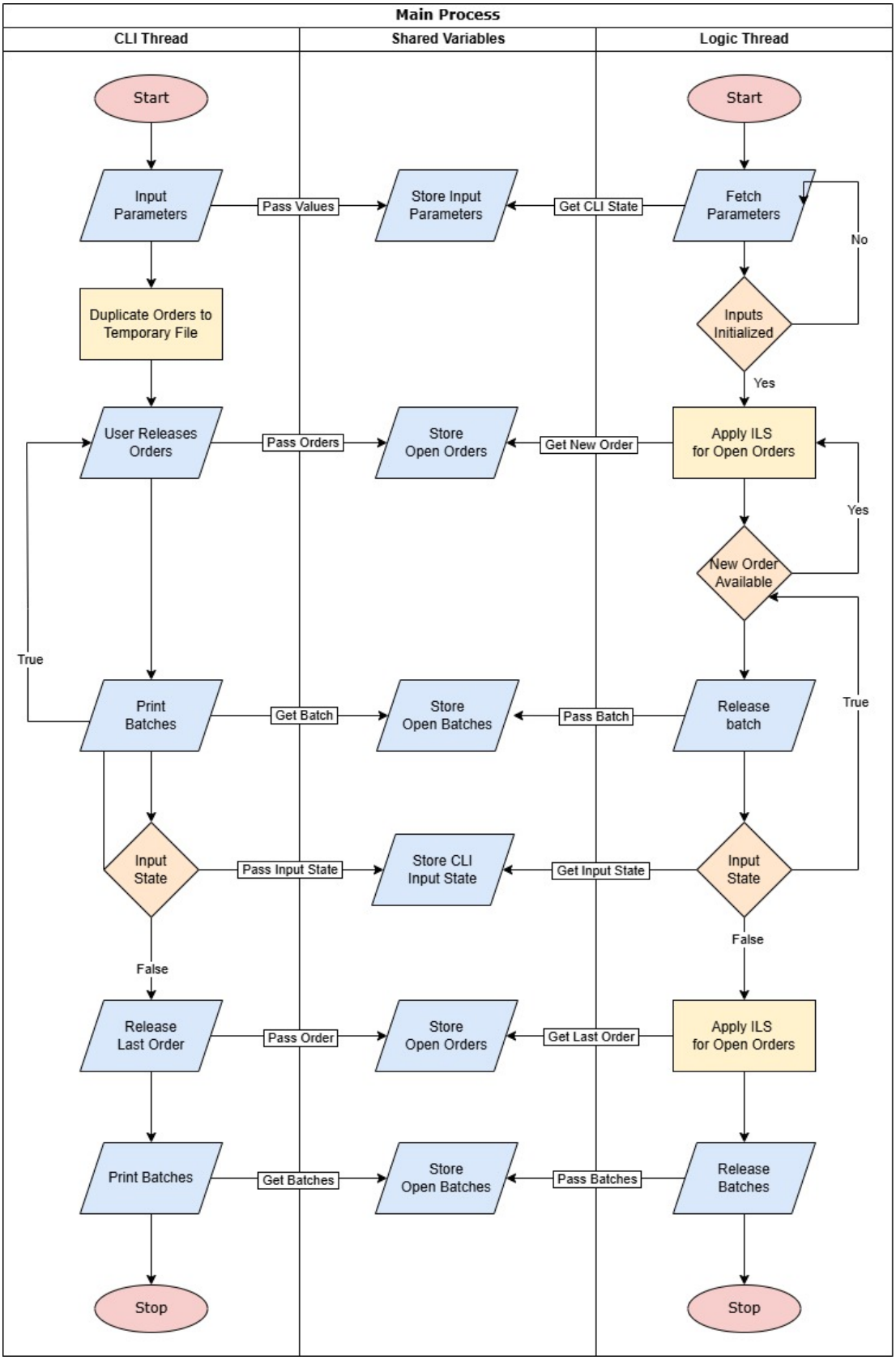
Submitted by:
Philipp Schmidt

Karlsruhe, July 24

Supervised by:
M.Sc. Maximilian Barlang

Table of Contents

Table of Figures



1 Introduction

In modern warehouse management, the efficiency of order picking is crucial for operational success. One of the significant challenges in this area is the order batching problem, which involves grouping customer orders into batches for efficient picking. Effective order batching can significantly reduce the total travel time of pickers, thereby increasing productivity and reducing operational costs.

The complexity of the order batching problem is further amplified in dynamic environments where orders arrive continuously. This scenario, known as the On-Line Order Batching Problem (OOBP), requires real-time decision-making to continuously optimize the batching process as new orders come in. Given the combinatorial nature of this problem, it is classified as $NP-Hard$, meaning that finding an exact solution in a reasonable timeframe is computationally infeasible. Consequently, heuristic methods are often employed to find practical, near-optimal solutions.

Sebastian Henn has made significant contributions to addressing the OOBP through the development of heuristic algorithms, particularly the Iterated Local Search (ILS) algorithm. Henn's algorithm iteratively refines solutions by exploring neighboring solutions and applying perturbations to escape local optima, thus improving the overall solution quality. The application of such algorithms in real-world scenarios requires robust and flexible software implementations.

1.1st Research Objectives

The primary objective of this work is to implement the algorithm developed by Sebastian Henn[1] for solving On-Line Order Batching Problems using Python. This involves:

- **Algorithm Implementation**: Developing a robust Python implementation of Henn's algorithm, ensuring it accurately follows the methodology outlined in the foundational research.
- **Real-World Applicability**: Designing the program to be deployable in real-world warehouse scenarios, including handling dynamic order arrivals and providing efficient batching solutions.
- **Scalability**: Making the program scalable to handle large datasets and high-frequency order arrivals, suitable for various warehouse environments.
- **Testing and Validation**: Conducting extensive testing to validate the accuracy and efficiency of the algorithm in different scenarios, ensuring it meets the required performance standards.
- **Future Integration**: Laying the groundwork for future integration into larger systems, including database connections, API development, and potential incorporation into microservice architectures.

1 Henn, “Algorithms for On-Line Order Batching in an Order Picking Warehouse.”

- By achieving these objectives, this work aims to demonstrate the practical application of Henn's algorithm in dynamic warehouse environments, providing a reliable and efficient solution for on-line order batching problems.

1.2nd Research Design

This paper outlines the general process of developing and implementing the algorithm for solving On-Line Order Batching Problems. The research design involves the following steps:

- **Algorithm Understanding**: Gaining a deep understanding of Henn's algorithm, including its theoretical foundation, the Iterated Local Search (ILS) algorithm, and the specific heuristics used for batch optimization and release timing.
- **Implementation Planning**: Designing the implementation plan, including the overall architecture of the program, the separation of logic into different threads, and the structure for data sharing and communication between threads.
- **Python Implementation**: Writing the code in Python to implement Henn's algorithm. This includes developing the core logic, setting up the Command Line Interface (CLI), and ensuring that all necessary parameters and processes are accurately represented in the code.
- **In-Depth Documentation:** Providing thorough in-line comments and documentation within the code to explain each step of the implementation. This ensures clarity and ease of understanding for future developers and researchers who may work with or build upon this implementation.
- **Testing and Validation**: Conducting extensive testing to ensure the accuracy and efficiency of the implementation. This includes testing with various datasets and scenarios to validate the algorithm's performance and robustness.
- **Optimization**: Identifying and implementing optimizations to improve the performance of the algorithm, such as reducing unnecessary data manipulations and optimizing communication mechanisms.
- **Real-World Applicability**: Ensuring the program is designed to be deployable in real-world warehouse scenarios, capable of handling dynamic order arrivals, and providing efficient batching solutions.

The actual implementation, along with in-depth, in-line comments explaining the code, can be found on the project's GitHub repository[2]. This repository provides a detailed view of the codebase and the specific methods used to achieve the research objectives.

2 Schmidt, “PhilippXXY/on-Line-Order-Batching.”

2 Foundation and Literature Review

In the foundational paper by Henn[3], the author explores various approaches to address the on-line order batching problem within an order picking system characterized by a parallel-aisle warehouse and a single picker. In this context, batching refers to the process of assigning incoming orders to distinct batches, ensuring that no single order is split across multiple batches.

A key distinction of Henn's work compared to existing methodologies is the dynamic nature of order arrivals during runtime. This dynamic arrival of orders implies that each new order could potentially lead to the formation of a new optimal batch. However, the combinatorial possibilities increase exponentially with the number of orders, rendering the problem $NP-Hard$. Consequently, heuristic methods must be employed to find practical solutions within a reasonable timeframe.

To tackle this challenge, Henn et al.[4] propose a modified Iterated Local Search (ILS) algorithm to generate satisfactory batching solutions. The ILS algorithm iteratively refines an initial solution by exploring neighboring solutions and applying perturbations to escape local optima, thus enhancing the overall solution quality. Additionally, the authors introduce a custom heuristic to determine the optimal timing for batch release, further improving the efficiency of the order picking process.

By leveraging these heuristic approaches, the research aims to provide effective solutions to the on-line order batching problem, ensuring timely and efficient order fulfillment in dynamic warehouse environments.

3 Henn, "Algorithms for On-Line Order Batching in an Order Picking Warehouse."
4 Henn et al., "Metaheuristics for the Order Batching Problem in Manual Order Picking Systems."

3 Algorithm Development

The implementation of the algorithm proposed by Henn was carried out in Python to facilitate its integration into the existing 4D4L project at the Institute of Material Handling and Logistics.

3.1st Design

The project is designed to mimic a real-life production scenario by separating the process into different threads. One thread, referred to as the `LogicThread`, contains the actual computing logic and is responsible for generating batches based on the provided data. The other thread, known as the `CLIThread`, is primarily for demonstration purposes.

In the absence of well-defined APIs, the simplest way to exchange information while maintaining a modular structure is through file-based communication. Both threads access a common file to read and write shared variables. This file is crucial for the current implementation, as the `LogicThread` requires the initial parameters provided by the user. The flowchart, as seen in Figure 1 Flowchart Program, illustrates the process.

It should be noted that this method of data transfer can lead to issues when data is accessed too frequently, resulting in information loss due to write and read conflicts. Therefore, it is highly recommended to run the program in a realistic manner and to implement a more robust communication method when integrating it into other projects.

3.2nd Assumptions

Several assumptions were made to facilitate the project's development:

1. Only one picker operates in the warehouse.

2. The warehouse is assumed to be a parallel-aisle warehouse with the starting and endpoint located at the aisle coordinates $(x, y)=(0,-1)$, as shown in Figure 2 Sample Warehouse Layout.

3. The height and depth of products on the shelves are not accounted for when calculating the tour length.

4. All products are assumed to require the same amount of space when selected, meaning that the maximum batch size is determined solely by the number of products in it.

5. No additional time is calculated for picking the product itself. Only the tour length through the warehouse is considered.

[5]

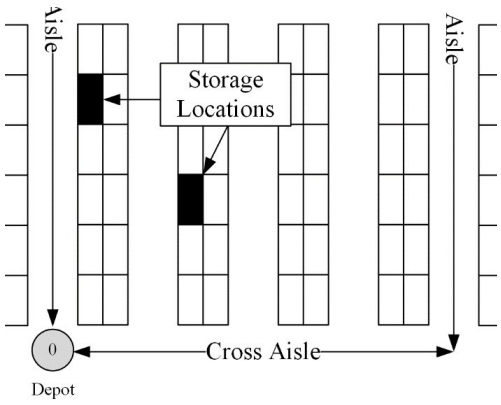


Figure 2 Sample Warehouse Layout

3.3rd Implementation

The program and its threads are initially managed by `main.py`, which sets up the shared variables through `shared_variables.py`. This file acts as a medium for data sharing between threads.

3.3rd1st Core Logic

The core logic of the implementation is the most critical component and will be discussed in detail. Initially, the program receives several essential parameters:

- **Warehouse Layout Path:** The path to a CSV file where each item ID is assigned to coordinates. These coordinates are used to calculate the tour length.
- **Maximum Batch Size:** The maximum number of items that can be included in a single batch.
- **Tour Length Units per Second:** Since the speed of the picker through the warehouse is unknown, a conversion rate must be determined.
- **Rearrangement Parameter:** This parameter is crucial for the perturbation phase of the Iterated Local Search (ILS) algorithm, which helps in escaping local optima and finding better solutions.
- **Release Parameter:** Used to determine the delayed release of a batch when only one batch is available, ensuring that the system waits for potentially better combinations.
- **Threshold Parameter**: Necessary for the ILS algorithm to decide when to terminate if subsequent iterations do not yield significant improvements, thus saving computational resources.
- **Time Limit**: A constraint for the calculations due to the $NP-Hard$ nature of the problem, ensuring that the algorithm completes in a practical timeframe.

[5] "Parallel-Aisle-Warehouse-and-Its-Graph-Representation.Ppm (722×1148)."

- **Selection Rule**: Defines how the batches are sorted after the last order is submitted.

Once these parameters are set, the program enters its runtime state, where new orders can be continuously added to the system. When a new order arrives, all open orders are considered, and the Iterated Local Search Algorithm is applied to generate more optimized batches.

To compare the tour lengths of different solutions, the program implements the S-Shape Routing[6] method. In this method, every aisle containing a needed item is indexed. Odd-indexed aisles are traversed in ascending order, and even-indexed aisles are traversed in descending order until the last item is reached. From there the shortest path back to the starting point is taken. The necessary information is taken from the given CSV file, which is processed using the Pandas[7] library to assign locations in the warehouse to each item in the batch.

If multiple batches are available and the last order has not been submitted, batches are released immediately once the arrival time of the previously released batch has passed, indicating that the picker has returned. If only one batch is available, a delayed start time is calculated as described by Henn to optimize the inclusion of new orders. After receiving the last order, the program creates newly optimized batches, sorts them according to the selection rule, and releases them sequentially.

When the arrival time of the last batch is determined, the `LogicThread` stops, ensuring no unnecessary computations are performed beyond the required operations.

For more detailed explanations, it is advisable to look at the underlying paper by Henn.

33rd1st1st Delayed Release Time

When only one batch remains and there is a possibility of a new incoming order, the release of the batch will be delayed to allow for potential optimizations. The formula, as described by Henn, is adjusted to convert the delay into predefined tour length units per second.

3.3rd2nd Command Line Interface

As mentioned in Section 3.1, the `CLIThread` is a provisional solution until the program is fully integrated. The CLIThread serves as an interface for the user to interact with the program, providing a means to input parameters, monitor progress, and view results.

In this implementation, the CLIThread handles the following functionalities:

- **User Input**: It allows users to input necessary parameters such as the warehouse layout path, maximum batch size, rearrangement parameter, release parameter,

6 Hong and Kim, “A Route-Selecting Order Batching Model with the S-Shape Routes in a Parallel-Aisle Order Picking System.”
7 “Pandas.”

threshold parameter, time limit, and selection rule. This is facilitated through a command-line interface that prompts the user for each input sequentially.

- **Data Sharing**: The `CLIThread` writes these inputs to a shared file (`shared_variables.py`), which the `LogicThread` reads to perform computations. This file-based communication ensures that both threads have synchronized access to the necessary parameters without requiring a more complex inter-process communication mechanism.
- **Order Management**: The `CLIThread` imports a given JSON file containing the orders into a Python data structure. Each time the user inputs a command to release an order, it is popped from the Python data structure and added to the shared variables. This enables the core logic in the `LogicThread` to interact with and process the order.
- **Progress Monitoring**: The `CLIThread` continuously monitors the status of the `LogicThread`, providing real-time updates to the user. This includes displaying the currently releases batch, ready to be picked.
- **User-Friendly Experience**: To enhance the user experience, several libraries[8][9][10][11] have been used for creating aesthetically pleasing and informative command-line outputs. These libraries help in formatting the output, making it easier for users to read and interpret the information.

The provisional nature of the CLIThread means it is designed to be replaced by a more sophisticated interface in the future, likely integrated within the 4D4L[12] project's existing systems. However, the current implementation ensures that the core functionalities are accessible and operable, providing a solid foundation for further development and integration.

8 “Click.”
9 “Inquirerpy.”
10 “Keyboard.”
11 “Tabulate.”
12 Wang (inaktiv), “KIT - IFL Forschung - Aktuelle Forschungsprojekte - 4D4L – Daten- und zielgetriebene sequentielle Entscheidungsfindung für zeitdynamische Logistiksysteme.”

4 Conclusion

The goals of the seminar, which were to provide a working program to solve On-Line Order Batching Problems using the algorithm developed by Henn, were not only fulfilled but even exceeded. The successful implementation of a Command Line Interface (CLI) means that the program could theoretically be deployed in a simplified warehouse scenario today.

The development process involved extensive testing, which ensured the robustness and reliability of the program. However, this extensive testing also resulted in a higher investment of time and effort than initially anticipated. Despite these challenges, the project has successfully demonstrated the practical applicability of Henn's algorithm in a dynamic warehouse environment.

By splitting the code into the actual core logic and a command line interface and sharing data via a shared file, the program is easily adaptable for implementation in other projects.

4.1st Outlook

For future integration and improvements, several enhancements are recommended to make the program more robust and flexible:

- **Enhanced Input Validation**: Currently, the program performs only basic checks for faulty inputs. Implementing more advanced validation mechanisms would improve the program's reliability and prevent errors due to incorrect data entry.
- **Database Integration**: Connecting the program to a database would significantly enhance its capabilities. A database connection would allow for better data management, storage, and retrieval, making the system more scalable and efficient.
- **API Development**: To use the program in a larger environment, it is necessary to develop a well-defined API. This API would facilitate interaction with other parts of a warehouse management system, potentially as part of a microservice architecture. This would ensure that the program can be seamlessly integrated into existing systems and can communicate effectively with other services.
- **Scalability and Performance Optimization**: As the program is intended for real-world use, further work on optimizing its performance for large datasets and high-frequency order arrivals would be beneficial. This includes improving the efficiency of the Iterated Local Search algorithm by removing unnecessary data manipulation operations, such as deep copies, and optimizing the file-based communication mechanism.

By addressing these areas, the program can be made more robust, flexible, and ready for deployment in a variety of warehouse environments. The successful integration of these enhancements would ensure that the program not only meets current needs but is also scalable for future demands.

Bibliography

"Click: Composable Command Line Interface Toolkit." OS Independent, Python. Accessed July 5, 2024. https://palletsprojects.com/p/click/.

Henn, Sebastian. "Algorithms for On-Line Order Batching in an Order Picking Warehouse." *Computers & Operations Research* 39, no. 11 (November 1, 2012): 2549–63. https://doi.org/10.1016/j.cor.2011.12.019.

Henn, Sebastian, Sören Koch, Karl F. Doerner, Christine Strauss, and Gerhard Wäscher. "Metaheuristics for the Order Batching Problem in Manual Order Picking Systems." *Business Research* 3, no. 1 (May 1, 2010): 82–105. https://doi.org/10.1007/BF03342717.

Hong, Soondo, and Youngjoo Kim. "A Route-Selecting Order Batching Model with the S-Shape Routes in a Parallel-Aisle Order Picking System." *European Journal of Operational Research* 257, no. 1 (February 16, 2017): 185–96. https://doi.org/10.1016/j.ejor.2016.07.017.

"Inquirerpy: Python Port of Inquirer.Js (A Collection of Common Interactive Command-Line User Interfaces)." Microsoft, Unix, Python. Accessed July 5, 2024. https://github.com/kazhala/InquirerPy.

"Keyboard: Hook and Simulate Keyboard Events on Windows and Linux." MacOS :: MacOS X, Microsoft :: Windows, Unix, Python. Accessed July 5, 2024. https://github.com/boppreh/keyboard.

"Pandas: Powerful Data Structures for Data Analysis, Time Series, and Statistics." OS Independent, Cython, Python. Accessed July 5, 2024. https://pandas.pydata.org.

"Parallel-Aisle-Warehouse-and-Its-Graph-Representation.Ppm (722×1148)." Accessed July 5, 2024. https://www.researchgate.net/publication/325554802/figure/fig1/AS:776776968060928@1562209469949/Parallel-aisle-warehouse-and-its-graph-representation.ppm.

Schmidt, Philipp. "PhilippXXY/on-Line-Order-Batching," June 19, 2024. https://github.com/PhilippXXY/on-line-order-batching.

"Tabulate: Pretty-Print Tabular Data." OS Independent, Python. Accessed July 5, 2024. https://github.com/astanin/python-tabulate.

Wang (inaktiv), Peiqi. "KIT - IFL Forschung - Aktuelle Forschungsprojekte - 4D4L – Daten- und zielgetriebene sequentielle Entscheidungsfindung für zeitdynamische Logistiksysteme." Text. Peiqi Wang (inaktiv), October 5, 2023. KIT. https://www.ifl.kit.edu/forschungsprojekte_5762.php.