2023 IEEE Symposium Series on Computational Intelligence (SSCI)
Mexico City, Mexico. December 5-8, 2023

On the Feasibility of Using a High-level Solver within Robotic Mobile Fulfillment Systems*

Maria Torcoroma Benavides-Robles, Jorge M. Cruz-Duarte,
José C. Ortiz-Bayliss, and Ivan Amaya
Tecnologico de Monterrey, Monterrey 64700, Mexico,
E-mails: {A00836554, jorge.cruz, jcobayliss, iamaya2}@tec.mx

***Abstract*—A Robotic Mobile Fulfillment System (RMFS) is a collaborative environment in which a robot delivers products to human for fulfilling orders. However, it is a computationally complex optimization problem. In this work, we analyze the feasibility of using high-level solvers for selecting suitable low-level methods. To this end, we generate 111 instances distributed into two datasets. Moreover, we implement two kinds of high-level solvers. The first one is a set of handcrafted rules. The second approach uses a decision tree. Our data reveals that it is possible to construct high-level solvers that benefit from the different strengths of the low-level methods by selecting which one to apply. The rules produced by hand and the decision trees high-level solvers are competitive concerning the best individual performer in terms of two standard metrics for this problem: throughput time and orders completed.**

***Index Terms*—RMFS, Robotic Mobile Fulfillment System, Warehouse, Kiva system, Mobile robots, E-commerce, RAWSim-O simulator.**

I. Introduction

RECENT years have been challenging for all disciplines. Among such challenges, retailers experienced a sudden rise in their online demand because of the pandemic. For some companies, this meant developing and integrating a new business scheme. For others, it implied a higher stress in their supply chain. Nowadays, people have kept on using online ordering systems and so automated warehouses must be improved.

In the not-so-distant past, products within warehouses were handled manually. Thanks to technology breakthroughs, especially in robotics, many systems integrate robots and human labor into a collaborative environment nowadays [1], [2]. The first automated warehouse dates from 1960 [3], but it was not until 2008 that the interest grew, starting with Wurman *et al.*'s proposal for automated warehouses [4]. They used an automated guided vehicle to transport products to and from their storage locations, and into a packing station. In doing so, they improved the performance of the warehouse. They also coined the terms 'pods' and 'workstation' for the storage locations and the packing station. Plus, although their approach was initially referred to as the *Kiva system*, it was later generalized into the name used in the present, *i.e.*, Robotic Mobile Fulfillment Systems (RMFS). There are other diverse kinds of automated warehouses [3], but we limit ourselves to RMFS for the sake of brevity.

Research on RMFS is sparse. From the scarce literature, one can find that some authors have focused on the system's sensitivity. For example, Feng *et al.*analyzed the location influence of workstations [5], Valle *et al.*studied the pod allocation [6], and Ma *et al.*examined the distribution of Stock Keeping Units (SKUs) [7]. Besides that, there have been a few attempts to incorporate Artificial Intelligence into RMFS. For example, Yang used a clustering model for grouping orders and items based on similarity metrics [8]. Conversely, Douchan and Kaminka developed a model based on Reinforcement Learning that learns when to use different collision-avoidance approaches [9]. Although such a work incorporates a high-level approach for selecting lower-level techniques, it does so for the robot movement component. Hence, it is easy to identify a knowledge gap regarding the feasibility of using high-level approaches to improve the pod storage component of the RMFS. This work addresses to fill such a gap.

II. Theoretical Foundations

We now present some foundations of our work.

A. Robotic Mobile Fulfillment Systems (RMFSs)

An RMFS integrates diverse variables within a warehouse, such as the number of robots, pods, products, and humans. These variables relate to different optimization problems, *e.g.*, the planning of routes for the robots, the scheduling of orders, and the distribution of products within pods. Fig. 1 shows an illustrative RMFS layout proposed by Wurman *et al.* in [4]. Hence, the RMFS corresponds to a complex optimization problem, where that complexity stems from more than just the inherent optimization problems. Instead, it emerges due to the interaction of those problems and the many details that must be simultaneously considered in real life. For example, in real life, a robot and a pod can exist at the exact location (the pod sits on top of the robot), but two robots (or two pods) cannot. In the case of a large order, it may take quite some time to fulfill it. Hence, processing an order at multiple workstations may be desirable in real-world applications. Nevertheless, this makes the modeling more complex and incurs the need for an additional station that integrates all parts of the order. These elements must be clearly defined since they interact when simulating the warehouse.

*Maria Benavides thanks the Consejo Nacional de Humanidades, Ciencia y Tecnología (CONAHCyT) for the financial support provided through her fellowship 866896. The authors also thank CONAHCyT for the financial support given through grant 287479.

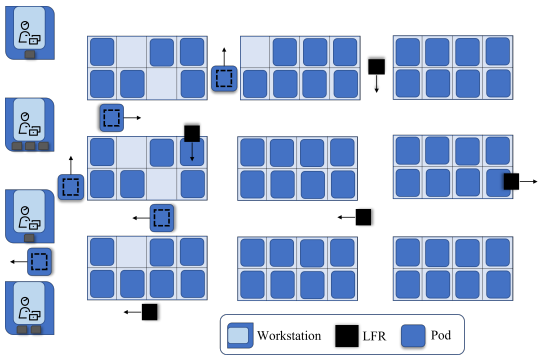


Fig. 1: Illustrative layout of a warehouse based on [4].

With the release of the Kiva system in [4], interest in these proposals was sparked. However, because of the problem's complexity, isolating some components is usually necessary. For instance, some authors have analyzed the pod allocation component of the RMFS, albeit under different conditions. An early approach was presented by Xie *et al*., who developed a Genetic Programming-based Hyper-Heuristic strategy [10]. Later, Yuan *et al*. worked on the Multi-Robot Task Allocation for e-commerce applications [11]. Further, Zhuang *et al*. implemented a Mixed Integer Programming model that accounts for workload balancing and pod conflicts [12].

Some companies have already attempted to implement an RMFS. For example, Amazon has leveraged it to improve worker safety, and they have opted for different kinds of robots for handling different kinds of products [13]. In contrast, FedEx has tested the system for sorting letters and small packages [14]. Plus, Walmart has considered it to improve product resupply [15]. In any case, alternatives are diverse, so an adequate approach to tackling the RMFS is required.

B. Some previous attempts at solving RMFSs

Many researchers have considered diverse methods for tackling the problem components related to RMFSs. Most of these methods are hybrid approaches since it is necessary to consider different subproblems simultaneously. For example, Yuan *et al*.implemented a heuristic method based on Greedy and Simulated Annealing algorithms [16]; Zou *et al*.reported a multi-component technique merging Variable Neighborhood Search, semi-open queuing networks, and a two-phase approximate approach [17]; Lee *et al*. presented an A* algorithm based on a cyber-physical system [18]; and Sun *et al*. deployed an A* approach for routing combined with a Simulated Annealing algorithm for scheduling [19].

Similarly, other authors have focused on simulating different aspects of the RMFS. A relevant example is that from Lienert *et al*., who simulated changes in the storage layout and found a way to increase warehouse throughput [20]. Another work worth mentioning is that from Wu *et al*., who focused on the structural parameter configuration of the warehouse [21].

Interestingly, many experts have also implemented Heuristics and Metaheuristics for tackling the RMFS. Some examples worthy of mentioning include the Integrated Bi-level Memetic algorithm [22], an auction-based heuristic method [23], and a Heuristic Beam Search algorithm [24]. Other implementations involve the use of Genetic Algorithms [10], [25], Mathematical Programming techniques [26], Machine Learning algorithms [8], [27], and Monte Carlo sampling [28].

C. Available Frameworks for RMFS

The literature contains different approaches for simulating the warehouse within an RMFS. Among them, one finds two open-source alternatives. The first one is Alphabet Soup, developed by Hazard *et al*. [29] but no longer supported. Nonetheless, this framework was employed by Merschformann *et al*. as a base for the second alternative: the RAWSim-O framework [30]. RAWSim-O includes many new parameters, striving to diminish the gap between simulation and reality. Among them, one can find different solvers for different subproblems; for example, for allocating pods, it incorporates six alternatives:

`Random`: Place pods in random positions.
`Fixed`: Store pods at locked (fixed) positions.
`Nearest`: Allocate pods based on distance information.
`Station Based`: Store pods close to workstations.
`Cache`: Keep pods, will be used in the near future, close to workstations.
`Turnover`: Assign positions to returning pods using item-frequency information.

It is essential to mention that different authors have used these two frameworks. Douchan *et al*. utilized Alphabet Soup in their work for evaluating the effectiveness index intrinsic reward [9]. Likewise, Merschformann *et al*. used RAWSim-O to analyze the RMFS's path planning component by generating ten different instances [31]. In more recent work, Xie *et al*. developed 27 instances to analyze the effect of splitting an order across some workstations [32]. As the reader may notice, the size of the datasets used in previous works is quite conservative.

III. Methodology

Throughout our experiments, we used the RAWSim-O framework [30] to emulate the warehouse and all the inner processes that lead to order completion. Bear in mind that this framework continuously generates orders for the simulation, and the termination criterion was, thus, set to a fixed simulation time of five hours. As mentioned, the RMFS involves many variables easily defined within the framework, such as the layout distribution, the number of robots, the total SKUs, and the techniques used to solve each subproblem. Moreover, the framework internally generates all the products and orders with this information. For the scope of this work, we strove to modify as few variables as possible, seeking to generate different kinds of instances. Hence, we concentrated on the pod allocation component of the RMFS. With this information, we could monitor the effect of selecting different techniques to solve such a component.

A. Problem Instances

We generated 27 instance types with different parameters based on a two-fold approach. So, instances can be sorted into two big groups, as Table I shows. Set I contains instances with some abrupt parameter variations, focusing on the number of replenishment stations, workstations, robots, pods, and SKUs. Our goal with this set was to generate instances with different patterns, motivated by the idea that different solvers perform differently. Here, we also want to analyzed the repeatability of the solvers. Thus, we generated 15 instances of each type using different seed values. So, we can find conflicting scenarios for the same instance type; for example, within the same type, cases where solver A performs great on one instance but poorly on another.

TABLE I: Features of the instance types in Set I.

Type	Replenishment	Workstation	Bots	Pods	SKUs
1	4	4	6	428	1000
2	4	4	75	428	1000
3	8	8	32	809	1000
4	4	4	32	428	10000
5	8	8	32	809	10000
6	4	4	32	428	100

The second set of instances, Set II, was simpler in nature and only modified the number of replenishment stations, workstations, and pods. Table II displays the features of the instance types for this set. Our goal with Set II was to study how the nature of instances changes with smoother transitions. Hence, we selected combinations of big, medium, and small values, which may provide more detailed patterns. Also, we only generated a single instance of each type to keep things as simple as possible.

TABLE II: Features of the instance types in Set II.

Type	Replenishment	Workstation	Bots	Pods	SKUs
7	32	1	1	3094	1000
8	32	1	32	3094	1000
9	32	32	1	3094	1000
10	32	32	32	3094	1000
11	1	1	1	3094	1000
12	1	1	32	3094	1000
13	1	32	1	3094	1000
14	1	32	32	3094	1000
15	12	6	6	1190	1000
16	12	6	12	1190	1000
17	12	12	6	1190	1000
18	12	12	12	1190	1000
19	6	6	6	1190	1000
20	6	6	12	1190	1000
21	6	12	6	1190	1000
22	6	12	12	1190	1000
23	1	1	6	142	1000
24	1	1	12	1190	1000
25	1	6	1	618	1000
26	1	6	6	618	1000
27	1	6	12	618	1000

In summary, we analyzed 111 instances: 15 of each of the first six types and one of the remaining 21 types. Although this value may seem small, we remind the reader that previous works have considered 10 [31] and 27 [32] instances, respectively. Hence, we are confident that our proposed number of instances is enough to elucidate some patterns. Finally, for all experiments, we utilized the same layout distribution for the warehouse as shown in Fig. 2. The reason is that the location of workstations and replenishment stations affect performance [33]. In our warehouse, we arranged the workstations on the right and the replenishment stations on the left, following the scheme proposed by Merschformann *et al.* [30]. Plus, we considered two metrics for analyses: throughput time and orders completed. The first indicates how long an order takes to exit the system. The second one represents the number of orders completed within the five-hour simulation window.

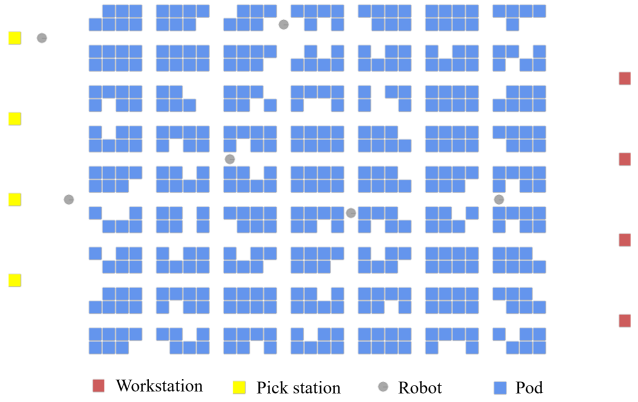


Fig. 2: Layout distribution generated by the RAWSim-O framework [30].

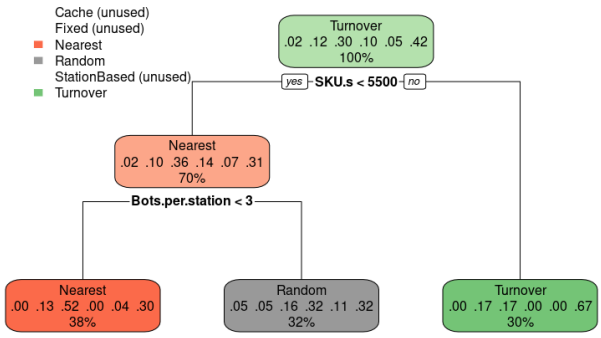
B. Effect of Using High-level Solvers

For this work, we produce three high-level solvers for the problem. The first one, SR, is based on an empirical analysis of the behavior of the low-level methods. This handcrafted approach uses a simple yet useful rule to select the low-level method to apply in each situation:

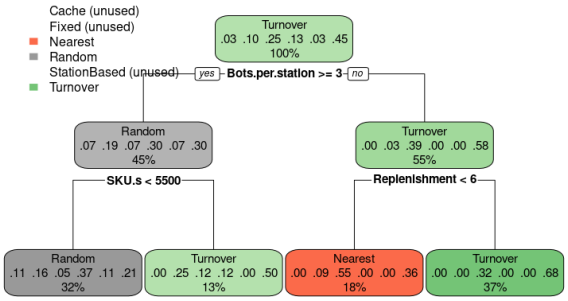
```
IF Type == 1 OR Type == 2 use Station Based
IF Type == 3 use Nearest
ELSE use Turnover
```

The remaining two solvers are motivated by reducing the need for human intervention in the design process. To do so, we relied on Decision Trees (DTs). Such trees identify the features that best characterize the instances and allow us to map an instance to a suitable solving strategy. We train such trees on 66% of the instances within Set I and test them on the remaining 33% of the instances from such a set. Then, we generate a DT for each one of the performance metrics (*i.e.*, throughput time and orders completed) using the package `rpart` in `R`. Although we know that hyper-parameter tuning may improve the performance of these classifiers, we opt for the default parameter setup for simplicity. Let us call these methods ST (Fig. 3a) and SO (Fig. 3b).

At this point, it is worth noticing that all high-level solvers incorporate Nearest and Turnover. However, our experience



(a) ST (throughput time)



(b) SO (orders completed)

Fig. 3: Internal structure of high-level solvers generated with decision trees for different metrics.

suggests that Station Based could be useful in some cases, as we consider in SR. Nonetheless, ST and SO opt for using Random instead.

IV. Experimental Results

Along the following lines, we analyze the results derived from running the methods described in Sect. II-C on the instances considered for this work. As we mentioned in Sect III, we also consider two approaches for the high-level solvers: a handcrafted rule and data-driven decision trees. Moreover, since we analyze two performance metrics, we divide our analysis into two folds, one per metric.

A. Problem instances

Let us begin by analyzing the generated instances and the performance of baseline solvers. Tables III and IV present all performance metrics. Both tables exhibit a similar behavior, where only a few heuristics have the best performance, and it is easy to see that the Random solver never wins, at least in this set. Moreover, Cache is the best alternative only for a single instance. Notwithstanding, it is also evident that there is no such thing as the 'perfect' solver that works best across all instances. This strengthens the idea that combining solvers is a good approach. Such an effect can also be deduced from the last column. Here, we provide information about a synthetic oracle, for comparison purposes. However, such a solver is unfeasible in practice as it requires solving each instance with all solvers. Nonetheless, this synthetic solver serves as a reference value. In fact, this approach allows for an average throughput time 11.31 time units smaller than that of the best solver. Similarly, it allows for an average of 91.33 more orders. Bear in mind that such values disregard the cases where a solver fails to finish in the allotted computation time.

TABLE III: Performance of baseline solvers on the generated instances when considering the throughput time metric.

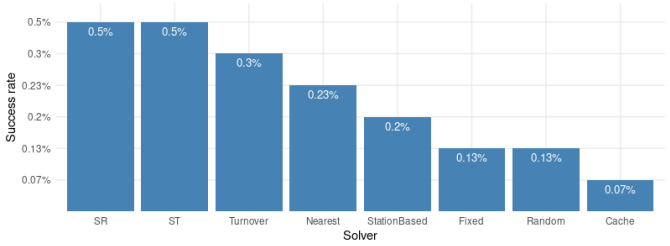
Type	Random	Fixed	Nearest	Station Based	Cache	Turnover	Oracle
1	483.83	489.17	444.34	444.08	465.68	467.00	444.08
2	185.49	184.91	185.81	184.63	186.64	186.71	184.63
3	194.16	194.73	184.99	197.36	195.85	186.78	184.99
4	188.22	188.22	187.16	187.01	188.41	185.28	185.28
5	343.41	316.69	304.42	329.58	314.67	299.86	299.86
6	190.90	189.77	190.96	191.74	190.71	188.98	188.98
7	560.84	496.10	461.24	461.24	548.74	581.82	461.24
8	184.16	185.34	183.32	185.10	185.75	186.66	183.32
9	693.97	718.54	690.36	680.25	Inf	546.91	546.91
10	Inf	Inf	Inf	673.89	Inf	Inf	673.89
11	574.03	542.15	502.09	502.09	532.65	560.82	502.09
12	183.40	183.21	186.48	184.10	184.76	181.40	181.40
13	740.75	672.86	581.95	784.77	Inf	694.81	581.95
14	644.38	603.08	Inf	631.67	Inf	Inf	603.08
15	597.86	554.92	562.81	585.48	545.12	550.75	545.12
16	418.52	385.36	379.63	404.30	423.48	379.19	379.19
17	615.57	558.51	583.39	580.12	616.87	540.78	540.78
18	595.76	571.75	543.74	602.84	578.32	557.22	543.74
19	602.07	552.70	504.32	534.60	605.69	534.93	504.32
20	418.25	396.25	377.76	410.65	400.61	384.52	377.76
21	648.03	591.13	544.99	579.78	578.26	543.04	543.04
22	610.97	571.67	560.50	612.72	579.73	536.70	536.70
23	187.16	186.94	186.20	185.81	188.09	187.41	185.81
24	183.42	184.98	183.88	185.43	184.73	182.74	182.74
25	627.92	599.62	544.78	538.05	611.17	578.57	538.05
26	547.86	548.91	522.16	552.81	541.92	492.49	492.49
27	396.92	403.62	359.85	397.97	406.83	386.50	359.85

TABLE IV: Performance of baseline solvers on the generated instances when considering the orders completed metric.

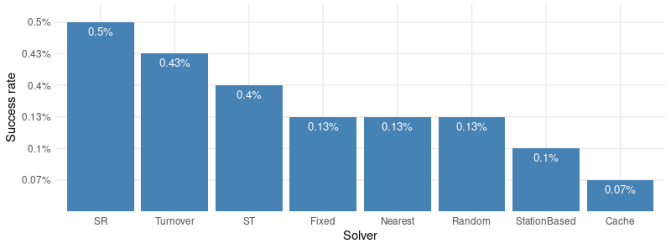
Type	Random	Fixed	Nearest	Station Based	Cache	Turnover	Oracle
1	1493	1448	1599	1534	1497	1519	1599
2	4322	4339	4327	4347	4301	4306	4347
3	7934	8015	8214	7951	7981	8242	8242
4	4177	4167	4202	4202	4175	4228	4228
5	4319	4400	4781	4449	4514	4817	4817
6	4335	4362	4334	4325	4330	4345	4362
7	290	302	381	381	301	305	381
8	1092	1086	1096	1087	1079	1076	1096
9	224	229	238	214	0	261	261
10	0	0	0	4652	0	0	4652
11	279	313	350	350	322	298	350
12	1092	1093	1073	1085	1083	1105	1105
13	201	241	250	196	0	229	250
14	4949	5355	0	5085	0	0	5355
15	1146	1177	1204	1131	1138	1238	1238
16	2431	2591	2746	2548	2407	2608	2746
17	1089	1189	1205	1124	1112	1210	1210
18	2169	2307	2381	2183	2235	2332	2381
19	1104	1195	1180	1222	1088	1275	1275
20	2459	2569	2648	2491	2546	2574	2648
21	1060	1174	1196	1166	1165	1228	1228
22	2170	2181	2342	2210	2224	2422	2422
23	1073	1074	1074	1078	1067	1066	1078
24	1088	1080	1087	1078	1082	1093	1093
25	275	296	313	314	295	299	314
26	1189	1233	1243	1243	1220	1269	1269
27	2676	2621	2776	2611	2598	2669	2776

B. High-level Solvers for Throughput Time

Among our three classifiers, SR is generic, and ST focuses on the throughput time. Hence, we now analyze their performance over the test set. Our data indicates that it is quite complex to properly match instance types and low-level solvers. Even so, both approaches improve upon the baseline solvers. For example, SR obtain the best overall value (7170.43 time units). In contrast, the best standalone low-level solver (Turnover) exhibits a throughput time 21.98 time units higher. In the case of ST, the improvement is smaller, being only 14.53 time units better than Turnover. Notice that the modest performance of ST is due to the low accuracy that its internal classifier exhibits (40% on the test set). Regardless of this phenomenon, we must stress that it helps to improve the overall process and reduce the throughput time. Also, the humble savings SR and ST offer is somewhat expected at this point since the classification process for this task still presents many areas for improvement. Then, as we improve the classification process in the future, we shall also improve the overall performance of the high-level solvers. To emphasize the benefit of using SR and ST, we analyze the percentage of instances where each method obtains the best result (success rate). As observed, both SR and ST obtain the best results in 50% of the instances in the test set (Fig. 4a). Moreover, they outperform the low-level methods. Do note that in case of ties, both solvers are successful. Hence, the total success rate can be larger than one.



(a) Throughput time



(b) Orders completed

Fig. 4: Success rate of the high-level solvers and low-level methods on the test set, considering both performance metrics.

So, What happens when the high-level solvers are tested on unseen instances? Aiming to answer this question, we solved Set II with ST. It is relevant to mention that SR cannot be applied to unseen instances since the rule contains only seen instance types (types 1 to 6). This time, however, the results could be more favorable, as ST failed to improve upon the time yielded by the best low-level solver (Station Based). Again, this is expected since ST never saw these instances during the training and because their parameters differ from those of the first six instance types. Despite this adverse situation, ST managed to rank second while providing solutions with 183.2 more time units than Station Based (the best performer in Set II with 10,273.69 time units).

C. High-level Solvers for Orders Completed

In terms of the second metric, Turnover remains the best low-level solver. Once again, both SR and SO represent the best alternative, each yielding a total of 38 more orders than Turnover on the test set. This is noteworthy since these two solvers exhibit a different internal structure. Recall that SR incorporates Station Based as an available low-level method, and SO prefers Random instead. Thus, using different low-level methods may produce the same exact performance. Moreover, the classifier that powers SO has an accuracy of 50%, which is low. Hence, the performance improvement w.r.t. Turnover may yet be expanded.

In terms of the success rate (Fig. 4b), SO falls slightly behind Turnover. However, it is still competitive when compared to the other low-level solvers.

Migrating to Set II exhibits the generalization limitations of using a hand-made rule since it is impossible to use it on unseen instances. In contrast, the best low-level solver, Station Based, yields 33449 orders completed. This time, SO cannot improve such a mark, although it ranks above Cache, SR, and Turnover. However, it remains far from Station Based, the best performer for this set, with a difference of 8735 orders completed. As with the previous metric, this result is expected since the test is clearly beyond the actual generalization capabilities of the models.

V. Conclusions

Throughout this paper we analyzed the feasibility of using a high-level solver, such as an algorithm portfolio, for improving the performance of a Robotic Mobile Fulfillment System (RMFS). To this end, we generated three different approaches and analyzed two metrics: throughput time and orders completed. One of our approaches (SR) represent a handcrafted set of rules based on domain knowledge about the instance subset. The remaining two (ST and SO) were generated using decision trees. Additionally, we considered two sets of instances: one with 90 instances and another one with 21.

Our data revealed that ST and SO work for both metrics. They both perform properly on the testing subset of Set I. But more interestingly, they remain competitive when tackling Set II, which contains never-before-seen instances of different types. Nonetheless, the performance gains obtained by ST and SO are not outstanding. But despite this, such a behaviour is consistent with the current stage of our research. In the case of SR, and although it performs well on the training set,

it remains unfeasible for datasets with instances of different types. Hence, decision trees seem to be a better option.

When analyzing the baseline solvers we noticed that the Random solver is not a good option for the datasets. Notwithstanding, it can be useful when combined with other approaches, as ST and SO showed. This agrees with the fact that the best low-level solver changes depending on the dataset. For example, in Set I, the best low-level option is Turnover. However, in Set II, it is Station Based and Turnover experiences a decrease in performance. In contrast, ST and SO remain competitive. We deem this as noteworthy since it implies that some generalization capability was learnt from the data.

We are aware that there is a lot of work ahead of us. For starters, the performance gains are modest. This can be improved by tweaking the parameters of the decision trees or by implementing a different algorithm altogether. Similarly, the features that we analyze in this work are too broad. Hence, there is a need for developing relevant features that allow to differentiate among different instances of the same type. The distribution of SKUs within pods may prove a relevant source for such data. Finally, one may also expand upon the low-level solvers and integrate other alternatives as well as more instances.

REFERENCES

[1] A. Joseph, "Impact of robotic automation in e-commerce after pandemic," *COMMERCE & MANAGEMENT*, p. 125.

[2] S. Tyrała, A. Orwat, and Ł. Makowski, "Trends and sales models in e-commerce: Examples of best practices," *Zeszyty Naukowe*, vol. 96, no. 1, pp. 89–105, 2022.

[3] K. Azadeh, R. De Koster, and D. Roy, "Robotized and automated warehouse systems: Review and recent developments," *Transportation Science*, vol. 53, no. 4, pp. 917–945, 2019.

[4] P. R. Wurman, R. D'Andrea, and M. Mountz, "Coordinating hundreds of cooperative, autonomous vehicles in warehouses," in *AI Magazine*, vol. 29, pp. 9–19, 2008.

[5] L. Feng, X. Liu, M. Qi, S. Hua, and Q. Zhou, "Picking Station Location in Traditional and Flying-V Aisle Warehouses for Robotic Mobile Fulfillment System," in *2018 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM)*, pp. 1436–1440, IEEE, dec 2018.

[6] C. A. Valle and J. E. Beasley, "Order allocation, rack allocation and rack sequencing for pickers in a mobile rack environment," *Computers & Operations Research*, vol. 125, p. 105090, jan 2021.

[7] Z. Ma, G. Wu, B. Ji, L. Wang, Q. Luo, and X. Chen, "A Novel Scattered Storage Policy Considering Commodity Classification and Correlation in Robotic Mobile Fulfillment Systems," *IEEE Transactions on Automation Science and Engineering*, pp. 1–14, 2022.

[8] N. Yang, "Evaluation of the Joint Impact of the Storage Assignment and Order Batching in Mobile-Pod Warehouse Systems," *Mathematical Problems in Engineering*, vol. 2022, pp. 1–13, apr 2022.

[9] Y. Douchan and G. A. Kaminka, *The Effectiveness Index Intrinsic Reward for Coordinating Service Robots*, vol. 6. 2018.

[10] J. Xie, Y. Mei, A. T. Ernst, X. Li, and A. Song, "A genetic programming-based hyper-heuristic approach for storage location assignment problem," *Proceedings of the 2014 IEEE Congress on Evolutionary Computation, CEC 2014*, pp. 3000–3007, 2014.

[11] R. Yuan, J. Li, X. Wang, and L. He, "Multirobot Task Allocation in e-Commerce Robotic Mobile Fulfillment Systems," *Mathematical Problems in Engineering*, vol. 2021, 2021.

[12] Y. Zhuang, Y. Zhou, Y. Yuan, X. Hu, and E. Hassini, "Order picking optimization with rack-moving mobile robots and multiple workstations," *European Journal of Operational Research*, vol. 300, no. 2, pp. 527–544, 2021.

[13] A. Staff, "New technologies to improve Amazon employee safety," 2021.

[14] W. Knight, "Robots Won't Close the Warehouse Worker Gap Anytime Soon," 2021.

[15] J. Metzger, "Chain Reaction: We're Partnering with Symbotic to Bring High-Tech Automation to Our Supply Chain," 2021.

[16] R. Yuan, J. Li, W. Wang, J. Dou, and L. Pan, "Storage Assignment Optimization in Robotic Mobile Fulfillment Systems," *Complexity*, vol. 2021, pp. 1–11, nov 2021.

[17] B. Zou, Y. Y. Gong, X. Xu, and Z. Yuan, "Assignment rules in robotic mobile fulfilment systems for online retailers," *International Journal of Production Research*, vol. 55, no. 20, pp. 6175–6192, 2017.

[18] C. Lee, B. Lin, K. Ng, Y. Lv, and W. Tai, "Smart robotic mobile fulfillment system with dynamic conflict-free strategies considering cyber-physical integration," *Advanced Engineering Informatics*, vol. 42, p. 100998, oct 2019.

[19] Y. Sun, N. Zhao, and G. Lodewijks, "An autonomous vehicle interference-free scheduling approach on bidirectional paths in a robotic mobile fulfillment system," *Expert Systems with Applications*, vol. 178, p. 114932, sep 2021.

[20] T. Lienert, T. Staab, C. Ludwig, and J. Fottner, "Simulation-based performance analysis in robotic mobile fulfilment systems analyzing the throughput of different layout configurations," *SIMULTECH 2018 - Proceedings of 8th International Conference on Simulation and Modeling Methodologies, Technologies and Applications*, no. Simultech, pp. 383–390, 2018.

[21] S. Wu, C. Chi, W. Wang, and Y. Wu, "Research of the layout optimization in robotic mobile fulfillment systems," *International Journal of Advanced Robotic Systems*, vol. 17, p. 172988142097854, nov 2020.

[22] S. Teck and R. Dewil, "A bi-level memetic algorithm for the integrated order and vehicle scheduling in a RMFS," *Applied Soft Computing*, vol. 121, p. 108770, may 2022.

[23] Y. Bao, G. Jiao, and M. Huang, "Cooperative optimization of pod repositioning and AGV task allocation in Robotic Mobile Fulfillment Systems," *Proceedings of the 33rd Chinese Control and Decision Conference, CCDC 2021*, pp. 2597–2601, 2021.

[24] N. Boysen, D. Briskorn, and S. Emde, "Parts-to-picker based order processing in a rack-moving mobile robots environment," *European Journal of Operational Research*, vol. 262, no. 2, pp. 550–562, 2017.

[25] J. Zhang, F. Yang, and X. Weng, "A Building-Block-Based Genetic Algorithm for Solving the Robots Allocation Problem in a Robotic Mobile Fulfilment System," *Mathematical Problems in Engineering*, vol. 2019, pp. 1–15, feb 2019.

[26] S. Teck and R. Dewil, "Optimization models for scheduling operations in robotic mobile fulfillment systems," *Applied Mathematical Modelling*, vol. 111, pp. 270–287, nov 2022.

[27] Y. Niu and F. Schulte, "Human Aspects in Collaborative Order Picking – What if Robots Learned How to Give Humans a Break?," pp. 541–550, Springer International Publishing, 2021.

[28] F. J. Aldarondo and Y. A. Bozer, "Expected distances and alternative design configurations for automated guided vehicle-based order picking systems," *International Journal of Production Research*, vol. 60, pp. 1298–1315, feb 2022.

[29] C. J. Hazard, P. R. Wurman, and R. D'Andrea, "Alphabet Soup: A Testbed for Studying Resource Allocation in Multi-vehicle Systems," *Proceedings of the AAAI Workshop on Auction-Based Robot Coordination*, 2006.

[30] M. Merschformann, L. Xie, and H. Li, "RAWSim-O: A simulation framework for robotic mobile fulfillment systems," *Logistics Research*, vol. 11, no. 1, pp. 1–11, 2018.

[31] M. Merschformann, L. Xie, and D. Erdmann, "Path planning for Robotic Mobile Fulfillment Systems," no. December 2018, 2017.

[32] L. Xie, N. Thieme, R. Krenzler, and H. Li, "Introducing split orders and optimizing operational policies in robotic mobile fulfillment systems," *European Journal of Operational Research*, vol. 288, no. 1, pp. 80–97, 2021.

[33] T. Lamballais, D. Roy, and M. B. De Koster, "Estimating performance in a Robotic Mobile Fulfillment System," *European Journal of Operational Research*, vol. 256, no. 3, pp. 976–990, 2017.