SN Computer Science (2022) 3:431
https://doi.org/10.1007/s42979-022-01296-6

ORIGINAL RESEARCH

Optimizations of a Multi-Agent System for a Real-World Warehouse Problem

Botond Ács[1] · László Dóra[1] · Olivér Jakab[1] · Alpár Jüttner[2] · Péter Madarasi[3] · László Z. Varga[1]

Received: 31 January 2022 / Accepted: 30 June 2022 / Published online: 8 August 2022

Abstract
In recent years, many warehouses applied mobile robots to move products from one location to another. We focus on a traditional warehouse where agents are humans, and they are engaged with tasks to navigate to the next destination one after the other. The possible destinations are determined at the beginning of the daily shift. Our real-world warehouse client asked us to minimize the total wage cost, and to minimize the irritation of the workers because of conflicts in their tasks. We define a heuristic for the optimizations for splitting the orders into warehouse carts, defining the sequence of the products within the carts, and the assignment of the carts to workers. We extend Multi-Agent Path Finding (MAPF) solution techniques. Furthermore, we have implemented our proposal in a simulation software, and we have run several experiments. According to the experiments, the make-span and the wage cost cannot be reduced with the heuristic optimization, however the heuristic optimization considerably reduces the irritation of the workers. We conclude our work with a guideline for the warehouse.

Keywords Multi-agent path finding · Task assignment · warehouse · Multi-agent optimization

Introduction

In recent years, many warehouses applied mobile robots to move products from one location to another. These applications gave rise to intensive research on problems related to the optimization of moving a team of agents in discrete time-steps on a graph while avoiding collisions. Big e-commerce companies are interested in this research [1]. This research has become more important nowadays, as the recent pandemic has increased the demand for e-commerce.

In traditional warehouses, the agents go to the shelves, they pick up the products from the shelves and put them into their warehouse cart. Then the agents take the cart to the exit of the warehouse and park the cart for transportation.

The movement of the agents need to be optimized, which is the main topic of Multi-Agent Path Finding (MAPF) [2]. The classical MAPF is a "one shot" problem. Each agent has a starting position and a destination position, anywhere in the warehouse. In the lifelong MAPF problem [3], the destination positions appear online, and the agents are constantly engaged with new tasks to navigate to the next destination. In our traditional warehouse, agents are also engaged with tasks to navigate to the next destination one after the other, but the possible destinations are determined at the beginning of the daily shift, because our wholesale warehouse client operates with daily orders. We call this traditional version of the problem, "semi-lifelong" MAPF problem.

Our client is a traditional warehouse where the agents are humans. Our real-world warehouse client asked us to tell the number of workers that is needed to be allocated to complete the daily shift in an optimal way. The optimization goals are to minimize the total wage cost, and to minimize the irritation of the workers when they have to walk around each other or they have to wait in front of a shelf for the other worker to finish. Such real-world scenario is not directly addressed by

This article is part of the topical collection "Frontiers in Practical Applications of Agents, Multi-Agent Systems, and Social Good" guest edited by Fernando De la Prieta and Frank Dignum.

✉ László Z. Varga
lzvarga@inf.elte.hu

[1] Faculty of Informatics, ELTE Eötvös Loránd University, Pázmány Péter sétány 1/C, Budapest 1117, Hungary

[2] Faculty of Science, ELTE Eötvös Loránd University, Pázmány Péter sétány 1/C, Budapest 1117, Hungary

[3] Faculty of Science, ELTE Eötvös Loránd University, and the MTA-ELTE Egerváry Research Group on Combinatorial Optimization, Eötvös Loránd Research Network (ELKH), Pázmány Péter sétány 1/C, Budapest 1117, Hungary

classic MAPF methods. We advance the state of the art by combining and extending classic MAPF methods to solve a real-world problem with heuristic optimizations.

In "Problem definition", we define the problem to be solved. In "Related work", we review the related work. In "Optimization of order splitting and routing", we present the proposed solution how to assign the orders to warehouse carts, and its experimental demonstration. In "Optimization of task assignment", we present the proposed solution to assign the warehouse carts to the agents of a multi-agent system (MAS), and how we have evaluated it in a simulation software. In "Evaluation", we evaluate the experimental results, and we give a guideline for our real-world warehouse client. "Conclusion" concludes the paper.

Problem Definition

We focus on a traditional warehouse where humans collect the products, but our proposal and results can be applied to robots as well. We use the word agent for human workers. Figure 1 shows the typical layout of the warehouse. The warehouse has a grid layout, and an agent with a warehouse cart occupies one cell in this grid. The shelves are organised in aisles. One shelf occupies one cell. The aisles are wide enough for three agents: one agent can pass, while two other agents stand in front of the shelves.

There is a given set of orders in each daily shift. There are $\mathcal{O}$ number of orders per day. The orders consist of several products. The orders have to be split into packages that can be collected into one or more warehouse carts.

The carts are moved by the agents on a route from the entrance door of the warehouse to the exit door. One agent can move maximum one cart. The agent and its cart move together, and they occupy one cell in each time step. When an agent finishes with a route, then it goes back to the entrance outside of the warehouse, and the agent can start its next route after $\mathcal{T}$ time.

The products of the orders have to be collected from the shelves. The agent and the cart stays for a short $\mathcal{S}$ time in front of a shelf while the products are collected from the shelf. If an agent is in front of a shelf and another agent wants to go to the same shelf, then the later arriving agent has to wait.

The products of an order have to be assigned to carts so that the products fit into the least number of carts. The set of products assigned to a cart is a package. The agent with the cart must visit the shelves containing the products assigned to the package.

There are $\mathcal{N}$ number of agents to move the carts. The daily shift lasts maximum $\mathcal{H}$ hours. The number $\mathcal{H}$ is fixed. The work-time of an agent within a shift starts when the first agent starts with its first cart, and the work-time ends when the last agent exits from the warehouse with its last cart.

Defining the sequence of the packages is the task assignment (TA). The make-span ($\mathcal{M}$) of the schedule is the time difference between the start of the work-time of the first agent and the end of the work-time of the last agent. The total wage cost is $\mathcal{C} = \mathcal{M} \times \mathcal{N}$. Note that the wage cost is proportional to the make-span, i.e. an agent also incurs cost when it is assigned to the warehouse work, but it is only waiting for the other agents to finish. This is because the agents are assigned to the job for the total make-span.

Our client asked us to solve the following problem: optimize the splitting of the orders into packages, create a schedule of the packages in accordance with the above description, and assign the packages to the agents. The schedule

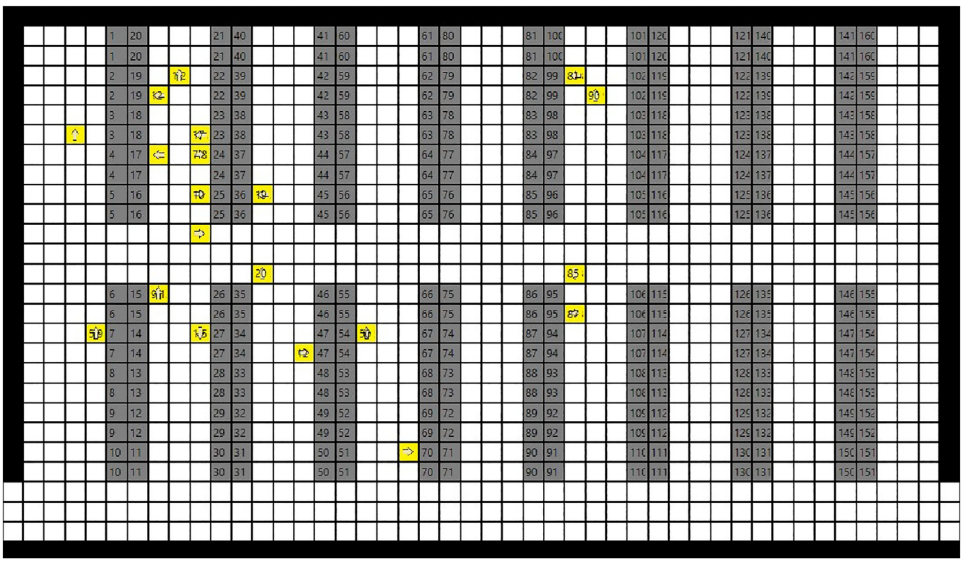

Fig. 1 A typical warehouse layout. The entrance is at the left-hand side and the exit is at the right-hand side. The shelves are indicated with grey cells. The agents pulling the carts are indicated with yellow cells.

must fit within the shift. Find the schedule that needs the least amount of wage cost, and in addition, try to minimize the extra movements and waits of the agents.

Related Work

A direct approach to solve the "one shot" MAPF problem optimally is to treat the team of agents as a single joint agent and then to apply a version of the A^* algorithm [4]. The Conflict Based Search (CBS) [5] algorithm treats the agents individually and applies a two level search. The optimal and complete CBS algorithm outperforms the A^* approach in many cases. Finding an optimal solution for the MAPF problem is computationally intractable [6]. Computationally tractable MAPF solvers produce sub-optimal solutions. Sub-optimal MAPF solvers include the *CA** [7] (incomplete and suboptimal), the PBS [8] (incomplete and suboptimal), and the ECBS [9] (complete and bounded suboptimal) algorithms.

The "one shot" MAPF addresses the problem when all the agents start at the same time to their destinations. This cannot be applied to our problem directly, because in our problem the agents usually start at different times. However, at specific moments, these algorithms can be applied to compute conflict free routes for the agents to their next destinations.

In a more realistic warehouse scenario, there are products on shelves and the agents have to deliver the products to delivery points. In this warehouse scenario, the agents can choose among the products, but then the destination of the product is fixed. The CBS-TA algorithm [10] is a complete and optimal solution for the task assignment (TA) and route finding problem, where the agents start at the same time, but they can choose among the targets to find an optimal solution for the conflict free routes. The CBS-TA algorithm is computationally intractable.

The CBS-TA algorithm cannot be applied to our problem, because in our case, the agents chose not only a single target, but a set of targets, which are the products that have to be collected to a warehouse cart. In addition, the CBS-TA algorithm is also a "one shot" MAPF problem, because the agents start at the same time. In our problem, the agents start to their set of destinations in a sequence. The heuristic optimization of our solution addresses the problem of this sequential choice of target sets.

In an even more realistic warehouse scenario, the MAPF problem is a lifelong problem where the optimal collision-free routes may change when new tasks appear. There are different methods [11] to approach the re-planning in the lifelong MAPF. One method is to solve the lifelong MAPF for all tasks as a whole [12]. Another method is to treat it as a sequence of MAPF problems at every time-step when a new task is assigned to an agent. At this time-step, the routes for all agents are re-planned [13, 14]. In the third method, the route is re-planned only for those agents which get new tasks [15].

These methods can be applied in our problem, because our agents have to re-plan their routes when they start for a new product. The computationally most demanding method is to solve the lifelong MAPF as a whole, and this method gives the best solution. The computationally less demanding is to re-plan only for those agents which get new tasks, but in this case the route is optimized only for a part of the agents. We chose to re-plan the routes for all agents when new tasks are assigned, because it optimizes the routes for all agents, but it is computationally not so demanding as the first method.

Frequent re-planning makes the computational complexity of the MAPF solvers more difficult to handle. The windowed MAPF approach helps to reduce this computational complexity [11]. In the windowed MAPF solver, the collisions are resolved only for the well-chosen next w time-steps ahead. The empirical results show [11] that the windowed approach produces close to optimal solutions.

In fact, there is no need to resolve the conflicts for the whole routes, if the routes are re-planned before the agents complete them. Because we chose to re-plan the routes for all agents when new tasks are assigned, and in our problem the agents start to new destination frequently, we chose to apply the windowed approach.

The following optimizations are also needed to solve our problem: the orders have to be split into smaller packages fitting the carts used to collect the items, and the sequence of the products within a package has to be determined so that the routes of the carts are optimal.

The first optimization problem is essentially a *Bin Packing Problem*, which can be easily modeled as an integer program [16], or alternatively, one can also use the approximation scheme given in [17].

The second problem is to find the best route for the collection of the products within a package. This means that the picking order of the items is to be determined. In many cases, a plain heuristic based on the map of the warehouse may be good enough, because the agents go from the entrance towards the exit, and they visit the aisles in this order. Once the agents enter an aisle, they collect all the products that are assigned to them and can be found in the given aisle. The sequence within an aisle follows a U-shape. However, in warehouses of more complex layout, obtaining a good solution is less obvious. In general, the problem to be solved is the *Travelling Salesman Problem* [18].

In this paper, we focus on an alternative approach, which is the simultaneous optimization of the splitting and the routing sub-problems. This problem can be considered as a *Vehicle Routing Problem* [19], which is computationally more

difficult, but usually provides significantly better solutions. In fact, our proposed algorithm quickly obtains an optimal solution for the vast majority of the real-world problem instances, as we will see in "Experiments".

Optimization of Order Splitting and Routing

Each order consists of a set of products that must be collected from the warehouse. This task is done by agents who move carts around the warehouse and pick the items from the shelves. However, an order might contain too many products that together do not fit on a single cart. In such cases, the order first has to be split into packages, and then each package is collected separately. Hence, the underlying multi-objective optimization problem asks for a *feasible split* of the order into packages (i.e., for each package, the corresponding products should fit on a single cart) while minimizing the *total number of carts* and the *total length of the routes* needed to complete the orders, and also ensuring the existence of *conflict-free paths* for the agents to follow.

Splitting the Orders into Packages

Products in large orders must be partitioned into smaller packages satisfying given size constraints. Furthermore, as an additional restriction, items in the same package must be placed on a cart in compliance with the following product placement rules.

Area based picking. Orders might contain products that can be found outside the warehouse area. Such items are filtered, forwarded to a dedicated worker, and then removed from the order. However, as at the end of the picking procedure these products are also placed on carts that correspond to the given order, their sizes must be taken into consideration when optimizing the total number of carts.

Volume constraints. Naturally, the dimensions of a cart imply upper bounds on both the dimensions and the total volume of items that can be placed on it. Each product has a detailed description in the database containing its dimensions as well, and this data is used when the size-constraints are added to the IP. Nevertheless, the range of available products in the warehouse changes dynamically, resulting in uncertainties and inaccuracies in the database. As a workaround, packages are formed with a safety margin that ensures the feasibility of the package even in problematic cases.

Weight constraints. Weight constraints are added in a similar way to the volume case. There is an upper bound on the total weight of items placed on the same shelf of a cart, while the total weight of a cart together with the products on it is also limited. It is worth mentioning that the latter limit might be different for different workers (depending on the physical strength of the worker).

Product placement rules. Due to hygiene and safety reasons, the placement of products on carts must respect certain rules. Chemical substances must be placed on the lowest shelf of the cart, separated from edible foods and feeds. Vegetables are also not allowed to be placed above other, non-chemical products. When partitioning the items of an order into packages, then these rules must be respected by all packages.

The problem of dividing an order into a fixed number of packages is basically a bin packing problem with additional constraints. It can be formulated as an integer linear program (IP) in a straightforward way by introducing a binary variable for all item-package pairs. This in turn can be solved using any of the open source mixed integer optimization solvers. This allows us to increase the number of the packages one by one as long as the model becomes solvable—when we have the minimum number of necessary packages. Our computational experiments show that an optimal solution can be obtained quickly for all instances coming from real life applications, hence one can easily determine the minimum number of necessary packages.

Picking Route Optimization

Keeping the number of carts as low as possible is only one aspect of the objective function. To ensure a maximum throughput of the whole picking procedure, products assigned to the same cart should be located 'close' to each other, thus helping the path finding heuristic to determine conflict free routes of small total length. Therefore, the objective function of the IP, that is used for partitioning the orders into packages, also depends on the pairwise distances of the items added to the same package.

This section describes a Mixed Integer Programming model for solving the joint problem of splitting the order into single carts and finding an optimal route for each of them.

The primary objective is to minimize the total length of the paths of the carts, while the number of carts is as small as possible.

Thus, the first step is to determine the number of necessary carts. One can easily see that this problem is a standard multiple constrained binpacking problem.

Then, assume that the current order fits to k carts, and let s_j and t_j denote the source and the destination for each cart $j = 1, \dots k$. Let S and T denote the set of all sources and all destinations, respectively. The items in the order will be denoted by I.

In what follows, we describe an approach to split the order to carts subject to resource constraints such that the total length of the paths of the carts is minimized. For the sake of simplicity, we restrict ourselves to the weight constraints, i.e. each item $i \in I$ has a given positive weight w_i

and the sum of the items to be placed on cart j shall not exceed its weight capacity W_j.

First, we build up a directed graph $G = (V, A)$, where $V = I \cup S \cup D$, and A is the union of the following four sets

$$A_{II} := \{uv : u, v \in I, u \neq v\} \tag{1a}$$

$$A_{SI} := \{sv : s \in S, v \in I\} \tag{1b}$$

$$A_{IT} := \{ut : u \in I, t \in T\} \tag{1c}$$

$$A_{TS} := \{t_j s_{j+1} : j = 1, \ldots, k-1\} \tag{1d}$$

We define a length function $l : A \longrightarrow \mathbb{R}$ on the arcs such that l_{uv} is the length of the shortest path from u to v, except for the dummy arcs $t_j s_{j+1} \in A_{TS}$, whose length is defined to be zero.

It is easy to see that a possible splitting of the order with a well-defined order in which the items must be collected corresponds to a Hamiltonian path from s_1 to t_k. The solutions of the following integer programming model will be such Hamiltonian paths, and it also ensures that the weight constraints are met.

$$\begin{aligned} &\min \sum_{uv \in A} l_{uv} x_{uv} \\ &\text{s.t.} \\ &x \in \{0,1\}^A \end{aligned} \tag{2a}$$

$$y \in \mathbb{R}^V \tag{2b}$$

$$\sum_{uv \in \Delta_{\text{out}}(u)} x_{uv} = 1 \quad \forall u \in V \setminus \{t_k\} \tag{2c}$$

$$\sum_{uv \in \Delta_{\text{in}}(u)} x_{uv} = 1 \quad \forall u \in V \setminus \{s_1\} \tag{2d}$$

$$y_u + w_v \leq y_v + (1 - x_{uv})M \quad \forall uv \in A_{II} \cup A_{SI} \tag{2e}$$

$$y_u \leq y_v + (1 - x_{uv})M \quad \forall uv \in A_{IT} \tag{2f}$$

$$y_{t_j} \leq \sum_{j'=1}^{j} W_{j'} \quad \forall j = 1, \ldots, k \tag{2g}$$

$$y_{s_j} = \sum_{j'=1}^{j-1} W_{j'} \quad \forall j = 1, \ldots, k \tag{2h}$$

where x_{uv} is a binary variable indicating whether arc $uv \in A$ is in the Hamiltonian paths or not. Constraints (2c) and (2d) ensure that the solution is a directed path from s_1 to t_k plus possibly some disjoint circles all together covering each node exactly once. The existence of such unconnected circles are prevented by the introduction of potential variables y. Due to Equations (2b) and (2e), the value of y increases by at least w_u along each edge uv of the solution - which would be impossible for a disjoint circle.

Finally, Eqs. (2g) and (2h) ensure that the sum of the weights along each subpath between s_j and t_j is at most W_j.

Speed up with Callback Heuristic

The model above may directly be solved by a general purpose MIP solver, however, its efficiency is rather limited on larger instances. Most MIP solvers provide flexible ways of problem-specific programmatic fine-tuning. One of them is implementing a *heuristic improving callback*. This callback function is regularly called by the MIP solver when a new feasible solution is found. In this callback, one can implement any algorithm that heuristically tries to improve the feasible solution. A good problem specific heuristic may significantly speed up the convergence of the MIP solver.

We implemented the following two improving steps.

The first improving step is the 2-OPT change [20] in one of the subpaths. Assuming that the current sequence of nodes of a subpath is $u_1, \ldots, u_m$, it basically looks for a pair of arcs $u_i u_{i+1}$ and $u_j u_{j+1}$ in the solution such that the path $u_1 \ldots, u_i, u_j, u_{j-1}, \ldots, u_{i+1}, u_{j+1}, \ldots, u_m$ is shorter than the current one. Then this process is repeated until no improving change is found.

Second, for each tour, we remove its nodes one-by-one and reinsert it to the best position. This step is iterated as long as a node can be inserted to a better position.

Experiments

We implemented the model given in " Picking route optimization" using CBC [21], and extended the solver with the heuristic callback described in "Speed up with callback heuristic". The effect of this extension was evaluated on realistic orders consisting of 12–24 items, four instances of each size. The experiments were carried out on an Intel i7-8650U CPU using CBC version 2.9.8 and the timeout was set to one hour. On the instances above, the callback improves the running time of the solver significantly, as shown in Fig. 2.

In another experiment, we use the same instances, but limit the running time of the algorithm to one minute, and investigate the quality of the found solution in comparison to the length of the shortest tour. Fig. 3 shows that the callback enables us to find the theoretically optimal tour in just under one minute in the vast majority of cases—in contrast to the default operation of CBC, which typically takes longer time to find a best tour.

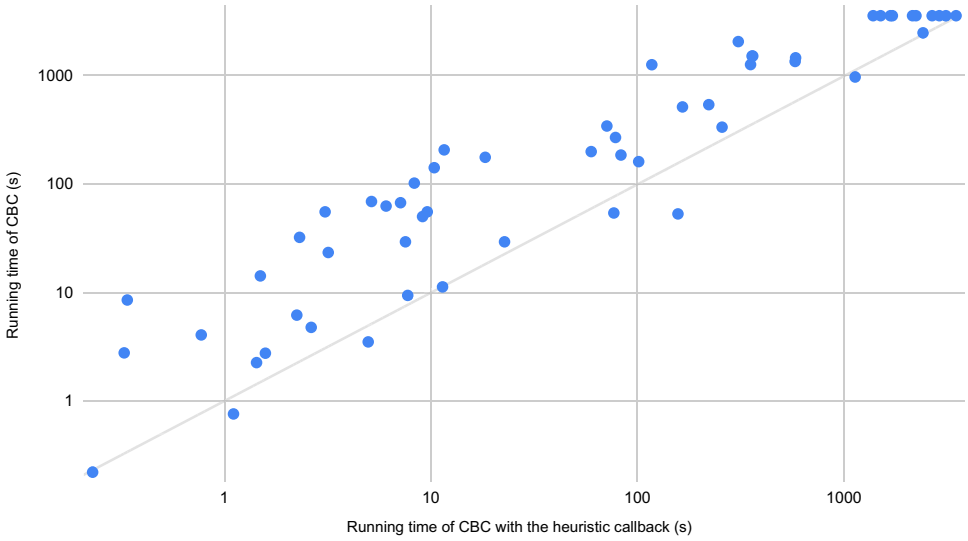

Fig. 2 Running time of CBC with and without the heuristic callback

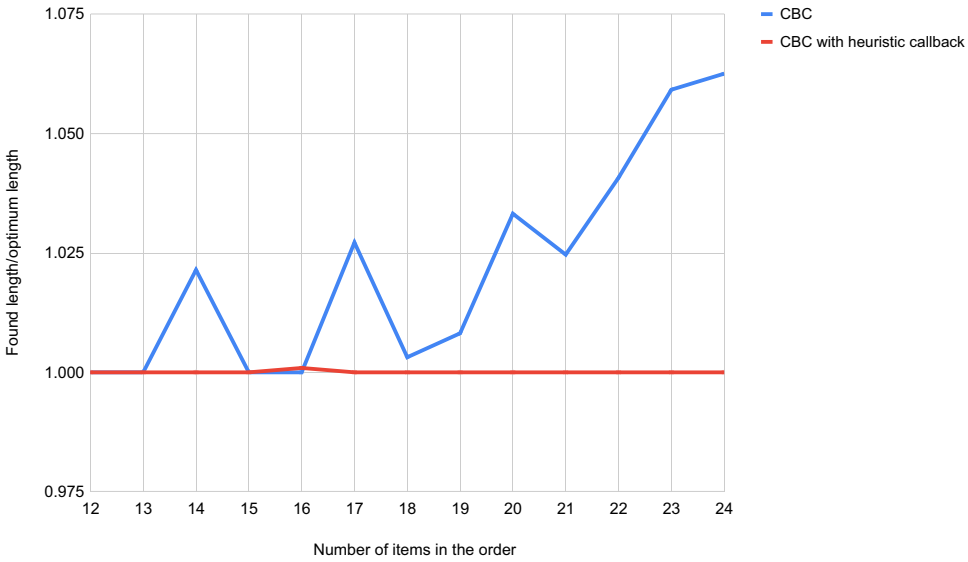

Fig. 3 Average length of the shortest tour found under one minute in proportion to the optimal solution

Optimization of Task Assignment

In our real-world warehouse, the MAPF problem has lifelong like features on two levels: the agents are engaged with new products one after the other within one package, and the agents are engaged with new packages one after the other within one daily shift. Therefore, we have to optimize the route for each package, and we have to optimize the order of the packages within one shift.

Heuristic Optimization of the Sequence of the Packages

The task assignment for the optimization of the route for a single package is given (see the previous "Optimization of order splitting and routing"), and we only have to optimize the conflict free routes of the agents from one product to the next product. We treat this problem as a sequence of MAPF problems at every time-step when an

agent starts for a new product. At this time-step, the routes for all agents are re-planned to find the best route for each agent. Although this needs a lot of computing effort, but if only the route for a single agent is re-planned, then completeness is not always guaranteed [11]. We use the CBS algorithm to find the optimal conflict-free routes. The CBS algorithm performs better in the case of bottlenecks, and the warehouse aisles are likely to be bottlenecks. Although the CBS algorithm is computationally hard, we used this algorithm, because the number of agents in the real-world warehouse is expected to be below 20. We used the windowed MAPF approach to reduce computational complexity, although it weakens the optimality of the CBS algorithm.

The classic MAPF solvers do not handle the problem of the waiting time in front of the shelves. Let us consider the situation when an agent arrives at a shelf and wants to stay in front of the shelf for $\mathcal{S}$ time-steps to complete its job. If a second agent also arrives at the same shelf during this time, then a classic MAPF solver might produce a solution where the first agent interrupts its job, the second agent stands in front of the shelf, the second agent completes its job, and then the first agent returns. A classic MAPF solver may even consider to interleave the jobs of the two agents. Such solution would not be acceptable in a real-world warehouse, therefore we modified the CBS algorithm to make the $\mathcal{S}$ time-steps in front of a shelf indivisible.

To optimize the order of the packages within one shift, we would like to avoid that agents have to go to the same shelf at the same time. This should be true all the time during the workshift. We know all the packages in advance, but avoiding the conflicts of the packages is a combinatorially hard problem. So we defined a heuristic algorithm (Algorithm 1) to solve it. Because an agent gets a new package when it finishes with the previous package, we want to assign the next package to the agent in a way that there are minimal number of conflicts with any of the precedingly assigned $\mathcal{N}$ number of packages.

Let us assume that we have a package $p_1 = \{pr_{(1,1)}, pr_{(1,2)}, ...pr_{(1,k)}\}$, where $pr_{(i,j)}$ are the products in the package in the order of their planned collection, and we have another package $p_2 = \{pr_{(2,1)}, pr_{(2,2)}, ...pr_{(2,l)}\}$ with the ordered list of products the same way. Let us assume that p_1 was assigned to agent a_1, p_2 was assigned to agent a_2, and the "delay" between the two assignments is d, which means that packages were assigned to $d-1$ number of agents after the package assignment to agent a_2 and before the package assignment to agent a_1.

The distance $distance(p_1, p_2, d)$ in the case of the above two packages p_1, p_2 and the delay d is defined with the number of products that are in both $\{pr_{(1,1)}, pr_{(1,2)}, ...pr_{(1,k)}\}$ and $\{pr_{(2,d)}, pr_{(2,d+1)}, ...pr_{(2,l)}\}$, if $d \leq l$, otherwise $distance(p_1, p_2, d) = 0$. The assumption in this definition is that agent a_1 cannot have a conflict with agent a_2 in collecting products $\{pr_{(2,1)}, pr_{(2,2)}, ...pr_{(2,d-1)}\}$, because agent a_1 starts with a delay d, and agent a_2 probably has already collected these products of its package. This assumption does not take into account the physical distances of the products in the warehouse, but it can be used as a rough estimate.

Note that if $distance(p_1, p_2, d) = 0$, then agent a_1 is happy to work on package p_1 with a delay d to agent a_2 working on package p_2. If $distance(p_1, p_2, d) > 0$, then agent a_2 would prefer bigger delay than d.

Given a set of packages P and a package p, then we collect from P the identifiers of the packages $p_i \in P$ that have the minimal $distance(p_i, p, d)$ values into the set C with the function $C = closest(P, p, d)$.

We use the heuristic defined by Algorithm 1. The function $next(R, S)$ returns a package to be assigned to an agent which is free to start to collect a package. The packages that have not yet been assigned to any agent are in set R. The currently assigned packages are in the ordered list $S = \{s_1, s_2, \ldots, s_k\}$. The package s_k was started by an agent the earliest, then the next agent started to collect package s_{k-1}, then the next agent started to collect package s_{k-3}, etc. When a package s_i is assigned to an agent, and the agent starts to collect s_i, then s_i is removed from R, and s_i is added to the beginning of S. When an agent finishes with a package s_j, and delivers s_j to the exit, then s_j is removed from S, and the agent asks for another package with $next(R, S)$ if R is not empty.

When the first agent in the shift asks for a package with $next(R, S)$, then S is empty, and Algorithm 1 assigns a random package from R to the first agent (lines 2–3). If S is not empty, then we collect the identifiers of those packages from R that are closest to s_1 into a set C (line 7), and here we take into account that there is a delay 1. If there is only one package in this set C, then this package will be assigned to the next agent (lines 8–10). Otherwise, we continue with s_2, and we reduce the set C to those packages that are closest to s_2 in case of a delay 2 (line 7). If there is only one package in C, then this package will be assigned to the next agent (lines 8–10). We continue with the packages in S until we reach the number of packages in S. Finally, if C still contains more than one package, then we return a random package from C (line 14–15).

An additional optimization is that when the shift starts, then we do not start the agents at the same time, to reduce the collision conflicts in the first aisles. The agents start in the beginning of the shift with a delay bigger than the waiting time $\mathcal{S}$ in front of the shelves.

Algorithm 1 $p = next(R, S)$: select the next package

Require: A non-empty finite set $R = \{r_1, r_2, \ldots, r_m\}$ of remaining packages, and an ordered list $S = \{s_1, s_2, \ldots, s_k\}$ of currently assigned packages

Ensure: p is the package to be assigned to the next agent

```
if S is empty then
    p ⇐ r_1
    return p
else
    C ⇐ R
    for i ⇐ 1 to k do
        C ⇐ closest(C, s_i, i)
        if |C| = 1 then
            p ⇐ the only c ∈ C
            return p
        end if
    end for
end if
p ⇐ any c ∈ C
return p
```

Experimental Setup

The above solutions were implemented in a simulation program written in C#. The inputs to the simulation program are the layout of the warehouse, the position of the products in the warehouse, the position of the agents in the warehouse, and the list of packages with the ordered list of products in them. The waiting time $\mathcal{S}$ in front of a shelf and the window w to resolve the conflicts ahead can be set as parameters. The simulation can be run with and without the heuristic optimization of the order of the packages. When the simulation ran for too long time (one day was the limit), then we stopped it.

The layout of the real-world is similar to the one in Fig. 1. We used this layout in our simulations. Each product type can be found on two neighbouring shelves. The products are evenly distributed among the shelves in the warehouse. The real-world warehouse typically employs about 15 agents to deliver the orders. We created scenarios for agents between 1 and 20. The typical size of a package is around 16 products in the real-world warehouse, and usually only one package has smaller size within an order. In our scenarios, all the packages have 16 products. We have limited the number of packages to 40 in our scenarios to be able to run as many experiments as possible. The package number 40 was selected, because it means two packages per agent in the case of 20 agents, which is the maximum in our scenarios. The scenarios with these settings are similar to the real-world scenarios.

We have created several scenarios for the experiments. Products were randomly generated for two package sets $p1$ and $p2$. We assume that these package sets are the output of the optimal splitting of the orders. The products in the packages are ordered similarly to the heuristic explained in "Optimization of order splitting and routing". The scenarios were run with and without the optimization of Algorithm 1. When Algorithm 1 was not used, the packages were assigned to the agents in the order as they were randomly generated in package sets $p1$ and $p2$.

The window size w has to be greater than the job completion time $\mathcal{S}$, so that the extended CBS algorithm can handle the conflict of two agents aiming to the same shelf at the same time. To reduce computation time, we set the job completion time $\mathcal{S}$ to 2 time-steps. We chose three different lookahead window sizes: 5, 10 and 100. Minimum window size 5 is needed for an agent to avoid the collision with another one. The window size 10 is about the same as the lookahead range of humans. The lookahead range 100 is about enough to find the optimal solution to the next product anywhere in the warehouse. Because replanning usually occurs more frequently than 10 steps, the plans above 10 steps are almost always dropped.

The simulation scenarios of all the above mentioned parameter combinations were run. Each scenario was a single run, because there is no uncertainty in execution. Most of the simulation runs could be completed in time, however a few of them had to be stopped because they ran for too long time. Table 1 shows which simulation runs were

Table 1 The completed simulations (marked with tick signs)

Parameters				Number of agents																	
Package set	Window size	Wait steps	Optimization	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
p1	5	2	Y	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
p1	5	2	N	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–
p1	10	2	Y	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
p1	10	2	N	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–	–	–	–
p1	100	2	Y	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
p1	100	2	N	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–	–	–	–	–	–	–
p2	5	2	Y	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
p2	5	2	N	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	-	✓	✓	✓	-	✓	✓
p2	10	2	Y	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
p2	10	2	N	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	✓	✓	✓	–	–
p2	100	2	Y	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	✓	✓
p2	100	2	N	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	✓	–	–	–	–	–

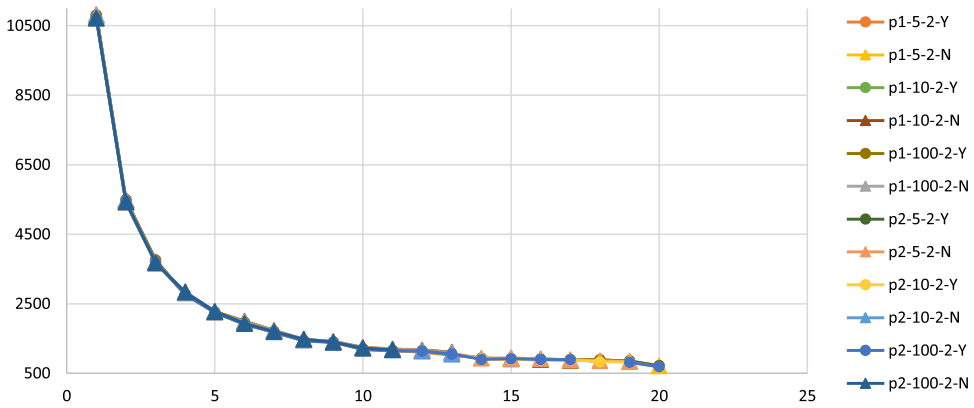

Fig. 4 The make-span $\mathcal{M}$ of the experiments

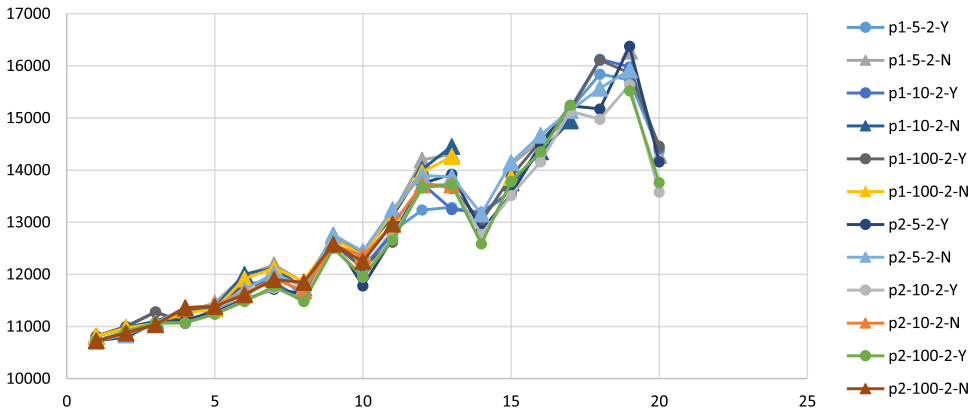

Fig. 5 The total wage cost $\mathcal{C}$ in the experiments

completed. In some cases (e.g. p2/5/2/N or p2/100/2/N for 14 agents) the not optimized simulation could not be completed because of unlucky coincidences causing excessive amount of conflict resolution.

Evaluation

The following diagrams show the results of the simulations with the multi-agent system using the task assignment heuristic of "Optimization of task assignment". The data

series legends in the diagrams use the same notations for the parameters as Table 1. If the experiment uses the heuristic optimization of the sequence of the packages, then the marker of the data series is a filled circle. If the experiment does not use this heuristic optimization, then the marker of the data series is a filled triangle. The horizontal axis is the number of agents on all diagrams.

The statistics of the make-span $\mathcal{M}$ (in time-steps) of the experiments are shown in the diagram of Fig. 4. It seems that neither the lookahead window nor the heuristic optimization has effect on the make-span. We will see on the diagram of Fig. 7 that there are only small differences.

The statistics of the wage cost $\mathcal{C}$ in the experiments are shown in the diagram of Fig. 5. There are significant decreases in the trend of the wage cost when the number of agents is a divisor of the number of packages, because in this case the packages are evenly distributed among the agents, and there is no need for an extra round with only a few agents. The wage cost is about 50% higher when we apply 19 agents instead of 1, i.e. 19 agents do not reduce the make-span to $\frac{1}{19}$ of the single agent make-span. It seems that neither the lookahead window nor the heuristic optimization has effect on the wage cost.

The statistics of the irritation $\mathcal{I}$ in the experiments are shown in the diagram of Fig. 6. The irritation $\mathcal{I}$ is the difference between the sum of the actions of all agents of the given experiment and the number of actions of the single agent experiment. The scenario with the single agent is the reference for the total number of steps of an "irritation free" solution. If there is only one agent, then it delivers the packages one after the other, and it can always go on the shortest possible route to the next product. There is a clear difference between the irritation in the heuristically optimized scenarios (circle markers) and the heuristically not optimized (i.e. random sequence of packages) scenarios (triangle markers). The heuristic optimization considerably reduces the irritation of the agents. It seems that the lookahead window does not have effect on the irritation. We will analyse this on the diagram of Fig. 8.

The effect of the heuristic optimization and the lookahead window on the wage cost $\mathcal{C}$ is shown in the diagram of Fig. 7. The small sized markers are for the lookahead

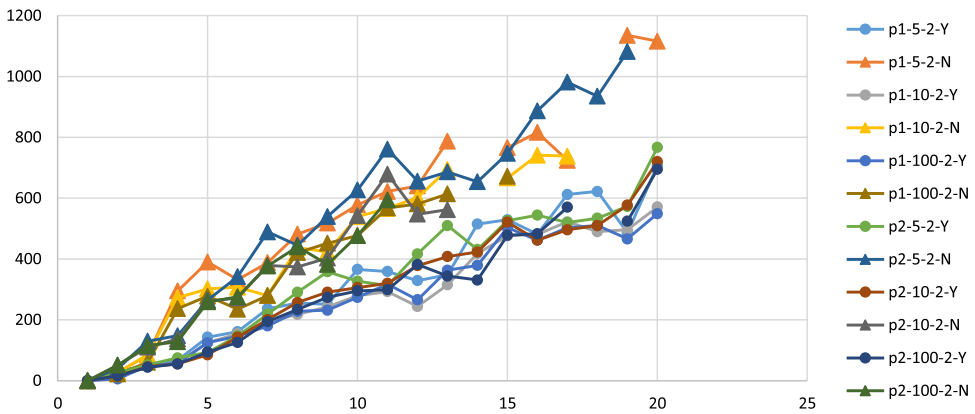

Fig. 6 The irritation $\mathcal{I}$ in the experiments

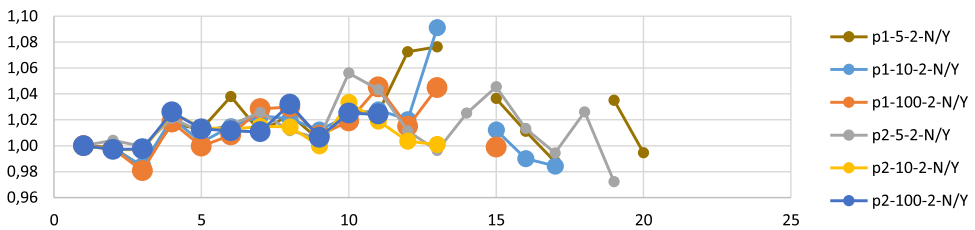

Fig. 7 The ratio between the wage cost of the heuristically not optimized and the wage cost of the heuristically optimized scenarios

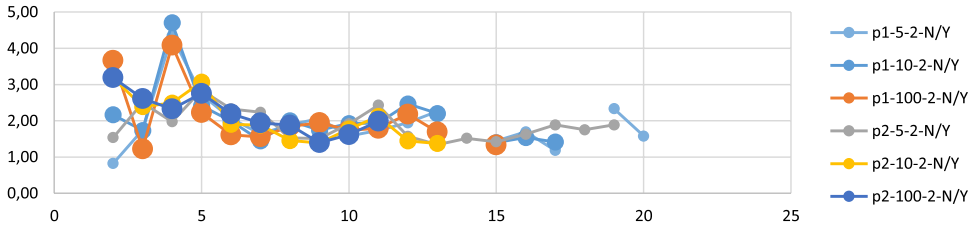

Fig. 8 The ratio between the irritation in the heuristically not optimized and irritation in the heuristically optimized scenarios

window 5, the middle-sized markers are for the lookahead window 10, and the large sized markers are for the lookahead window 100. The diagram for the make-span ratios would be the same, because $\mathcal{C} = \mathcal{M} \times \mathcal{N}$. There are no big differences between the optimized and the not optimized wage costs. The not optimized wage cost is at most 10% higher than the optimized wage cost, but in few cases the not optimized is at most 3% better. It seems that the lookahead window does not have effect on the wage cost, because there is no clear trend.

The effect of the heuristic optimization and the lookahead window on the irritation $\mathcal{I}$ is shown in the diagram of Fig. 8. There are significant differences between the optimized and the not optimized irritation. The not optimized irritation is in a few cases 3 times higher than the optimized irritation, and in most of the cases, the not optimized irritation is around 2 times higher. The optimization seems to help the most when there are fewer agents. If the number of agents is significantly less than the number of packages, then there is more room for optimization. It seems that the lookahead window does not have effect on the wage cost, because there is no clear trend.

After all, how many agents are needed?

The guideline for our real-world warehouse client is the following: Use the heuristic optimization of "Optimization of order splitting and routing" to split the orders into packages. Use Algorithm 1 for the heuristic optimization of the order of the packages to reduce the irritation of the agents, and apply as few agents as possible, because it reduces both wage cost and irritation. In our experiments, if $\mathcal{H} = 2500$, then (from the diagram of Fig. 4) $\mathcal{N} = 5$. This way the work can be completed within the daily shift (Fig. 4), the wage cost is reduced (Fig. 5), and the irritations of the workers are reduced (Fig. 6).

Conclusion

In this paper, we have investigated the problem of a real-world warehouse. The warehouse want to know the number of workers that is needed to minimize the wage cost, and to minimize the irritation of the workers when they have to walk around each other, or they have to wait in front of a shelf for the other worker.

We have proposed a solution which uses MAPF solution techniques with the following additions:

(1) We have used callback heuristic to optimize the splitting of the orders into packages.
(2) We have extended the CBS algorithm to handle the waiting time in front of the shelves as an indivisible action.
(3) We have solved a "semi-lifelong" MAPF. We have used the method of re-planning for all agents whenever there is a need for a new destination.
(4) We have used the windowed solution to reduce the computing complexity of the "semi-lifelong" MAPF.
(5) We have defined a heuristic optimization for the assignment of the packages.

According to the experiments, the callback heuristic finds the optimal splitting of the orders into packages in most of the real-world problem instances. The make-span and the wage cost cannot be reduced with the heuristic optimization of the sequence of the packages, however the heuristic optimization considerably reduces the irritation of the agents. The lookahead window seems to be indifferent, so the short range of human lookahead may perform as well as a long range computational lookahead. We have concluded our work with a guideline for the real-world warehouse client. We plan to investigate if and how to implement the ideas in the real ERP system of the client.

Acknowledgements We thank A. Kiss for facilitating the initial programming work. The work of B. Ács, O. Jakab and L. Dóra was supported by the European Union, co-financed by the European Social Fund (EFOP-3.6.3-VEKOP-16-2017-00002). The work of A. Jüttner and P. Madarasi was supported by GINOP-2.2.1-15-2017-00085. The work of L.Z. Varga was supported by the "Application Domain Specific Highly Reliable IT Solutions" project which has been implemented with the support provided from the National Research, Development and Innovation Fund of Hungary, financed under the Thematic Excellence Programme TKP2020-NKA-06 (National Challenges Subprogramme) funding scheme.

Funding Open access funding provided by Eötvös Loránd University.

Declarations

Conflict of interest On behalf of all authors, the corresponding author states that there is no conflict of interest.

References

1. Wurman PR, D'Andrea R, Mountz M. Coordinating hundreds of cooperative, autonomous vehicles in warehouses. AI Mag. 2008;29(1):9.

2. Stern R, Sturtevant NR, Felner A, Koenig S, Ma H, Walker TT, Li J, Atzmon D, Cohen L, Kumar T K S, Barták R, Boyarski E. Multi-agent pathfinding: Definitions, variants, and benchmarks. In: Proceedings of the Twelfth International Symposium on Combinatorial Search, SOCS 2019, Napa, California, 16-17 July 2019, pp. 151–159. AAAI Press, Palo Alto, California (2019)
3. Ma H, Li J, Kumar T K S, Koenig S. Lifelong multi-agent path finding for online pickup and delivery tasks. In: Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2017, São Paulo, Brazil, May 8-12, 2017, pp. 837–845. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC (2017)
4. Hart P, Nilsson N, Raphael B. A formal basis for the heuristic determination of minimum cost paths. IEEE Trans Syst Sci Cybernet. 1968;4(2):100–7. https://doi.org/10.1109/tssc.1968.300136.
5. Sharon G, Stern R, Felner A, Sturtevant NR. Conflict-based search for optimal multi-agent pathfinding. Artificial Intell. 2015;219:40–66.
6. Yu J, LaValle S. Structure and intractability of optimal multi-robot path planning on graphs. Proceedings of the AAAI Conference on Artificial Intelligence **27**(1) (2013)
7. Silver D. Cooperative pathfinding. In: Proceedings of the First AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment. AIIDE'05, pp. 117–122. AAAI Press, Palo Alto, California (2005)
8. Ma H, Harabor D, Stuckey PJ, Li J, Koenig S. Searching with consistent prioritization for multi-agent path finding. Proc AAAI Conf Artificial Intell. 2019;33(01):7643–50.
9. Max B, Guni S, Roni S, Ariel F. Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem. Front Artificial Intell Appl. 2014;263:961–2.
10. Hönig W, Kiesel S, Tinka A, Durham JW, Ayanian N. Conflict-based search with optimal task assignment. In: Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems. AAMAS '18, pp. 757–765, Richland, SC (2018)
11. Li J, Tinka A, Kiesel S, Durham JW, Kumar TKS, Koenig S. Lifelong Multi-Agent Path Finding in Large-Scale Warehouses, pp. 1898–1900. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC (2020)
12. Nguyen V, Obermeier P, Son TC, Schaub T, Yeoh W. Generalized target assignment and path finding using answer set programming. In: Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, pp. 1216–1223 (2017)
13. Grenouilleau F, van Hoeve W, Hooker JN. A multi-label A* algorithm for multi-agent pathfinding. In: Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling, ICAPS 2018, Berkeley, CA, USA, July 11-15, 2019, pp. 181–185. AAAI Press, Palo Alto, California (2019)
14. Wan Q, Gu C, Sun S, Chen M, Huang H, Jia X. Lifelong multi-agent path finding in a dynamic environment. In: 2018 15th International Conference on Control, Automation, Robotics and Vision (ICARCV), pp. 875–882. IEEE, not identified (2018)
15. Liu M, Ma H, Li J, Koenig S. Task and path planning for multi-agent pickup and delivery. In: Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems. AAMAS '19, pp. 1152–1160. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC (2019)
16. Martello S, Toth P. Knapsack problems: algorithms and computer implementations. New York, United States: Wiley; 1990.
17. de la Vega WF, Lueker GS. Bin packing can be solved within $1+\epsilon$ in linear time. Combinatorica. 1981;1(4):349–55.
18. Applegate DL, Bixby RE, Chvátal V. Traveling Salesman Prob. Princeton, New Jersey, United States: Princeton University Press; 2007.
19. Toth P, Vigo D. Vehicle routing: problems, methods, and applications. Philadelphia, USA: SIAM; 2014.
20. Lin S. Computer solutions of the traveling salesman problem. Bell SystTech J. 1965;44(10):2245–69.
21. Forrest J, Ralphs T, Santos HG, Vigerske S, Hafer L, Forrest J, Kristjansson B, jpfasano EdwinStraver Lubin M, rlougee jpgoncal1 Jan-Willem, h-i-gassmann, Brito S, Cristina Saltzman M, tosttost, MATSUSHIMA F, to-st: coin-or/Cbc: Release releases/2.10.7. Zenodo (2022). https://doi.org/10.5281/zenodo.5904374.