Warehouse Order Dispatching Problem

Federico Brandini Alessandro Mazzocchi

May 5, 2025

Abstract

In a warehouse, items are collected in zones and they can be moved between them by carts. When a new order that requires moving an item between two zones enters the dispatching system, the latter chooses a free cart to take charge of it according to a predefined policy. About the latter, it could be preferable to consider the time to manage orders, the travelled distance (to limit wear) and the energy consumption of carts (a very current issue). Traditional policies can hardly achieve all these objectives, then we try to find a good policy using a branch of Machine Learning: the Evolutionary Computation, in particular the Genetic Programming (GP). Using it, we found better policies on single objectives in part, but we found also other ones which join the best features of the all traditional ones.

1. Problem Explanation

We consider a warehouse of dimensions $W \times H$ where there are some zones and carts. A **zone** is a warehouse point $z = (x_z, y_z)$ where we can find items; a **cart** is a transport that allows us to move objects among zones and it can be defined by a tuple $c = (v_c, p_c)$, where v_c is the average speed and p_c the dissipated power of the cart when it is moving.

An **order** is defined by a couple of zones $o = (z_o^{pick}, z_o^{drop})$ and indicates that an object has to be moved from z_o^{pick} to z_o^{drop}. When an order is received by the dispatching system, it chooses a non-busy cart which must manage the order according to some predefined policy. The system manages orders sequentially and free carts do not move.

Let us indicate with $Z = \{z_i : i = 1, .., n_Z\}$ the set of warehouse zones and with $C = \{c_i : i = 1, .., n_C\}$ the set of warehouse carts. A **policy** can be defined as a function which receives

- the current order o to manage,
- Z,
- C,
- the current position of carts and
- their busy flags (i.e., indicates if a cart is busy or not),

and returns the index $i \in \{1, ..., n_C\}$ of the selected cart. If all carts are busy, the dispatching system waits until at least one cart becomes free before calling the policy; in fact, the existence of at least one free cart is a function precondition.

In this project, we try to find a policy that allows the achievement of some or all of the following objectives using the Genetic Programming:

- minimizing the average **time** to manage an order;
- minimizing the average **distance** to be travelled by carts to manage orders;
- minimizing the average **consumption** of carts on managing orders.

Particularly for the last objective, we suppose that if a cart c_i travels a distance d, its consumption e is obtainable with the following formula.

$$e = p_{c_i} \frac{d}{v_{c_i}} \tag{1}$$

Once found, the results of the GP policy will be compared with those of the following three traditional policies:

- RR (Round Robin), which consists in a cyclic assignment of orders to free carts;
- NAF (Nearest AGV First), which assigns the order to the nearest free cart;
- SPTF (Short Processing Time First), which assigns the order to the free cart that will take the least time to execute it.

In this project, we assume that warehouse carts are equipped with autonomous collision avoidance systems. These systems automatically manage possible conflicts by slowing down or temporarily stopping the carts if necessary. However, after avoiding collisions, carts may accelerate to recover lost time, resulting in an effective average speed consistent with the one modelled. Thus, the dispatching optimization remains valid without explicitly simulating collisions.

This problem takes place at the intersection among multi-objective optimization [1], the operative challenges in warehouse management [2] and the vehicle routing [3].

2. Proposed Solution

2.1 Individual

To reduce the complexity of the problem, we introduce a relaxation of the decision process. Instead of directly searching for a policy to assign an order to the correct free cart, the process is decomposed into two steps (as reported in Figure 1):

1. a GP-evolved evaluation function assesses the carts' goodness to manage the order;
2. a deterministic selection assigns the order to the best free cart.

Now, the problem is finding the best evaluation function using GP such that once received

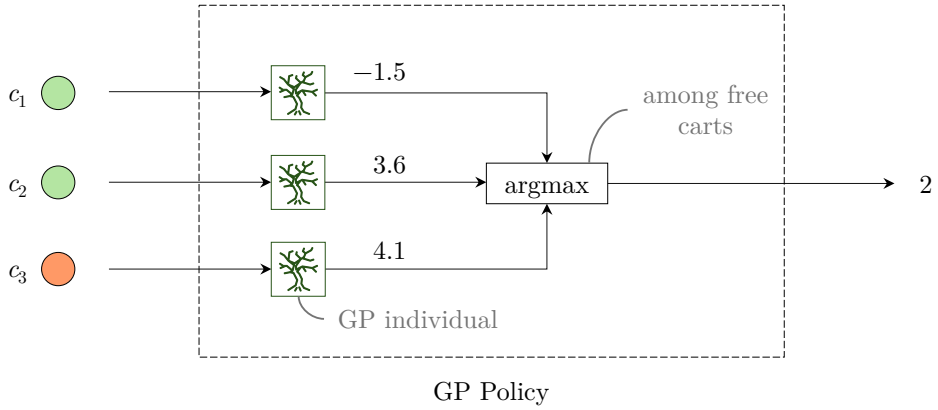

Figure 1: GP policy high-level architecture

- the order o to manage,
- a cart c_i and
- the current position of the cart c_i,

returns a real number representing the current goodness of the cart c_i to take charge of the order o. The greater the value, the better the cart.

2.2 Fitness function

To evaluate each individual of the population, the fitness function simulates a warehouse scenario where the dispatching system manages orders with a GP policy that uses the individual to evaluate as goodness criteria. Furthermore, fitness function computes internally the average of time, distance and consumption to execute orders and it returns

- the weighted sum of them, if we choose a **single-value** fitness function;
- a triple of values, if we choose a **multiple-value** fitness function.

In any case, if we consider two fitness $a = (a_1, ..., a_n)$ and $b = (b_1, ..., b_n)$ we can say that

$$a > b \text{ if } \begin{cases} \forall i \in \{1, ..., n\}, a_i \geq b_i \\ \exists i \in \{1, ..., n\} : a_i > b_i \end{cases} . \tag{2}$$

In this case, we can also say that a **dominates** b. This definition is for maximization problems, but an analogue one can be set for minimization or other mixed problems. In another way, we can multiply the objective values to minimize by -1. This order relation is used to compare two individuals in the population of a GP algorithm and it is named the **Pareto order relation**. We can note that if $n = 1$, this order relation degenerates into that between real numbers.

In this project, when we consider only one objective, we decide to use the Hall of Fame to capture the best-so-far individual in the population. Instead, when we consider more objectives, we decide to compare the results obtained using a single-value fitness function and the Hall of Fame with those obtained using a multiple-value fitness function and the Pareto Front.

The **Pareto Front** is a Hall of Fame without limits of size and contains all the best-so-far individuals on the population. The reason for which this data structure does not have a size limitation is because the Pareto order relation is a partial order relation, but not a total one. It means that, if we consider two fitness values a and b, it can happen that neither $a \geq b$ nor $b \geq a$, meaning what no individual dominates the other one. In the end, the Pareto Front should contain all optimal solutions, where each one is such that it is not possible to improve an objective value without worsening another one (the conditional is mandatory when we talk about non-deterministic strategies). This math object is useful to see the tradeoff among all objectives, especially when we aren't able to set their weights.

2.3 Algorithm general structure

The algorithm structure basically follows the *eaSimple* one (see https://github.com/DEAP/deap/blob/master/deap/algorithms.py), with some additions about elitism and early stopping.

Elitism. For each generation, after replacing the population with the new one, we check if all best-so-far individuals belong to the population yet. If it isn't true, all deleted individuals are added to the population, replacing the worst ones.

Early stopping. To prevent overfitting and reduce useless generations, we decide to create a validation set obtained from the dataset. Every certain number of generation, the evolution is stopped to validate the best-so-far individuals captured during training. To do this, we extend the `HallOfFame` and the `ParetoFront` classes in `ValidationHallOfFame` and `ValidationParetoFront` ones respectively. The latter two classes allow us to capture the best-so-far individuals on validation set among the best-so-far ones on the training set. After a certain number of consecutive non-improvements in validations, the training stops.

3. Results

To implement the GP algorithm we choose to use the DEAP framework (*Distributed Evolutionary Algorithm in Python*). In Table 1 we report the main operators chosen for the algorithm, in Table 2 the primitive set and in Table 3 more details about hyperparameters set. The information in those tables apply to every run. For more details, you can consult the code at https://github.com/fedebrando/WarehouseOrderDispatchingProblem.git.

Operation	**Operator**	**Note**
Selection	`selTournament`	`tournsize = 3`
Recombination	`cxOnePoint`	
Mutation	`mutUniform`	

Table 1: GP Operators

In this section, we try to solve a particular problem instance shown in Figure 2 and described in Table 4. To learn a good policy, we generated a random dataset of 12000

Terminal	**Normalization**		**Note**
	Min	**Max**	
$x_{z_o^{pick}}$	0	W	Pick zone's abscissa
$y_{z_o^{pick}}$	0	H	Pick zone's ordinate
$x_{z_o^{drop}}$	0	W	Drop zone's abscissa
$y_{z_o^{drop}}$	0	H	Drop zone's ordinate
x_c	0	W	Cart's abscissa
y_c	0	H	Cart's ordinate
v_c	0	v_{max}	Cart's average speed
ERC $\sim \mathcal{U}(\{-1, 0\})$	-	-	Ephemeral Random Constant in $\{-1, 0\}$

(a) Terminal set (we do not insert the power p_c among inputs because it is equal for all carts in our problem instance and let us define $v_{max} := \max\{v_{c_i} : i = 1, ..., n_C\}$)

Function	**Arity**	**Note**
$+$	2	Addition
$-$	2	Subtraction
$\times$	2	Multiplication
$-$	1	The opposite
$\sin(\cdot)$	1	Sine
$\cos(\cdot)$	1	Cosine

(b) Function set

Table 2: Primitive set

Hyperparameter	**Value**
Max generations	200
Validation frequency (generations)	5
Early stopping patience (generations)	5
Population size	1000
Population initialization	`genHalfAndHalf`
Recombination probability	0.7
Recombination height limit	20
Mutation probability	0.05
Mutation height limit	20
Mutation subtree min height	0
Mutation subtree max height	2

Table 3: Hyperparameters

orders, split in training set (7000 orders), validation set (1500 orders) and test set (3500 orders). The timing gap between two consecutive orders is set allowing the system usually

to choose among more than one cart, otherwise the choice would have been independent from the policy. In particular, it is sampled from an exponential distribution with failure rate $\lambda = \frac{1}{50}$, then with an average of $\frac{1}{\lambda} = 50$.

To compare GP policies with traditional ones, let us show the objective values of the latter ones on the test set with seven significant digits in Table 5.

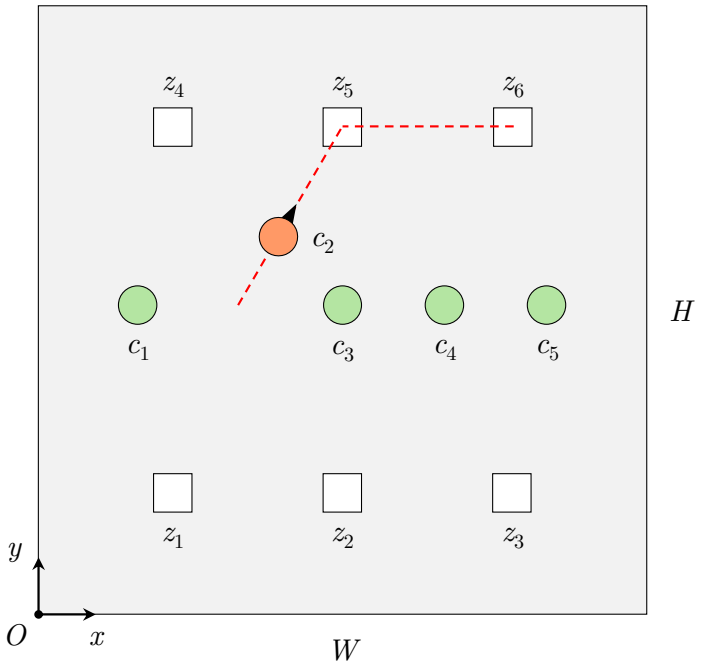

Figure 2: WODP instance representation

3.1 Single-value fitness function and Hall of Fame

First of all, we try to find a great solution for each objective separately, then for each unordered couple of them and at the end with all objectives. In the latter two cases, weights are set to normalize values about to 10^2 order. To do that, when they are present, we choose

- -10 for time objective,
- -1 for distance one and
- -10^{-2} for consumption.

Furthermore, only for a single-objective optimization, we add also a size penalty term to fitness function to prefer the smaller tree in presence of much similar fitness values. Note that weights are negative because the GP algorithm always works for maximization. Results are reported in Table 6 and, even if the algorithm considers only the weighted sum of the objectives to find a great solution, for more understanding, we decide to show all single objective values anyway.

z_i	x_{z_i}	y_{z_i}
z_1	100	100
z_2	250	100
z_3	400	100
z_4	100	400
z_5	250	400
z_6	400	400

(a) Zones

c_i	v_{c_i}	p_{c_i}
c_1	5	500
c_2	5	500
c_3	5	500
c_4	10	500
c_5	10	500

(b) Carts

Data	Value
W	500
H	500
c_1 start position	$(50, 250)$
c_2 start position	$(150, 250)$
c_3 start position	$(250, 250)$
c_4 start position	$(350, 250)$
c_5 start position	$(450, 250)$

(c) Other data

Table 4: WODP instance data

Policy	**Time**	**Distance**	**Consumption**
RR	82.24843	521.8959	40911.81
NAF	68.38775	412.9294	34093.59
SPTF	53.77169	438.4362	26835.38

Table 5: Objective values of traditional policies on test set

Seed	**Time**	**Distance**	**Consumption**
5392847103	**53.34285**	-	-
4728391056	-	**410.2400**	-
3950182478	-	-	26879.65
2847509139	55.09515	**411.3326**	-
1529473807	-	422.9373	27729.64
8479201530	54.27765	-	27096.91
5821049738	**53.37485**	426.0039	**26652.06**

Table 6: Objective values of GP policies on test set found by the GP algorithm using single-value fitness function and the Hall of Fame (for each configuration of enabled objectives, we choose the best seed between two on validation set)

3.2 Multiple-value fitness function and Pareto Front

Since it is useless finding solutions with only one objective using the Pareto front, in this case we always work with at least two objectives enabled. Differently from the

previous case, here the GP algorithm returns the Pareto front, which may contain more than one solution. For this reason, we will choose that one which allows us to show the Pareto front's power. For example, let us see the Pareto fronts of the run with time and distance objective enabled in Figure 3 and select our subjective best tradeoff found. In the same way, we choose solutions to test for each configuration of enabled objectives (see Figure 4, Figure 5 and Figure 6). In all graphics, we decide not to show RR point (Time: 83.06662, Distance: 524.1789, Consumption: 41336.25) because it is dominated by all other points and it would uselessly decrease the scale of the graphics.

In order to define a deterministic way to find a best solution in the Pareto front, we could define a cost function f to minimize on the front curve.

In Table 7 we report the result for multiple-value fitness function used in conjunction with the Pareto front.

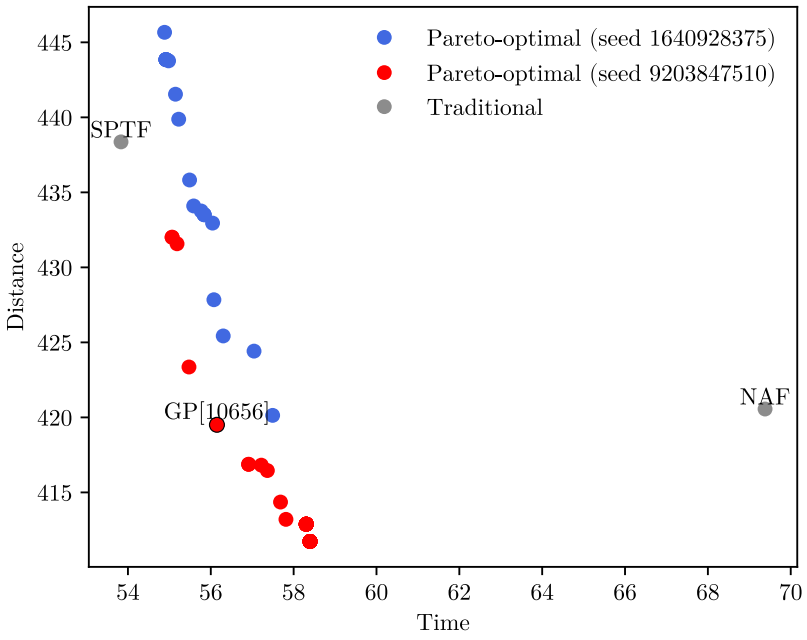

Figure 3: Validation Pareto fronts obtained from the GP algorithm with **time and distance** objectives enabled, compared with points of relevant traditional policies evaluated on the validation set

Seed	Choice	Time	Distance	Consumption
9203847510	GP[10656]	55.81822	416.2828	-
4938051729	GP[22]	-	415.0762	28001.76
8204719532	GP[0]	55.19450	-	27559.59
6074928513	GP[52]	55.89215	413.9534	27916.43

Table 7: Objective values of GP policies on test set found by the GP algorithm using multiple-value fitness function an the Pareto Front (for each configuration of enabled objectives, we choose the best seed between two ones on validation set, comparing the Pareto fronts)

4. Conclusions

After seeing results (see Table 6 and Table 7), we can say that GP allows us to find solutions which are better than the three traditional policies on single objectives, even if only slightly, and especially on tradeoffs among more of them. About the latter case, the objective values of the solutions found are often singularly very near to the best ones in traditional policies, especially using the Pareto front (see the last row of Table 7). Hence, we found policies that almost join best performance of traditional ones.

To continue this project, one could try to search for better solutions modifying primitive set, extending the research space to more complex functions.

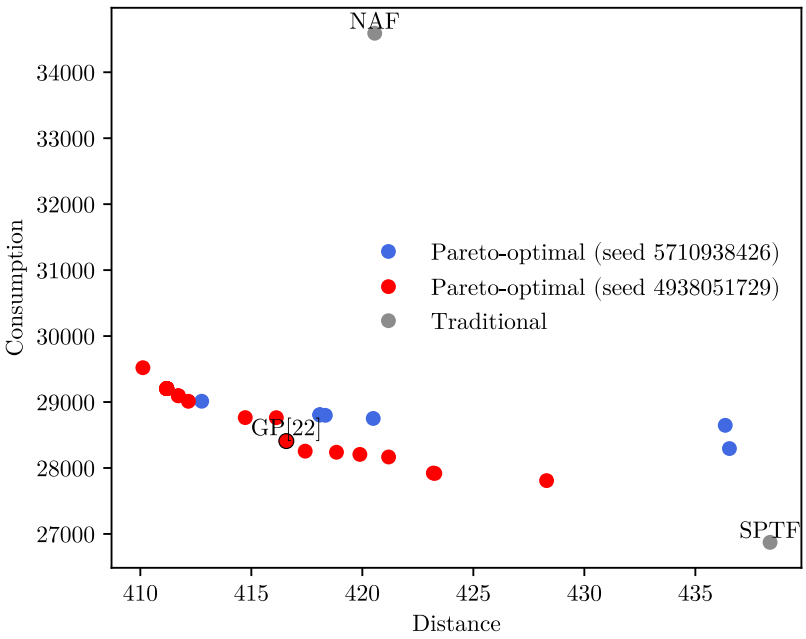

Figure 4: Validation Pareto fronts obtained from the GP algorithm with **distance and consumption** objectives enabled, compared with points of relevant traditional policies evaluated on the validation set

References

[1] Kalyanmoy Deb. *Multi-Objective Optimization Using Evolutionary Algorithms*. John Wiley & Sons, 2001.

[2] Jinxiang Gu, Marc Goetschalckx, and Leon F. McGinnis. "Research on warehouse operation: A comprehensive review". In: *European Journal of Operational Research* 177.1 (2007), pp. 1–21.

[3] Jairo R. Montoya-Torres et al. "A literature review on the vehicle routing problem with multiple depots". In: *Computers & Industrial Engineering* 79 (2015), pp. 115–129.

[4] F.-A. Fortin et al. "DEAP: Evolutionary Algorithms Made Easy". In: *Journal of Machine Learning Research* 13 (2012), pp. 2171–2175.

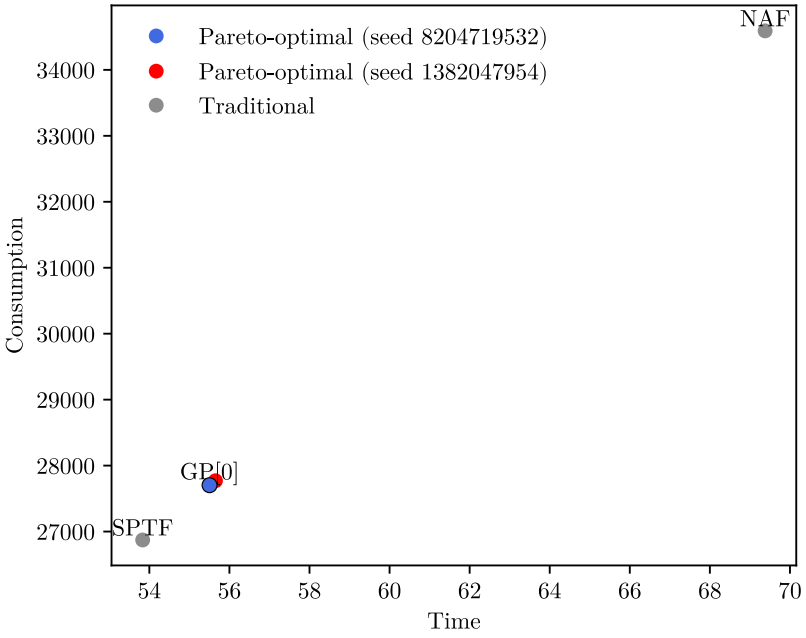

Figure 5: Validation Pareto fronts obtained from the GP algorithm with **time and consumption** objectives enabled, compared with points of relevant traditional policies evaluated on the validation set

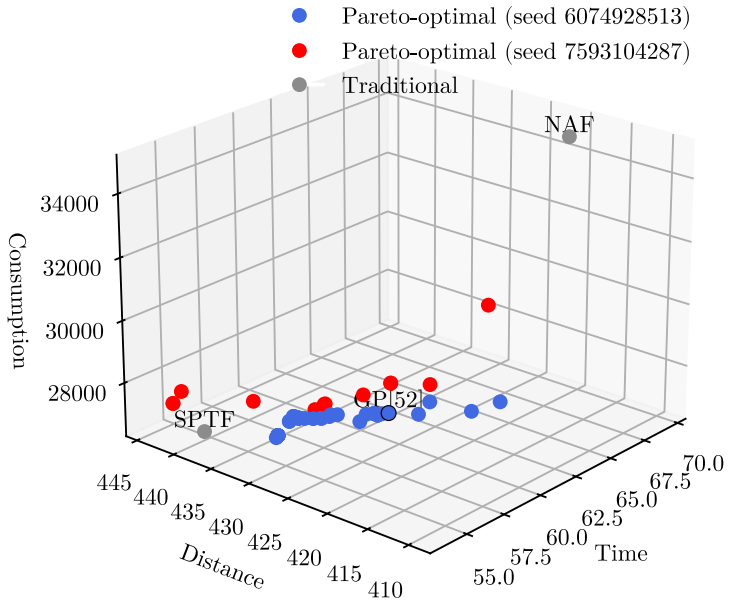

Figure 6: Validation Pareto fronts obtained from the GP algorithm with **all** objectives enabled, compared with points of relevant traditional policies evaluated on the validation set