Keywords: dynamic job shop scheduling problem, dynamic vehicle routing problem, ant colony optimization, container terminal, harbor logistics.

1. Introduction

With the development of trade activities which have continually increased (from less than 6 millions of TEU in 1970 to 500 millions of TEU in 2008[1]), container has become the first mode of packaging for exchanging goods. One of the cheapest way to carry thousands of tons of goods around the world are containers vessels which are able to carry at once ten of thousands containers. As a consequence, multimodal platforms have been created all around the world in order to facilitate the transfer between ships and trucks or trains. In order to reduce the costs of these transferts, the harbor has to provide a high quality of service and unceasingly develop new technologies and processes. The present work focuses on one particular point of the organization of container terminals: the scheduling of straddle carriers missions.

Schematically a container terminal is divided into 5 areas : the yard composed of lanes of stacked containers, the depot where the idle vehicles are parked, the trucks area, the trains area and the quays.

Straddle carriers are used within the terminal to move containers. Those vehicles can lift a container from above and designed for moving containers in the yard by driving over the lanes. They are also used to load or unload trucks or trains. Some of them are able to dynamically adapt the spreader[2] size to any container dimensions while some of them require to be set up in the depot.

Moving a container requires a straddle carrier to pickup, to move from one place to another one and to deliver the container once arrived at destination. This set of handling and moving is called a mission. There are four kinds of missions:

- Incoming container missions;
- Outgoing container missions;
- Transshipment missions;
- Yard container missions.

The first mission type concerns trucks, trains and ships unloading. Straddle carriers drive to the pick-up locations to unload the vehicles, and then lift the container, drive to the yard to stock it. Concerning ships, they are unloaded by quay cranes which stack containers on the quay. Then, straddle carriers come to pick-up the containers and move them to the yard. The second category concerns trucks, trains and ships loading. In this case, straddle carriers start by picking-up a container from the yard and then drive to the delivery location (trucks areas, trains area or ship areas) to deliver it to their recipient. The third category of missions concerns the move of a container from a ship to another one. Finally, the last kind of missions concerns internal yard optimization process. Indeed, in some cases, it can be useful to reorganize part of the

[1]Source: Containerization International (http://www.ci-online.co.uk)
[2]Grip used to grab container

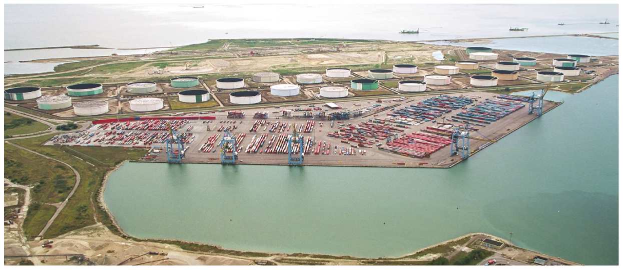

Figure 1: The Terminal of Normandy, Le Havre's harbor, France.

storage area in order to reduce further delivery times or to free strategic container slots for next unloading missions.

Each mission is composed of a pickup phase, a moving phase and a delivery phase. The first and last phases are each constrained by a time window. These time windows correspond to appointments between straddle carriers and customer vehicles (trucks, trains or ships). Straddle carriers have to reach the pickup or delivery location within the given time window. If a straddle carrier comes too early, it will have to wait. On the contrary, if it comes too late, the customer vehicle will have to wait and may require late fees from the container terminal exploitation compagny. Thus, overrunning a time window implies a cost for the container terminal. Note that, in the case of yard optimization missions, time windows can be overrun without entailing penalties or additional costs. So, according to the mission kind, time windows can be hard or soft. For incoming missions, the pickup time window is hard and the delivery time window is soft. For the outgoing container missions, the pickup time window is soft but the delivery time window is hard. For transshipment missions, both time windows are hard, and for yard optimization missions both time windows are soft.

On the terminal, straddle carriers perform missions one after the other with a double objective: respecting missions time windows and minimizing the overall covered distance [2]. Indeed, time windows must be respected to avoid penalty fees and the distance covered by the vehicles directly impacts the exploitation costs of those vehicles. So, part of the problem may be expressed as a static scheduling problem. However, in real life, container terminals are subject to dynamic events occuring in an unpredictible way. New missions are added to the system for answering custoners needs. Some customers may also miss the time windows, some straddle carriers may suddenly become unavailable or some lanes may be closed temporarily. Such dynamic uncertainties contribute to the unpredictible evolution of the environment.

As a consequence, the feasibility of a fully static schedule is unlikely. The work presented in this parper focuses on the allocation and the scheduling of the missions to the straddle carriers. The ant based method proposed aims at providing feasible solutions that are continously reoptimized in response to dynamic events.

The next section presents the modeling of the problem for both static and dynamic formulations and exposes related existing works. Section 3 is devoted to the main contribution of this paper, a swarm intelligence method, based on the ant metaphor able to cope with dynamic events, for generating relevant schedules.

Section 4 presents and discusses results obtained by the method on both static and dynamic versions of the problem. A final section concludes this work and opens some perspectives.

2. Problem Modeling and Related Works

In the static as well as in the dynamic version, the problem is considered for a fixed and limited period of time that will be called "day", for simplicity, in the sequel. It has to be noticed that on most containers terminals the activity is an ever-going process (24 hours a day, 7 days a week) justifying and motivating the design and implementation of optimization processes able to cope with dynamic events.

2.1. Problem modeling

In its static version, we assume that no dynamic event can occur during the day. The number of straddle carriers is supposed to be known and fixed for the whole day. This implies that, once computed a schedule remains valid. As additional hypothesis we suppose that the travel time between two locations within the container terminal is known and does not vary during the day. Moreover, we also assume that time windows are respected by the customers for both pickup and delivery. Finally all missions that are planned during the day are supposed to be known a priori.

According to the introduction, the problem may be formulated as a job shop scheduling problem (JSSP) in which the machines are the straddle carriers and the missions are the jobs.

The problem consists in finding the schedule S of n jobs J_i ($i = 1, \dots, n$) on m machines M_j ($j = 1, \dots, m$). This schedule is represented by the workload W_{M_j} of each machine M_j. The workload W_{M_j} is an ordered list of jobs allocated to the machine M_j. Each job is made of two operations: O_1: pickup of the container and O_2: delivery of the container. As O_1 and O_2 operate on the same container, they have to be processed on the same machine (the straddle carrier) in this order. Moreover, between the execution of both operations, the vehicle has to move from one place (the pickup location) to another place (the delivery location) and this movement entails a delay between O_1 and O_2 related to the distance between both locations.

$$\begin{cases} S = \{W_1, W_2 \dots, W_m\} & \text{and} \\ W_i = \{J_{\alpha_1}, \dots, J_{\alpha_k}\} & \text{with } k \leq n, \text{ and } W_i \cap W_j = \emptyset, \, \forall i \neq j \end{cases}$$

For starting a mission, the straddle carrier has to move to the pickup location. This move can be considered as a setup time or cost. This setup time directly depends on the location of the straddle carrier at the end of the previous mission, if any. If a previous mission was executed by the straddle carrier, then the distance used for computing the setup time is equal to the distance between the delivery location of the previous mission and the pickup location of the current one. If no mission was executed before, then the straddle carrier is supposed to be located at the depot. As a consequence, setup times are sequence dependent (ST_{sd}).

Let us consider two jobs, J_i and J_j each composed of two operations, O_1^i, O_2^i and O_1^j, O_2^j. Each operation has to be performed on a precise location on the terminal. Let us denote by $l(O_p^k)$ the location of the operation p ($p \in \{1, 2\}$) of job J_k. Then, if a machine is allocated both jobs in the order (J_i, J_j), then the setup cost $setup_{i,j}$ is proportional to $d(l(O_2^i), l(O_1^j))$ where $d(l, l')$ denotes the shortest path between the locations l and l'. As the machines are supposed to move at the same speed, we consider that the travel time (setup cost) for chaining job J_i to job J_j on the same machine is equal to $t(d(l(O_2^i), l(O_1^j)))$. Note that, if the machine performs job J_i coming from the depot, the setup is proportional to: $d(\text{depot}, l(O_1^i))$.

2.1.1. Other assumptions

In all generality, the machines may be heterogeneous, depending on their spreader size and adaptability, such that a job J_j may not be compatible with machine M_i. For instance, if the straddle carrier has a spreader only adapted to 40 feet containers, then it will not be able to process 20 feet containers missions. However, such cases have not be considered in the current work. In addition in the static form of the problem and for sake of simplicity, we consider that the speed of all the machines is equal.

No preemption is allowed in the straddle carrier mission scheduling problem. Preemption of a mission can only be considered during the movement phase. Indeed, both pickup and delivery are atomic operations. Thus, preempting a mission is equivalent to stopping the vehicle along the way from the pickup location to the delivery location. And stopping the mission entails an "intermediate delivery" of the container on a temporary location, followed later by picking up again the same container from that location to the final delivery one. Thus, preempting a mission can be considered as the transformation of one mission into two new missions. While such a situation is sometimes observed in real life, we assume that no mission can be split into several ones in the context of the present work.

The jobs are independent in the problem, there is no precedence constraint between them. Though, operations are constrained by time windows. So, if two jobs are scheduled successively without taking into account their time windows, those time windows can be exceeded. In the case of two missions concerning the same delivery slot in the container terminal, if the container of the second mission must be stacked onto the container of the first mission, then the first job should be achieved before the second one. But if the second job is processed before the first one, then a new mission will be added in the pool of jobs to switch the two containers. Again, in the context of the present work, such situations are ignored.

2.1.2. Optimization criteria

Two main criteria are taken into account for optimizing the scheduling of straddle carriers missions on container terminals: the distance traveled by all the straddle carriers, and the respect of the constraints induced by jobs pickup and delivery time windows.

The distance covered by the machines depends on both the distance between the pickup and the delivery places and also the distance between the location of the end of a job to the location of the beginning of the next one. Thus, for each sequence of jobs (J_i, J_j) performed by a machine, the covered distance is equal to $d(l(O_2^i), l(O_1^i))$ plus the distance $d(l(O_1^j), l(O_2^j))$. If J_i is the first job executed by the machine, then the distance is equal to $d(l(\text{depot}), l(O_1^i))$ and if the job J_j is the last job executed then an additional distance should be added to the total distance covered by the machine: $d(l(O_2^j), l(\text{depot}))$.

The second criterion is related to time windows associated to job operations. If a mission misses one or both of its time windows, then there is tardiness. Total tardiness for a job J_i allocated to machine M_k is the

Missions	**Pickup TW**		**Delivery TW**	
M_1	00 : 01 : 09	00 : 03 : 17	00 : 03 : 52	00 : 06 : 00
M_2	00 : 01 : 32	00 : 04 : 10	00 : 04 : 35	00 : 06 : 47
M_3	00 : 07 : 10	00 : 09 : 52	00 : 09 : 14	00 : 12 : 20

Time Windows ($hh : mm : ss$)

Vehicle	**AVG Speed (km/h)**
V_1	20
V_2	25

Vehicles speed (km/h)

Origin	**Destination**			
	$Depot$	M_1	M_2	M_3
$Depot$	0	173	334	328
M_1	347	306	642	636
M_2	344	317	413	407
M_3	348	399	399	396

Distance in meters (read: $Depot \rightarrow M_1 : 173m$)

Origin	**Destination**							
	$Depot$		M_1		M_2		M_3	
	V_1	V_2	V_1	V_2	V_1	V_2	V_1	V_2
$Depot$	00 : 00 : 00	00 : 00 : 00	00 : 00 : 31	00 : 00 : 25	00 : 01 : 00	00 : 00 : 48	00 : 00 : 59	00 : 00 : 47
M_1	00 : 01 : 02	00 : 00 : 50	00 : 00 : 55	00 : 00 : 44	00 : 01 : 56	00 : 01 : 32	00 : 01 : 54	00 : 01 : 32
M_2	00 : 01 : 02	00 : 00 : 50	00 : 00 : 57	00 : 00 : 46	00 : 01 : 14	00 : 00 : 59	00 : 01 : 13	00 : 00 : 59
M_3	00 : 01 : 03	00 : 00 : 50	00 : 01 : 12	00 : 00 : 57	00 : 01 : 12	00 : 00 : 57	00 : 01 : 11	00 : 00 : 57

Travel times according to vehicle (read: $Depot \rightarrow M_1 for V_1 : 31s$)

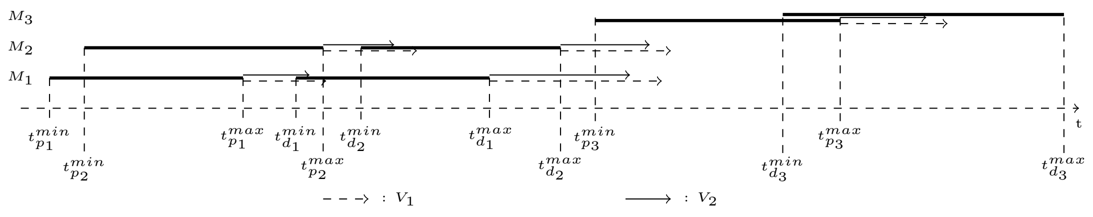

Figure 2: Example of a problem with 2 machines (V_1 and V_2) and 3 jobs (M_1, M_2 and M_3).

sum of pickup tardiness T^k_{i,O_1} and delivery tardiness T^k_{i,O_2}. The pickup tardiness is the difference between the arrival time of M_k at the pickup location of J_i and the end of the pickup time window of J_i. The delivery tardiness is the difference between the arrival time of M_k at the delivery location of J_i and the end of the delivery time window of J_i.

Let us consider that machine M_k executes the sequence (J_i, J_j), and that the delivery operation of job J_i, performed by M_k, ends at C^k_i. We also denote the time needed to travel from location l to l', $t(d(l, l'))$ (as for the setup times). We also denote by $tw_{min}(O^i_1)$ (resp. $tw_{max}(O^j_2)$) the starting date (resp. ending date) of the time window of the pickup (resp. delivery) operation of job J_i (resp. J_j). Then, the tardiness for job J_j, executed by M_k, may be expressed as follows:

$$\begin{cases} T^k_{j,O_1} = max(0, (C^k_i + t(d(l(O^i_2), l(O^j_1)))) - tw_{max}(O^j_1)) = max(0, C^k_i + setup_{i,j} - tw_{max}(O^j_1)) \\ T^k_{j,O_2} = max(0, C^k_j - tw_{max}(O^j_2)) \\ T^k_j = T^k_{j,O_1} + T^k_{j,O_2} \end{cases}$$

The tardiness may entail some penalty costs to the container terminal. Depending on the kind of mission, these costs may be more or less important. This can be represented by weighted tardiness. For summarizing, the optimization aims at minimizing two criteria, the total distance covered by the vehicles and the total weighted tardiness.

If we consider n jobs, and if we denote $M(i)$ the machine performing job J_i, the tardiness function to minimize can be expressed as follows:

$$F_S = \sum_{j=1}^{n} (w_j . T_j^{M(j)})$$

The total distance covered by the machine M_i is equal to:

$$D_i = d(l(\text{depot}), l(O_1^1)) + d(l(O_1^1), l(O_2^1)) + \sum_{k=2}^{card(W_i)} \left(d(l(O_2^{k-1}), l(O_1^k)) + d(l(O_1^k), l(O_2^k)) \right) + d(l(O_2^{card(W_i)}), l(\text{depot}))$$

where $card(W_i)$ denotes the number of jobs in the workload of the machine i.

The total distance covered by all the machines can be expressed as:

$$D_S = \left(\sum_{i=1}^{m} D_i \right)$$

Since the allocation of the jobs to the machines influence the distance covered by the vehicles, the criteria F_S and D_S are not independent. The algorithm we propose in the next section uses a weighted combination of those criteria.

2.1.3. Related works

It should be noticed first that if we relax the time windows constraints, the problem may be formulated in the following terms: given a set of tasks, such that each task is composed of a pickup operation, a movement between two places (a source and a destination) and a delivery operation, how to allocate the tasks to vehicles such as to minimize the covered distance?

Since the problem is about finding shortest routes for a fleet of vehicles, it seems natural to look at existing works dealing with Vehicle Routing Problems (VRP) [29, 13] and more precisely dealing with Pickup and Delivery Problem (PDP) [3]. This latter version of VRP consists in picking up goods before delivering them to the customers. A variant of PDP takes into account the time windows [19]. Here, the pickup and the deliveries must occur into given time intervals. There is also a version of these problems with vehicles characterized limited capacities [29].

In those problems each vehicle usually has a fixed capacity greater than one and the problem is to find shortest paths to deliver the goods to the customers. In our problem, each vehicle has a unit capacity. This means that only one delivery per pickup operation is possible. Moreover, in our problem a given customer requires only one type of good, located at a given fixed place, such that no choice is possible for the pickup location and the delivery one. Our main issue is related to the way the missions are chained one after the other on each vehicle, and a solution of this problem is highly constrained by the pickup and delivery time windows. Thus, our problem seems to be more related to job shop scheduling than to vehicle routing.

For the Job Shop Scheduling Problem (JSSP) a set of n jobs is given as well as a set of m machines. Each Job J_i consists of a sequence of k_i operations that have to be performed in this order on different machines.

Each operation is characterized by a processing time that may depend on the machine it is allocated to. In addition, a machine can process only one operation at a time and a given operation can only be processed by a single machine at a time.

The literature about the JSSP is huge and we invite the reader to refer to the work of Jain et al. [15] for a general and comprehensive survey about this problem. The short state of the art presented in the sequel focuses specifically on ant based methods dealing with JSSP since such an approach has been chosen for addressing the current problem.

In [15] and [9], the authors give a survey of Job Shop Scheduling Problems and of the methods used to solve the different sub-problems. Among the different methods used for solving the problem, an ant system algorithm has first been introduced by Colorni et al. [10]. The authors used a complete graph representing the possible schedules and tabu lists to mark the nodes (jobs) chosen by the ants. When all the jobs are in the tabu list, the solution represented by the list is evaluated by a fitness function. The authors concluded that Ant Colony algorithms are one of the most easily adaptable population based algorithm. More recently, Apinanthana et al. extended the use of Ant Colony based algorithm to solve the Multi-criteria Job Shop Scheduling Problem [30]. In [14], the authors proposed a corresponding algorithm for the Job Shop Scheduling Problem with Time Windows (JSSP-TW). They showed that Ant Colony Optimization is both efficient and effective for this class of optimization problems.

In the Flexible Job Shop Scheduling Problem (FJSSP), all the machines are able to process any operation. This version of the Job Shop Scheduling problem has been first introduced by Brucker and Schlie in [8]. They proposed a graphical methods for solving this problem with two jobs. Since the problem is NP-Hard for more than two jobs, meta-heuristics have been used. In [24], Ponnambalam et al. introduced an Ant Colony Optimization algorithm to solve large instances of the FJSSP up to 20 jobs and 15 machines. In [34], Xing et al. used a hybrid optimization method to solve the FJSSP using both knowledge and Ant Colony Optimization models. The algorithm memorizes good features of past iterations to guide the ACO algorithm toward best solutions.

In [1] the authors give a survey of algorithms used to solve Job Shop Scheduling Problems with Sequence Dependent Setup Times (JSSP-SDST). Many different methods have been used for that problem: hybrid genetic algorithms, disjunctive graphs, mixed integer linear programming model with local search scheme, fast tabu search, branch and bound, dynamic programming, polynomial insertion algorithm, Lagrangian relaxation or ant colony algorithms. In [28], Taillard developped a tabu search algorithm for the JSSP. This algorithm has been improved along the time ([7, 32]) and is known to be one of the most effective technique to solve the JSSP. Simulated Annealing (SA) or Genetic Algorithms (GA) are also widely spread to generate solutions to the JSSP.

However most of these methods can only been applied to a static version of the JSSP-SDST problem.

2.2. Dynamic version of the problem

In its dynamic version, the straddle carriers scheduling problem takes into account dynamic events. It implies that a computed schedule can become deprecated during the day. Many different dynamic events may occur during the execution of the schedule. According to [23], dynamic events are either resource or job related. For instance, the travel times between each location within the container terminal, which are known a priori, may vary during the day. The number of straddle carriers, known at the beginning of the day, can change since some of them can become unavailable for whatever reason. Time windows can also be subject to some modifications if, for instance, a truck, or a ship, is in late for meteorological reasons. And last, but not least, some new missions may arrive at any moment of the day, without prior notice.

Some of these situations have already been studied. For instance, Berbeglia et al. described the dynamic version of pickup and delivery problems in [4]. However, in their work they do not consider time windows. This temporal constraint is present in the work of S. Mitrovic-Minic [20]. She addressed the problem of Dynamic Pickup and Delivery with Time Windows (DPDP-TW). In [21], S. Mitrovic-Minic and her colleagues used a multiple Traveler Salesman Problem (m-TSP) formulation of the Vehicle Routing Problem with Time Windows (VRP-TW). They used precedence graphs to model the multiple Traveler Salesman Problem with Time Windows (m-TSPTW). They proposed algorithms to compute bounds on the number of vehicles required to complete the deliveries. They also showed that this problem is NP-Hard.

The problem of Job Shop Scheduling, when new jobs must be inserted into a pre-computed schedule, is known as the Dynamic Job Shop Scheduling Problem (DJSSP) [26].

In [23], the authors distinguish four different approaches to the dynamic version of the problem: completely reactive scheduling, predictive reactive scheduling, robust predictive-reactive scheduling, or robust pro-active scheduling. Suresh and Chaudhuri, in [27], regroup theses approaches in two classes: predictive and reactive scheduling. In reactive scheduling, the schedule is computed according to the events. The most common strategy is the predictive and reactive scheduling where a schedule is first computed, and then updated according to the events. On the other hand, in predictive scheduling, the schedule is computed in the way that it facilitates the handling of possible dynamic events.

Each of these strategies requires rescheduling. According to Oulhadj and Petrovic in [23], three different policies can be found in the literature: the periodic rescheduling, the event driven rescheduling, and a hybrid rescheduling which is a combination of the two other policies. The authors explain that the periodic policy can be sufficient to deal with the dynamic events, but that the size of the period should be determined carefully. Besides, the authors concluded that in every studyied paper of their survey, the event driven policy gives better results than the periodic policy.

Since the objective of the reschedule is to take into account the new characteristics of the problem, two strategies can be used: either reschedule from scratch or repair the previous computed schedule [23]. The first strategy appears to be unusable in practice because of the lack of time to compute the whole new schedule.

As for dynamic vehicle routing problems, exact algorithms can not be used anymore to compute the schedule, due to the lack of time between jobs arrivals. Several studies have been carried on heuristics and meta-heuristics methods, AI based techniques and multi-agent approaches to solve the DJSSP. In [33], Gere introduced heuristics for both JSSP and DJSSP. He tested the efficiency of each heuristic and showed that the choice of the good heuristic is more important for solving the problem than the choice of a priority rule for the jobs. He concluded that heuritics based on anticipation of the schedule improved significantly the quality of the solution while only slightly raising the computing time. Artificial Intelligence based methods are also used for DJSSP, such as knowledge based algorithms, neural networks, case based reasoning, fuzzy logic or petri nets. More recent approach using Multi-Agent Systems (MAS) to provide solutions has been developed. In [35], Yoo and Müller introduced a MAS using a simulated annealing algorithm to optimize the solutions proposed by the agents. Regarding meta-heuristics, they represent the most common way to provide solutions for DJSSP. Genetic Algorithms are one of the most used meta-heuristics to solve the dynamic version of the problem ([18, 25]). Despites the use of Ant Colony algorithms for solving the static versions of Vehicle Routing and Job Shop Scheduling problems, only a few researches have been applied to the dynamic versions of these problems. Vogel et al. proposed a continously operating Ant algorithm to solve the Real World Shop Floor Scheduling Problem [31]. They concluded that their algorithm does not achieve the quality of the Genetical Algorithm solution but that it will lead further research on Ant algorithms toward better results. In [22], Montemanni et al. introduced an Ant Colony System (ACS) for solving the Dynamic Vehicle Routing Problem. They applied their algorithm to a realistic test case study of fuel distribution in the city of Lugano in Switzerland. They showed that their ACS provides good solutions on both theorical and real problems. More recently, Zhou et al. applied an Ant Colony Optimization algorithm to the Dynamic Job Shop Scheduling Problem [36]. They perform their algorithm on two problems having the same jobs to complete but different degree of dynamism. They showed that the algorithm gives good results in both cases.

We propose in the next section an on-line ant colony based algorithm able to propose near optimal solutions at any time for the problem in its dynamic version. The present work is motivated by the lack of studies about the DJSSP with time windows and sequence dependent setup time. Moreover, we aim at minimizing the total distance covered by the machines and the tardiness where most of the algorithms in the literature focus on minimizing the makespan.

3. Colored Ants Algorithm

Bio-inspired algorithms have been more and more used for providing solutions for hard optimization problems since the early nineties [12]. Ant-based approaches (Ant System, Ant Colony Optimisation, Ant Colony System), take advantage of some properties and characteristics that have been discovered by entomologists and sometimes have been reproduced through experiments, as it was done by Deneubourg in 1983 [11], that showed stigmergy between ants. Ant-based or stigmergic systems are particulary well adapted to dynamic optimization problems because of some of their intrinsic characteristics like: decentralized control, indirect communications (entailing positive feedback), and the possibility of forgetting outdated solutions thanks to a bio-inspired mechanism: the evaporation of the pheromone tracks (entailing a negative feedback).

Our method relies on multiple colonies of ants. In our version, ants are attracted by pheromones of their own colony and repulsed by pheromones of foreign colonies. This mechanism results in a collaboration between the ants of a colony and a competition between the different colonies. The idea consists in making them compete for having access to jobs that are compatible, in terms of time windows.

A similar approach has been designed and implemented by Bertelle et al. in [5, 6] for determining dynamically the best distribution of a parallel program on a network. They used a collaboration/competition process between colonies of artificial ants to distribute the data and the computations of a program among heterogeneous processing ressources while minimizing the communication overhead between computing ressources.

The authors used a threshold to avoid ants from choosing a destination containing too much ants entailing an implicit mechanism of repartition of the ants over the search space. Instead of using a threshold that might be difficult to tune correctly, we will see below that we choose a linear combination of the repulsion aspect in the pseudo-random-proportional transition rule.

Let us now describe our graph structure and how our ant-based algorithm operates on this structure and modifies it for building solutions.

3.1. Ants and graph dynamics

For modeling the problem we choose to take into account all the elements of the problem, the missions with their constraints (time windows) and characteristics (locations) and ressources characteristics (speed).

3.1.1. General graph description

Let us define a dynamic directed graph $G^{(t)} = (V, A)$ where V is the set of vertices and A the set of arcs. Initially, each job J_i to be scheduled is modeled as a node $v_i \in V$ and the arcs $a_{ij} = (v_i, v_j) \in A$ represents the possibility for the machines to chain up the job J_i with the job J_j without overrunning the time windows. Thus, an arc is added between two vertices v_i and v_j if and only if: $tw_{max}(O_2^i) + t(d(l(O_2^i), (O_1^j))) \leq tw_{min}(O_1^j)$, where $t(d(l(O_2^i), (O_1^j)))$ denotes the minimum time needed to reach $l(O_1^j)$ from $l(O_2^i)$. During the resolution process, as soon as some missions are executed by straddle carriers, the corresponding vertices (jobs) are removed from the graph and this graph is updated so as to reconnect nodes without predecessors to v_{source} (as it will be explained later), making the graph dynamic.

3.1.2. Vertices

In addition to the jobs, two dummy nodes are added to the graph: a source node v_{source} , and a sink node v_{sink}. The source node is connected to each node of the graph which has an in-degree equals to zero, while the sink node is linked to each node with an out-degree equals to zero.

3.1.3. Colonies and vehicles

In our model, a colony represents a vehicle (machine). The vehicle has to choose the best sequence of missions to process, the one that both minimizes weighted tardiness and the covered distance. Thus, sequences of missions on different vehicules are modeled as distinct paths in the graph. The ants of the colony have to colonize the mission graph in order to find paths in the graph such as to optimize the criteria. When an ant moves to a node (corresponding to a mission), it drops pheromones on it. This pheromone will be used to guide other ants of the colony toward the node and to repulse ants of other colonies to other nodes. As each colony has the same behavior, the nodes are partitioned between the different colonies and this distribution tends to minimize the overall covered distance of the vehicles.

Each colony is modeled by a color. In this way, each node of the graph is colored by the color of the highest amount of pheromone on this node. The solution is obtained by constructing the best paths for each color. Each path P_c of color c is built by starting at v_{source}. During the building of the path, from a vertex v_i of color c, the next node is chosen among the set of successors of v_i colored in c: S_i^c. The chosen node is the one with the highest level of pheromone of color c. The process is repeated until either v_{sink} has been reached or $S_i^c = \emptyset$.

3.1.4. Weights on arcs

Let us consider an arc $a_{ij} = (v_i, v_j)$. This arc is weighted by the cost of sequencing, on the same machine, the missions corresponding to vertices v_i and v_j. This weight must take into account, on the one hand, the travel time between the delivery location of job J_i, $l(O_2^i)$, and $l(O_1^j)$, and on the other hand, the potential tardiness that would result from chaining the two missions on the same machine. In case of heterogeneous machines, each arc should contain as many weights as there are machines in the problem, since travel times may differ from one machine to another if their speeds are different. In all generality, the weight associated to an arc (v_i, v_j) depends on the date t and on the machine c[3] chaining the corresponding jobs (J_i and J_j) and is equal to:

$$w^{(t)}(v_i, v_j, c) = t(v_i, v_j, c) * F_1 + tardiness^{(t)}(v_i, v_j, c) * F_2$$

where $t(v_i, v_j, c)$ represents the time required by machine c to move from $l(O_2^i)$ to $l(O_1^j)$, and F_1 and F_2 are constants used to balance the relative importance of the two criteria: travel time and tardiness.

For computing the weights, we may distinguish three cases according to the extremities of the arc.

In the first case, $v_i = v_{source}$, in the second case, $v_j = v_{sink}$ and in the last case neither v_i nor v_j is the source vertex or the sink one.

In the first scenario the machine is located at the depot and is available for processing a mission. Let us suppose that the mission is represented by node v_j. In that case, the weight associated to the arc (v_{source}, v_j) is given by the previous formula.

For the arcs linking a node v_i to v_{sink}, the weight corresponds to the travel time between $l(O_2^i)$ and $l(depot)$.

For the last scenario, let us consider a node v_j corresponding to a mission allocated to a given machine. Let us suppose that v_j is a successor of v_{source} and a predecessor of v_k. As soon as the mission begun on the machine, v_j is removed from the graph and a new arc (v_{source}, v_k) is added to the graph. The weight of this arc is computed according to the expected date of the end of J_j plus the time required by the straddle carrier to move from $l(O_2^j)$ to $l(O_1^k)$.

The graph represents the possibilities of job scheduling for the machines. The allocation problem is next solved using colored ants colonies algorithm on this graph.

To reduce the size of the search space, when a node is being connected into the graph, the connexity with other nodes is analyzed. Indeed, we try to avoid redundant arcs as in the case $(a, b), (a, c)$ and (b, c), in that case the arc (a, c) will be removed.

[3]Remember that the machine is equivalent to the colony and the color in the proposed method

3.1.5. Back to the ant algorithm

Remember that a vehicle is associated to each colony. We denote c the colony, the vehicle but also the color associated to the colony. The weight $w^{(t)}(v_j, v_k, c)$ represents the travel cost of an ant of the colony c for going from node v_j to node v_k at time t. It corresponds to the weight of the edge linking v_j and v_k for the colony c. We choose to compute it as the sum of two terms: the travel time $t(v_j, v_k, c)$ between $l(O_2^j)$ and $l(O_1^k)$ for the straddle carrier c and the potential tardiness $tardiness^{(t)}(v_j, v_k, c)$ at time t on the pickup time window of mission j after executing mission k. F_1 and F_2 are constants used to balance the relative importance of the two criteria: travel time and tardiness.

$$w^{(t)}(v_j, v_k, c) = t(v_j, v_k, c) * F_1 + tardiness^{(t)}(v_j, v_k, c) * F_2$$

3.2. Dynamic events

In the dynamic version of the problem, missions can be added, removed or updated and the graph has to evolve according to these modifications. Moreover, the set of vehicles may also be subject to changes.

When a new mission is added, a new node, v_{new} is created in the graph. New arcs connecting v_{new} to some other vertices are created following the rules described in the previous section. If the insertion of v_{new} in the graph entails the creation of two arcs (v_{source}, v_{new}) and (v_{new}, v_j) and if the arc (v_{source}, v_j) already existed, then this latter has to be removed. The same with the sink node. This process is required to force the vehicles to process all the missions. Otherwise, the best solution found by the algorithm would be not to process any job because the covered distance would be null.

The second scenario corresponds to the cancelation or the completion of a mission. In such a case, the corresponding vertex has to be removed from the graph as well. Its incident arcs are also removed from the graph, but new arcs should be added to the graph in order to connect its predecessors with some of its successors, according to the rules already presented previously.

When a mission is updated, the corresponding node is deleted from the graph as well as its incident arcs. It is then inserted again and new arcs are also added to the graph following always the same rules.

When a vehicle starts a mission, it must complete the mission unless it breaks down. In this case, the mission is updated because the pickup location may have changed if the vehicle started to move the container before broking down. Moreover, if the vehicle becomes unavailable, the ants of the corresponding colony are reset to the source node and must remain at this node until the vehicle becomes available again. In the meantime, the evaporation process makes the previous allocation solution disappear.

If the vehicle does not broke down, it must achieve the mission. To represent this constraint in the algorithm, the ants of the colony of the vehicle start their path finding from the current mission node. The pheromone of other colonies on this node are removed to make this node unreachable to the ants of other colonies.

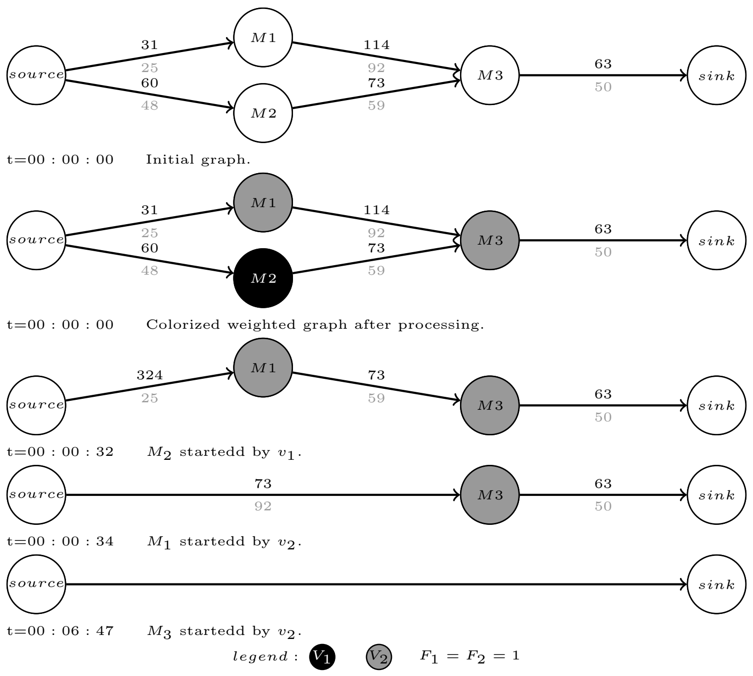
t=00 : 00 : 00 Initial graph.
t=00 : 00 : 00 Colorized weighted graph after processing.
t=00 : 00 : 32 M_2 startedd by v_1.
t=00 : 00 : 34 M_1 startedd by v_2.
t=00 : 06 : 47 M_3 startedd by v_2.
legend : V_1 V_2 $F_1 = F_2 = 1$
source
sink
M1
M2
M3

Figure 3: Dynamic graph of the previous example (see 2.1.1). For the first graph, the values are obtained according to the formula described in the text. For instance the weight of the arc from v_{source} and M_1 is obtained by looking at the numbers given in figure 2.1.1. For vehicle V_1, speed is $20km.h^{-1}$, the distance between the depot and the pickup location of M_1 is $173m$. No tardiness and $F_1 = F_2 = 1$, thus the value is equal to: $(0.173/20) * 3600 \simeq 31s$

3.3. Algorithm

In the sequel c denotes either the colony or the color, which is equivalent in the context of our work, and $\mathcal{C}$ denotes the set of colors (or colonies). Given a node n, n_{out} denotes the set of its successors in the graph.

After each update on the graph, Algorithm 1 is executed Θ times.

3.4. Pheromone handling

The quantity of pheromone of color c on the node j at time t is noted $\tau^t(j,c)$. The quantity of pheromone of other colors than c is noted $\hat{\tau}^{(t)}(j,c)$.

$$\hat{\tau}^{(t)}(j,c) = \left(\sum_{q \in \mathcal{C} \setminus \{c\}} (\tau^{(t)}(j,q)) \right)$$

3.4.1. Positive feedback

The algorithm *spread_pheromone*(Colony c, Node j, Node k) computes the quantity of pheromone $\Delta^{(t)}(k,c)$ of color c which will be dropped on the node k between the time t and $t+1$. This quantity directly depends on the ants previous location j because it takes $w^{(t)}(j,k,c)$ into account.

$$\Delta^{(t)}(k,c) = \lambda * \frac{1}{w^{(t)}(j,k,c)}$$

Algorithm 1 Colored ant colony based algorithm

```
for all ant $a$ of each colony $c$ do
  Node $destination \leftarrow choose\_destination(a)$
  Node $n \leftarrow location(a)$
  if $destination = null$ then
    $return\_to\_source\_node()$
  else
    $move(a, destination)$
    $spread\_pheromone(c, n, destination)$
  end if
end for
for all node $n$ of the mission graph do
  $evaporation(n)$
end for
for all colony $c$ do
  $compute\_path(c)$
end for
```

3.4.2. Negative feedback

The evaporation process computes the new amount of pheromone $\tau^{(t)}(j,c)$ of color c on node j at time t as below:

$$\tau^{(t)}(j,c) = \rho\tau^{(t-1)}(j,c) + \Delta^{(t)}(j,c)$$

3.5. Pseudo-random-proportional rule

The algorithm *choose_destination*(Ant a) (see Algorithm 1) returns the destination chosen by the ant a of the colony c according to its current location node j and the probability $p^{(t)}(j,k,c)$ to choose the destination node k at time t computed by the following pseudo-random-proportional rule:

$$p^{(t)}(j,k,c) = \frac{\tau^{(t)}(k,c)^{\alpha}\left(\frac{1}{w^{(t)}(j,k,c)}\right)^{\beta}\left(\frac{\hat{\tau}^{(t)}(k,c)}{\sum_{q\in C}\tau^{(t)}(k,q)}\right)^{\gamma}}{\sum_{d\in j_{out}}\left(\tau^{(t)}(d,c)^{\alpha}\left(\frac{1}{w^{(t)}(j,d,c)}\right)^{\beta}\left(\frac{\hat{\tau}^{(t)}(d,c)}{\sum_{q\in C}\tau^{(t)}(d,q)}\right)^{\gamma}\right)}$$

If the chosen node k has a probability $p^{(t)}(j,k,c) < \Delta$ then the choice is refused and the ant goes back to the source node. This process is used to avoid the continous colonization of the graph by all the colonies. Indeed, if there are more vehicles than missions to allocate, then it is not possible to allocate a mission to every vehicle. Moreover, we chose to integrate the competition aspect of the algorithm in the pseudo-random-proportional transition weighted by the γ parameter.

3.6. Anytime schedules

At any moment the schedule computed by the method may be obtained. It consists in $card(\mathcal{C})$ paths of distinct colors. Since the graph is directed and acyclic, each path is built by starting at the source node and by choosing from each accessible node colored by the same color that the path to build, the one with the highest quantity of pheromone. The built ends when the sink node is reached or when there is no accessible node of the color of the path.

3.7. Path reinforcement

When the first mission of a path has been started by a vehicle, a reinforcement quantity of pheromone of the color of the vehicle is spread on the whole path. This process avoid useless changes in the solution while the previous computed path is being used.

Algorithm 2 Pheromone track reinforcement of the started solution path

```
for all Node $n$ in the solution $S$ do
    $\tau^{(t)}(n,c) = \tau^{(t-1)}(n,c) + \Lambda$
end for
```

3.8. Parameters

As described above, the algorithm is set up with the following parameters:

- α: relative importance of the pheromone track in the destination choice;
- β: relative importance of the weight heuristic in the destination choice;
- γ: relative importance of the repulsion process in the destination choice;
- Δ: environment pressure rate. If the chosen destination pheromone rate is less than Δ, then the ant die (start over from the source node);
- η: number of ants per colony;
- λ: fixed quantity of pheromone spread on a destination (with $\lambda > 1$);
- Λ: quantity for reinforcement : quantity of pheromone spread on the whole path when the first mission of the path has been started;
- ρ: rate of pheromone conserved after each evaporation;
- F_1: relative importance of the travel time in the weighting function of arcs;
- F_2: relative importance of the tardiness in the weighting function of arcs;
- Θ: number of ants algorithm iterations executed after each event.

Concerning the ant algorithm parameters, we distinguish two classes: on one hand the ones about the choice of destination for the ants (α, β, γ and Δ), and on the other hand, the ones about the pheromone handling (λ, Λ and ρ).

After testing different values for the parameter η, we decided to fix it to the number of missions in the pool. Indeed, when a mission is added, a new ant is created for each colony compatible with the mission. On the contrary, when a mission is removed from the pool, an ant is removed of each colony.

According to our simulation results, it also appears that the quality of the solution found and also the time required to find this best solution is more strongly connected to the fitness weighting the arcs than to the values of the parameters. The parameters of the first class also have more influence on the quality of the solution and the convergence of the algorithm than the parameters of the second class.

4. Experiments and Results

We developed an algorithm able to handle dynamics and to provide near-optimal solutions in a reasonable computation time to our problem. Since the algorithm had also to be failsafe, it ensures feasible solutions at anytime. In this section we first execute the algorithm on static versions of the scheduling problem. Then we test it on several dynamic scenarios. All of the simulations are executed on D^{2}CTS (Dynamic and Distributed Container Terminal Simulator)[17] with configuration files corresponding to the Terminal of Normandy of Le Havre's harbor. Le Havre's harbor is the biggest harbor of France in container traffic. It is located at the North West cost of France, beside the Channel, sea door between the Atlantic and the North Sea.

4.1. Measure of dynamics

In his PhD thesis [16], A. Larsen proposed two measures to determine how dynamic is an instance of a Vehicle Routing Problem. The first one is the degree of dynamism (DOD) and is calculated as the ratio between the number of dynamical requests and the total number of requests. If $dod = 0$ the problem is static, if $dod = 1$ then the problem is fully dynamic. The main weakness of this measure is that it does not take into account the arrival time of those requests. For this reason, Larsen introduced the effective degree of dynamism by the following formula:

$$edod = \frac{\sum_{i=1}^{\eta_{imm}} \left(\frac{t_i}{T}\right)}{\eta_{tot}}$$

Here η_{imm} are the requests arriving during the day which have to be planned immediately. η_{tot} corresponds to the sum of immediate requests and the requests arrived in advance. The planning horizon starts at 0 and ends at T and t_i is the time the i^{th} immediate request is recieved. This measure takes into account the average of the incoming time of the requests and gives a better appreciation of the dynamics of a scenario. Concerning the Vehicle Routing Problem with Time Windows (VRP-TW), Larsen extended the definition of *edod* by this formula:

$$edod - tw = \frac{1}{\eta_{tot}} \sum_{i=1}^{\eta_{tot}} \left(1 - \frac{r_i}{T}\right)$$

Here, r_i is the reaction time of the task i which is the difference between the latest time the request can starts and t_i which is the incoming time of the i^{th} request.

We will use *dod* and $edod - tw$ to measure the dynamics of our scenarios by setting T to the lower bound of the pickup time window of each mission.

4.2. Measure of performance

As discussed in previous sections, we want to minimize exploitation costs of the container terminal. This is the reason why we measure the performance of our algorithm by two criteria: the overall covered distance of the straddle carriers and the overspent time of the missions time windows. In addition, we also count the number overrun time windows.

4.3. Static case

To measure performance of our algorithm, we developed a Branch-and-Bound procedure to solve the problem in the static case and to obtain optimal solutions. This algorithm is also used to solve the instances of the static problem generated from the dynamic one.

We tested here different scenarios of 5 to 12 missions and each of these scenarios has been performed with variable number of straddle carriers between 2 and 3. The small scale of the scenarios in both jobs and machines dimensions is due, of course, to the impossibility to compute optimal solutions in reasonable time. As we experiment our algorithm in the satic form of the problem $dod = 0$ and there is no dynamic events occuring during the simulations.

Missions	**Vehicles**	**Distance**		**Overspent time**		**Overrun TW**		**Execution time**	
		ACO	B&B	ACO	B&B	ACO	B&B	ACO	B&B
5	2	5073	4987	00:00:00	00:00:00	0	0	00:00:02	00:00:00
5	3	5286	4987	00:00:00	00:00:00	0	0	00:00:04	00:00:00
7	2	7807	7510	00:00:00	00:00:00	0	0	00:00:04	00:00:00
7	3	7659	7510	00:00:00	00:00:00	0	0	00:00:04	00:00:00
10	2	10207	9565	00:33:09	00:03:01	10	6	00:00:08	00:00:06
10	3	10421	9261	00:00:11	00:00:00	2	0	00:00:08	00:01:21
12	2	9336	8994	00:21:11	00:02:35	13	6	00:00:10	00:00:17
12	3	9673	9261	00:00:00	00:00:00	0	0	00:00:17	01:11:17

Table 1: results of the static version of the simulations.

Concerning the static case, the results show that the ant colony algorithm provides near optimal solutions in a short computation time. However, we distinguish two kinds of algorithm behaviors according to the feasability of the scenarios. Indeed, when there is enough straddle carriers to process the missions then the ant colony algorithm provides near optimal solutions. But on the contrary, when there is not enough resources, the ant colony algorithm starts to provide lower quality solutions. This behavior shows the difficulty for the algorithm to determine how to distribute the tardiness on missions.

4.4. Dynamic case

In the dynamic case, we can not have determinist algorithm to compute optimal solution since the characteristics of the problem change all along the simulation. As showed on the static results, the algorithm seems to provide a near optimal solution. With a dynamic background, the algorithm will have to adapt to the evolution of the characteristics of the problem and provide a solution at any time. The event driven reschedule policy has been chosen for these simulations (see 2.2).

We only use dynamics on the arrival of the missions for this experiment. We tested here different scenarios of 10 to 100 missions with variable number of straddle carriers between 2 and 20. We also use different values for $edod - tw$. Indeed, the first series of experiments uses an effective degree of dynamism around 0.45. It means that each mission is known relatively early before the begining of its pickup time window.

Missions	**Vehicles**	*dod*	$edod - tw$	**Distance**	**Overspent T**	**Overrun TW**	**Execution t**
10	2	1.0	0.44	10217	00:12:55	6	00:00:07
10	3	1.0	0.44	8908	00:00:33	2	00:00:08
20	5	1.0	0.43	19508	00:05:15	4	00:00:44
20	7	1.0	0.43	21794	00:01:27	1	00:00:55
30	7	1.0	0.44	34458	00:06:05	3	00:01:56
30	10	1.0	0.44	35290	00:00:00	0	00:03:10
40	7	0.975	0.44	43532	00:04:52	2	00:02:48
40	10	0.975	0.44	43563	00:00:00	0	00:04:59
50	7	0.98	0.46	51855	00:46:37	19	00:05:57
50	10	0.98	0.46	51490	00:00:00	0	00:05:38
75	15	0.973	0.46	76384	00:55:00	1	00:13:43
100	20	0.98	0.46	98697	00:00:10	1	00:31:55

Table 2: results of the dynamic version of the simulations ($edod - tw \simeq 0.45$).

The second series of experiments uses the same missions than the first series but here, the missions are known at the very begining of their pickup time windows. The effective degree of dynamism is 1.0. It represents the worst case for the algorithm because it prevents it from optimizing the linking of the missions. In fact, the missions are colonized by the artificial ants and then removed from the graph. Since ants use only local information, when other missions arrive in the schedule, they can not be linked efficiently to current missions.

Missions	**Vehicles**	*dod*	$edod - tw$	**Distance**	**Overspent T**	**Overrun TW**	**Execution t**
10	2	1.0	1.0	11142	00:23:29	9	00:00:04
10	3	1.0	1.0	11946	00:00:22	2	00:00:04
20	5	1.0	1.0	22177	00:08:20	7	00:00:11
20	7	1.0	1.0	25680	00:00:03	1	00:00:17
30	7	1.0	1.0	37518	00:00:00	0	00:00:20
30	10	1.0	1.0	37565	00:00:00	0	00:00:26
40	7	1.0	1.0	48275	00:00:26	3	00:00:26
40	10	1.0	1.0	49325	00:00:00	0	00:00:37
50	7	1.0	1.0	53730	00:06:11	8	00:00:38
50	10	1.0	1.0	60237	00:00:00	0	00:00:44
75	15	1.0	1.0	84178	00:00:00	0	00:01:36
100	20	1.0	1.0	108099	00:00:00	0	00:02:35

Table 3: results of the fully dynamic version of the simulations ($edod - tw \simeq 1.0$).

We can see with these results that the distance is not near optimal with a high effective degree of dynamism. On the contrary the time windows are mostly respected. So, the results show the robustness of the algorithm wich provide good solutions even in a unknown background.

5. Conclusions and Open Perspectives

We proposed in this paper an ant colony based algorithm to solve a real world application of the job shop scheduling problem. Indeed, the applicated problem concerns the scheduling of the missions on a straddle carrier container terminal. A container terminal is a highly dynamic environment where a lot of entities interact with each others, and its objective is to maintain a high quality of service while minimizing the exploitation costs. The theorical job shop scheduling problem associated with this application has been caracterized this way: $J|ST_{sd}, R_{sd}| \sum w_j.T_j, \sum distance(i)$.

The dynamic graph used to model the problem allows to take into account dynamic events occuring within the container terminal. Moreover, the instrinsic features of the ant colony system provide the required flexibility to adjust the proposed solutions to the changes of the environment.

Our results show that the algorithm provides, in a short computation time, near optimal solutions in low dynamic backgrounds, and proposes good solutions in higly dynamic environments.

Our future work will be focused on improving our model by adding more real life constraints related to the container terminal and the considered model, like mission preemption, precedence relations between missions due to stacking constraints, etc. The Ant Colony approach that we use seems flexible enough to integrate them easily.

[1] Ali Allahverdi, C.T. Ng, T.C.E. Cheng, and Mikhail Y. Kovalyov. A survey of scheduling problems with setup times or costs. *European Journal of Operational Research*, 187(3):985–1032, June 2008.

[2] Stefan Balev, Frédéric Guinand, Gaëtan Lesauvage, and D. Olivier. Dynamical Handling of Straddle Carriers Activities on a Container Terminal in Uncertain Environment - A Swarm Intelligence approach -. In *ICCSA 2009 The 3rd International Conference on Complex Systems and Applications*, volume 2, page 290, Le Havre, France, June 2009. 7p.

[3] Gerardo Berbeglia, Jean-François Cordeau, Irina Gribkovskaia, and Gilbert Laporte. Static pickup and delivery problems: a classification scheme and survey. *Top*, 15(1):1–31, 2007.

[4] Gerardo Berbeglia, Jean-François Cordeau, and Gilbert Laporte. Dynamic pickup and delivery problems. *European Journal of Operational Research*, 202(1):8 – 15, 2010.

[5] Cyrille Bertelle, Antoine Dutot, Frédéric Guinand, and Damien Olivier. Organization detection using emergent computing. *International Transactions on Systems Science and Applications*, 2(1):61–70, 2006.

[6] Cyrille Bertelle, Antoine Dutot, Frédéric Guinand, and Damien Olivier. Organization detection for dynamic load balancing in individual-based simulations. *Multi-Agent and Grid Systems*, 3(1):42, 2007.

[7] Jacek Blazewicz, Wolfgang Domschke, and Erwin Pesch. The job shop scheduling problem: Conventional and new solution techniques. *European Journal of Operational Research*, 93(1):1–33, August 1996.

[8] P. Brucker and R. Schlie. Job-shop scheduling with multi-purpose machines. *Computing*, 45:369–375, 1990. 10.1007/BF02238804.

[9] Peter Brucker. *Scheduling Algorithms.* Springer Publishing Company, Incorporated, 5th edition, 2010.

[10] V. Maniezzo Colorni A., M. Dorigo and M. Trubian. Ant system for job-shop scheduling. *Belgian Journal of Operations Research, Statistics and Computer Science*, 34(1):39–53, 1994.

[11] Jean-Louis Deneubourg, Jacques M. Pasteels, and J. C. Verhaeghe. Probabilistic behaviour in ants: A strategy of errors? *Journal of Theoretical Biology*, 105:259–271, 1983.

[12] Marco Dorigo, Mauro Birattari, and T. Stützle. Ant Colony Optimization: Artificial ants as a computational intelligence technique. *IEEE Computational Intelligence Magazine*, 1(4):28–39, 2006.

[13] Gilbert and Laporte. The vehicle routing problem: An overview of exact and approximate algorithms. *European Journal of Operational Research*, 59(3):345 – 358, 1992.

[14] Rong-Hwa Huang and Chang-Lin Yang. Ant colony system for job shop scheduling with time windows. *The International Journal of Advanced Manufacturing Technology*, 39:151–157, 2008. 10.1007/s00170-007-1203-9.

[15] Anant S. Jain and Sheik Meeran. A state-of-the-art review of job-shop scheduling techniques. *European Journal of Operations Research*, (113):390–434, 1999.

[16] A. Larsen. *The Vehicle Routing Problem.* PhD thesis, Department of Mathematical Modelling, Technical University of Denmark, 2000.

[17] Gaëtan Lesauvage, Stefan Balev, and Frédéric Guinand. D^{2}CTS : A dynamic and distributed container terminal simulator. In *HMS 2011 : The* 13rd *International Conference on Harbor, Maritime & Multimodal Logistics Modelling and Simulation*, September 2011.

[18] Shyh-Chang Lin, Erik D Goodman, and William F Punch. A genetic algorithm approach to dynamic job shop scheduling problems. *The 17th International Workshop on*, pages 126–131, 1997.

[19] S Mitrovic-Minic. Pickup and delivery problem with time windows: A survey. *SFU CMPT TR*, 12(1):1–43, 1998.

[20] S. Mitrovic-Minic. *The dynamic pickup and delivery problem with time windows.* Simon Fraser University, 2001.

[21] Snežana Mitrović-Minić and Ramesh Krishnamurti. The multiple tsp with time windows: vehicle bounds based on precedence graphs. *Operations Research Letters*, 34(1):111 – 120, 2006.

[22] R. Montemanni, L. Gambardella, A. Rizzoli, and A. Donati. Ant colony system for a dynamic vehicle routing problem. *Journal of Combinatorial Optimization*, 10:327–343, 2005. 10.1007/s10878-005-4922-6.

[23] Djamila Ouelhadj and Sanja Petrovic. A survey of dynamic scheduling in manufacturing systems. *J. of Scheduling*, 12(4):417–431, August 2009.

[24] S G Ponnambalam, N Jawahar, and B S Girish. An ant colony optimization algorithm for flexible job shop scheduling problem. *Advanced Science Letters*, 4(Pinedo):2127–2131, 2005.

[25] J. G. Qi, G. R. Burns, and D. K. Harrison. The application of parallel multipopulation genetic algorithms to dynamic job-shop scheduling. *The International Journal of Advanced Manufacturing Technology*, 16:609–615, 2000. 10.1007/s001700070052.

[26] R. Ramasesh. Dynamic job shop scheduling: A survey of simulation research. *Omega*, 18(1):43 – 57, 1990.

[27] V. Suresh and Dipak Chaudhuri. Dynamic scheduling—a survey of research. *International Journal of Production Economics*, 32(1):53 – 63, 1993.

[28] Éric D. Taillard. Parallel taboo search techniques for the job shop scheduling problem. *INFORMS Journal on Computing*, 6(2):108–117, 1994.

[29] Paolo Toth and Daniele Vigo, editors. *The vehicle routing problem.* Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2001.

[30] Apinanthana Udomsakdigool and Voratas Kachitvichyanukul. Ant colony algorithm for multi-criteria job shop scheduling to minimize makespan, mean flow time and mean tardiness. *International Journal of Management Science and Engineering Management*, 6(2):117–123, 2011.

[31] Andre Vogel, Marco Fischer, Hendrik Jaehn, and Tobias Teich. Real-world shop floor scheduling by ant colony optimization. In Marco Dorigo, Gianni Di Caro, and Michael Sampels, editors, *Ant Algorithms*, volume 2463 of *Lecture Notes in Computer Science*, pages 173–198. Springer Berlin / Heidelberg, 2002.

[32] Jean-Paul Watson, Darrell L. Whitley, and Adele E. Howe. A Dynamic Model of Tabu Search for the Job-Shop Scheduling Problem. *Proc. of the 1st Multidisciplinary Int. Conf. on Scheduling: Theory and Applications*, pages 320–336.

[33] Jr. William S. Gere. Heuristics in job shop scheduling. *Management Science*, 13(3):167–190, November 1966.

[34] Li-Ning Xing, Ying-Wu Chen, Peng Wang, Qing-Song Zhao, and Jian Xiong. A knowledge-based ant colony optimization for flexible job shop scheduling problems. *Applied Soft Computing*, 10(3):888 – 896, 2010.

[35] Min-Jung Yoo and Jean-Pierre Müller. Using multi-agent system for dynamic job shop scheduling. In *ICEIS*, pages 517–525, 2002.

[36] Rong Zhou, Heow Pueh Lee, and Andrew Y. C. Nee. Applying ant colony optimization (aco) algorithm to dynamic job shop scheduling problems. *International Journal of Manufacturing Research*, 3(3):301–320, 2008.