Distributed Quay Crane Scheduling with Overlapping Constraints

A distributed constraint optimization approach

I. Zabet* and R.R. Negenborn+
* Vienna University of Technology, Austria
+ Dept. of Marine & Transport Technology, Delft University of Technology, The Netherlands
E-mail: iman.zabett@student.tuwien.ac.at, r.r.negenborn@tudelft.nl

Abstract—**For sea port container terminals, a key objective is to increase the container throughput by minimizing the amount of time necessary to load into and discharge containers from a ship using quay cranes (QCs). In this paper we discuss the situation in which some tasks can be handled by multiple QCs, represented by so-called overlapping area constraints. Overlapping area constraints determing the tasks that more than one QC could take care of. We formulate a distributed QC scheduling problem with overlapping area constraints and cast this problem as a Distributed Constraint Optimization Problem (DCOP). A new negotiation algorithm called Extended Asynchronous BackTracking (E-ABT) is then proposed for solving the DCOP.**

Keywords—Quay Crane Scheduling Problem; Distributed Quay Crane Scheduling; Distributed Constraint Optimization Programming; Extended Asynchronous BackTracking

I. Introduction

Every year millions of TEU (Twenty-foot Equivalent Unit container) are handled by container terminals. Maritime container terminals all over the world are the most important part for transshipment and intermodal container transfers. By the end of 2013 almost 170 million TEU are expected to be handled [1]. Today container ships still become larger and can already carry up to 15000 TEUs [2]. To cope with the rapid growing traffic of international trade, the most important problem will be the reduction of the amount of dwell time and associated transaction cost. The way to do this is to ensure that load/unload processes of containers to/from ship are done as quickly as possible.

In this paper we focus on improving the scheduling of the tasks of quay cranes (QCs). In particular, we consider how performance could be improved if QCs would have overlapping task areas; i.e., if multiple quay cranes would be able to handle particular containers, instead of the situation in which only a single QC can handle a particular container. This will require a new approach to scheduling, in which quay cranes themselves can decide, in coordination with one another, which container to handle next. We recommend that in order to obtain better performance of loading/discharging containers by QCs, a different type of stowage plan is adopted. Nowadays, groups of containers with the same destination are typically considered in a same ship's bay [3]. This limits the number of QCs that can handle these containers. E.g., if a group of containers is placed at the bottom of a hold under other irrelevant containers, a QC handling the bay will have to unload and load (re-stow) them, which takes much time. If we change the stowage plan such that groups of containers along the length of ship are considered instead of along the width, we expect that the number of QCs that can handle them will be higher, lowering the time required and lowering number of re-stowage moves.

We consider a multi-agent system (MAS) approach [4] for representing the decision making in the container terminal. A MAS for a container terminal automation system has been considered before [5], [6]. For such a system, agents are considered for different parts of the terminal. The agents control and optimize their own environment while communicating with others. In the MAS architecture, we consider a three-layered structure (Fig. 1). The higher level is a supervisory system that provides guidelines for agents that interact with each other at the middle layer. These middle layer agents are scheduling, making decisions and controlling movements of QCs at the operational layer. In our case, we focus on the actions and interactions of Quay Crane Agents (QCAs), which handle the scheduling and sequencing of containers from/to berthed ships to/from Automated Guided Vehicles (AGVs) [7]. The QCAs together have to solve the so-called distributed quay crane scheduling problem (D-QCSP) cooperatively with a minimum of data obtained from the supervisory system.

The problem that the agents in the MAS have to solve can be cast as a distributed constraint optimization problem (DCOP). For solving this problem, they have to perform a search through the action space to determine actions that will result in the best performance. Asynchronous BackTracking (ABT) is one of the classic algorithms for solving DCOPs. Distributed search algorithms for MAS work either synchronously or asynchronously [8]. In asynchronous algorithms, such as ABT, the order in which agents make choices is flexible and arbitrary. For such asynchronous algorithms, computational requirements for performing local optimization are typically relatively small; moreover, each step of the algorithm typically does not require global knowledge of the problem. In this paper we propose *extended* asynchronous backtracking (E-ABT) for solving the DCOP. This extension enables the assignment of multiple values for a variable.

This paper is organized as follows. In Section II, we propose our approach E-ABT for solving the distributed QCSP. In

Section III simulation experiments are performed to illustrate the behavior of the proposed distributed approach and to make a comparison with a centralized approach. Section IV provides conclusions and directions for future research.

II. Distributed Quay Crane Scheduling with Overlapping Constraints

A. Distributed constraint optimization

We can formalize a DCOP [9] as a *pentuple* $\langle \mathbb{A}, \mathbb{X}, \mathbb{D}, \mathbb{T}, \mathfrak{j} \rangle$, where $\mathbb{A} = \{a_1, \dots, a_m\}$ is a set of m agents, $\mathbb{X} = \{\mathbb{X}_1, \dots, \mathbb{X}_n\}$ is a set of n sets of physical components per agent where $\mathbb{X}_i = \left\{x_i^1, \dots, x_i^{n_{\mathbb{X}_i}}\right\}$ is the set of components of agent a_i, with $x_i^j \in \mathcal{X}$ a physical component from all physical components $\mathcal{X} = \{x_1, \dots, x_{n_X}\}$. Moreover, $\mathbb{D} = \{\mathbb{D}_1, \dots, \mathbb{D}_n\}$ is set of n sets of doable tasks $\mathbb{D}_i$ per component $x_i \in \mathcal{X}$. $\mathbb{D}_i = \left\{d_i^1, \dots, d_i^{n_{\mathbb{D}_i}}\right\}$ is the ordered set of doable tasks of physical component $x_i \in \mathcal{X}$, where $d_i^j \in \mathfrak{D}$ is the jth task of doable tasks set of physical component $x_i \in \mathcal{X}$ and $\mathfrak{D} = \{\delta_1, \dots, \delta_{n_{\mathfrak{D}}}\}$ is the ordered set of all doable tasks that have to be done. $\mathbb{T} = \{\mathbb{T}_1, \dots, \mathbb{T}_n\}$ is the set of allocated tasks sets and $\mathbb{T}_i = \left\{t_i^1, \dots, t_i^{n_{\mathbb{T}_i}}\right\}$ is the ordered set of allocated tasks for component x_i, where $t_i^j \in \mathbb{D}_i$ is the jth allocated task of component x_i. The general DCOP problem can now be formulated as finding an assignment $\mathbb{T}^*$ that minimizes the overall cost function $\mathfrak{j}(\mathbb{T})$:

$$\mathbb{T}^* = \underset{\mathbb{T}}{arg\,min}\ \mathfrak{j}(\mathbb{T}) \qquad (1)$$

where $\mathfrak{j}(\mathbb{T})$ is overall cost function that can be defined arbitrary with respect to the definition of the problem. In DCOP, the agents must choose the values for the set of variables assigned to each agent, such that (1) is solved.

B. Distributed QCSP as DCOP

We next formulate a distributed QCSP and subsequently cast this formulation into the DCOP framework. The QCSP formulation given next is based on the formulation given in [3] with overlapping constraints. Fig. 1 illustrates a D-QCSP involving 4 QCAs. Each QCA considers 1 physical component (a QC). Each QC can be used to handle a number of tasks. There could be some non-overlapping and some overlapping tasks. The problem is to determine which QCA should address which task at what time in which sequence.

Let $\mathbb{D}_{ij}^{\text{ov}} = \mathbb{D}_i \cap \mathbb{D}_j = \left\{d_{ij}^{\text{ov},1}, \dots, d_{ij}^{\text{ov},n_{\mathbb{D}_{ij}^{\text{ov}}}}\right\}$ be the set of overlapping tasks between component $x_i \in \mathcal{X}, x_j \in \mathcal{X}$, and let $\mathbb{D}_i^{\text{nv}} = \mathbb{D}_i \,/\, \bigcup_{j \in \mathcal{N}_i} \mathbb{D}_{ij}^{\text{ov}}$ be the set of non-overlapping tasks for component $x_i \in \mathcal{X}$. Furthermore, on shipboard, containers are put into several container columns, called holds. We can divide the whole tasks $\mathfrak{D}$ into subsets of holds $\mathbb{D}_h^{\mathbb{H}}$ represents as ordered set of tasks of each hold $\mathbb{H}_h$ and defined as: $\mathbb{D}_h^{\mathbb{H}} = \left\{\delta_1^{\mathbb{H}_h}, \delta_2^{\mathbb{H}_h}, \dots, \delta_{n_{\mathbb{D}_h^{\mathbb{H}}}}^{\mathbb{H}_h}\right\}$ with total number of $n_{\mathbb{D}_h^{\mathbb{H}}}$. $\mathbb{H}_h$ is referred

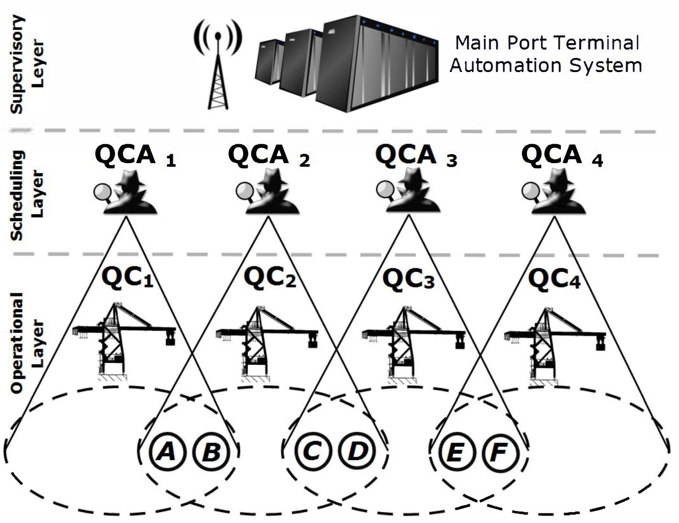

Fig. 1. Illustration of QCAs with only overlapping

to hth hold of containers on board, $h = \{1, \dots, n_{\mathbb{H}}\}$ with total holds of $n_{\mathbb{H}}$. $\delta_i^{\mathbb{H}_h}$ is the ith container of hth hold.

two main objectives regarding the allocation of the overlapping tasks for QCAs can be defined:

1. Minimize differences between completion time of adjacent QCs.
2. Minimize the completion time for every QC.

Hence, we consider the distributed QCSP as a multi-objective scheduling problem. The two objectives can be converted into two terms within a single cost function. We define the overall cost function $\mathfrak{j}(\mathbb{T})$ as:

$$\mathfrak{j}(\mathbb{T}) = \left(\sum_{i=1}^{n-1} \alpha_i^2 \big(\mathfrak{j}_i(\mathbb{T}_i) - \mathfrak{j}_{i+1}(\mathbb{T}_{i+1})\big)^2 + \sum_{i=1}^{n} \beta_i^2 \mathfrak{j}_i^2(\mathbb{T}_i)\right)^{\frac{1}{2}} \qquad (2)$$

where α_i, β_i are weighting coefficients. $\mathfrak{j}_i$ is a cost function associated with component x_i defined in this paper as:

$$\mathfrak{j}_i(\mathbb{T}_i) = \sum_{j=1}^{n_{\mathbb{T}_i}} \gamma_{ij} c_i\big(t_i^j\big) \qquad (3)$$

where γ_{ij} is a weighting coefficient and $c_i\big(t_i^j\big)$ is the cost of doing the jth task of doable set $\mathbb{D}_i$ related to component x_i, defined later in this section. The first term in (2) represents that the differences between costs of subsequent components, i.e. x_i and x_{i+1}, should be minimized in order to be equal tasks among QCs (objective 1). The second term in (2) represents that the value of $\mathfrak{j}_i$ itself is important to us and that we need to minimize it(objective 2). If we compare these terms with the objectives of the QCSP proposed in [3], the first term in (2) represents a realization of the first objective of [3], minimizing the overall completion time of QCs and the second term of (2) denotes the second objective of [3] in order to minimize the completion time of each QC. We consider the situation that each agent aims to minimize its own objective and has only partial information of the whole problem. Therefore, the first objective is proposed instead of the first objective of the QCSP proposed in [3] in order to find a near optimal solution of the QCSP.

As an example, consider Fig. 1, QCSP problem can be formulated as a DCOP using the notation just introduced as follows. The set of agents is given by: $\mathbb{A} =$ $=\{"QCA_1","QCA_2","QCA_3","QCA_4"\}$ (m=4). The set of n sets of physical component $\mathbb{X} = \{\mathbb{X}_1, \mathbb{X}_2, \mathbb{X}_3, \mathbb{X}_4\} = \{\{QC_1\}, \{QC_2\}, \{QC_3\}, \{QC_4\}\}$ from the set of all physical components is given by: $\mathcal{X} = \{QC_1, QC_2, QC_3, QC_4\}$. The set of doable tasks is $\mathbb{D} = \{\mathbb{D}_1, \mathbb{D}_2, \mathbb{D}_3, \mathbb{D}_4\}$ and the set of all tasks is $\mathfrak{D} = \{"A","B","C","D","E","F"\}$. According to Fig. 1, the sets of doable tasks per physical component are:

$\mathbb{D}_1 = \{d_1^1, d_1^2\} = \{"A","B"\}$,
$\mathbb{D}_2 = \{d_2^1, d_2^2, d_2^3, d_2^4\} = \{"A","B","C","D"\}$,
$\mathbb{D}_3 = \{d_3^1, d_3^2, d_3^3, d_3^4\} = \{"C","D","E","F"\}$,
$\mathbb{D}_4 = \{d_4^1, d_4^2\} = \{"E","F"\}$,

Moreover, for this example we get:

$\mathbb{D}_{1,2}^{\text{ov}} = \{"A","B"\} = \mathbb{D}_{2,1}^{\text{ov}}$, $\mathbb{D}_{2,3}^{\text{ov}} = \{"C","D"\} = \mathbb{D}_{3,2}^{\text{ov}}$, $\mathbb{D}_{3,4}^{\text{ov}} = \{"F","E"\} = \mathbb{D}_{4,3}^{\text{ov}}$, $\mathbb{D}_1^{\text{nv}} = \mathbb{D}_2^{\text{nv}} = \mathbb{D}_3^{\text{nv}} = \mathbb{D}_4^{\text{nv}} = \{" "\}$.

The overall cost (2) for the problem can be easily calculated based on the constraints of the problem, as will be discussed in more detail below.

We consider handling a container as a task, represented by a tuple: $task\ i = < l_i, \acute{l_i} >$, where, l_i refers to the physical source location of container i and $\acute{l_i}$ refers to the destination of the container. As we will see, the sequence of tasks handling will be important in this problem. We assume that the QCs are the same and all tasks need the same handling time by a QC. Using this, we can define a transition time matrix T, which contains the traveling time and processing time required for each QC in order to handle tasks. Matrix T is independent of the particular QC and defined as:

$$T_{n_{\mathbb{D}} \times n_{\mathbb{D}}} = \begin{bmatrix} t_{11} & t_{12} & \cdots & t_{1n_{\mathbb{D}}} \\ t_{21} & t_{22} & & \\ \vdots & & \ddots & \vdots \\ t_{n_{\mathbb{D}}1} & & \cdots & t_{n_{\mathbb{D}}n_{\mathbb{D}}} \end{bmatrix} \tag{4}$$

where $(t_{ij} | i \neq j)$ is the time required between finishing task i starting task j (i.e., the time required by the QC to perform task j right after completing task i); $(t_{ij} | i = j)$ is the processing time of task i (i.e., the time required by the QC to handle task i). When the transition time matrix $T_{n_{\mathbb{D}} \times n_{\mathbb{D}}}$ has been given by the supervisory system, all data regarding the time required to perform each task and time spend on traveling between two tasks are provided for QCAs. The QCAs can use this information to decide which tasks should be handled and in which sequence the tasks have to be chosen. In this section, we discuss how the QCAs allocate tasks and in which order.

We can fulfill objective 1 by defining the overall cost as defined in (2). This definition limits cost functions per QCAs, until the two neighbor QCs cost functions $\mathfrak{j}_i(\mathbb{T}_i)$ and $\mathfrak{j}_{i+1}(\mathbb{T}_{i+1})$ close to each other. This means, the overall processing time for neighbor QCs become as same as possible until all QCs have the same load for doing their tasks to fulfill objective 1. Objective 2 will be performed by defining a cost for the DCOP in equation (3) such as:

$$c_i(t_i^j) = \begin{cases} t_{jj} + t_{jk} & \text{if there is task } k \text{ after task } j \\ t_{jj} & \text{if task } j \text{ is the last allocated task} \end{cases} \tag{5}$$

This means, $c_i(t_i^j)$ is the cost represents time for performing task j plus time for traveling from the previous task j to the next task k for QCi.

We define the following constraints for distributed QCSP to assure QCs will perform correctly without interferences (based on the centralized QCSP formulation of [3]):

$$\bigcup_{i=1}^{n} d_i = \mathfrak{D} \tag{6}$$

$$\mathbb{T}_i \cap \mathbb{T}_j = \emptyset \qquad \forall i \in \{1, \dots, n\}, \ \forall j \in \{1, \dots, n\}, i \neq j, \tag{7}$$

$$\nexists (d_i^n \in \mathbb{D}_i \wedge d_j^m \in \mathbb{D}_j) \qquad \forall i,j \in \{1, \dots, n\}, i < j \ \ \forall n \in \{1, \dots, n_{\mathbb{D}_i}\}, \forall m \in \{1, \dots, n_{\mathbb{D}_j}\}, n < m, \tag{8}$$

$$\nexists (\delta_i^{\mathbb{H}_h} \in \mathbb{D}_m | \delta_j^{\mathbb{H}_h} \in \mathbb{T}_m) \qquad \forall h \in \{1, \dots, n_{\mathbb{H}}\}, \forall i,j \in \{1, \dots, n_{\mathbb{D}_h^{\mathbb{H}}}\}, i < j, \tag{9}$$

where constraint (6) ensures that all tasks will be handled, constraint (7) guarantees that the non-preemption constraints between QCs are satisfied (i.e., each task can be handled by one and only one QC), constraint (8) prevents the QC from interfering and impossible allocations at the same time, and constraint (9) represents precedence relationships between tasks and describes the limitation that one QC at a time can handle the containers within a hold. For the example of the previous section, we can easily calculate the overall cost (2) and after that remove the non-consistent solutions from the complete set of solutions with respect to (6)-(9). To calculate the overall cost (2), we need to first calculate the costs related to each QC in (3). The costs of doing tasks in (5) relating to a QC have to be defined in advanced by the supervisor. Below, we will see how the above two steps are implemented within the algorithm proposed.

C. E-ABT Algorithm

In this section, we propose an algorithm that can solve the distributed QCSP problem introduced above, E-ABT. The procedure for solving the distributed QCSP is divided into two main steps: "allocation" and "re-scheduling":

Step 1 "Allocation" Assign the overlapping tasks of each QCAi based on their cost and calculate for each QCAi the completion time (t_i). To realize the second term of (2), each agent tries to allocate the overlapping tasks and compares the completion time of each agent with the completion time of a neighboring agent, until both agents have a near equal

```
• procedure **initialize**
1. initialize all overlapping ($\mathbb{D}_{ij}^{\mathrm{ov}}$) and non-overlapping ($\mathbb{D}_i^{\mathrm{nv}}$) tasks;
2. initialize $T_{threshold}$;
// reset *agent_view* for all $QCAs$;
3. $a_i \leftarrow$ agent identifier number from supervisory system, $\forall i \in \mathbb{A}$;
4. $s_i \leftarrow active, \forall i \in \mathbb{A}$;
5. $d_i \leftarrow \mathbb{D}_i^{\mathrm{nv}}, \forall i \in \mathbb{A}$;
6. $t_i \leftarrow 0, \forall i \in \mathbb{A}$;
7. **do re-schedule_values**;     // re-scheduling non-overlapping tasks $\widetilde{D}$
// ordering overlapping tasks with valid tag numbers
// for all $QCAs$ based on their location w.r.t neighbor agents
8. Initialize sequences of all overlapping tasks;

• **when** received ( **run!**, $(a_j, s_j, d_j, t_j)$ ) **do**
    add $(a_j, s_j, d_j, t_j)$ to *agent_view*; // update *agent view*
    **do check_agent_view**;
**end_do**;

• **when** received ( **request_value?**, $(a_i, s_j, d_j, t_j)$ , $d$ ) **do**
    add $(a_j, s_j, d_j, t_j)$ to *agent_view*; // update *agent view*
    **if** ($a_i$ is *done*!) **do**
        $s_i \leftarrow passive$;
    **end do**;
    remove $(a_i, d)$ value ($d$) from *agent_view*;
    **send** ( **add_value!**, $(a_i, s_i, d_i, t_i)$ , $d$ ) to $a_i$;
    **do schedule_values**;
**end_do**;

• **when** received ( **add_value!**, $(a_j, s_j, d_j, t_j)$ , $d$ ) **do**
    add $(a_j, s_j, d_j, t_j)$ to *agent_view*;     // update *agent view*
    **do select_value** ($d \in \mathbb{D}_{ij}^{\mathrm{ov}}$);     //add value ($d$) to *agent_view*;
    **do check_agent_view**;
    **do re-schedule_values**;
**end_do**;

• **when** received ( **no_value!**, $(a_j, s_j, d_j, t_j)$ ) **do**
    add $(a_j, s_j, d_j, t_j)$ to *agent_view*;     // update *agent_view*
    **if** ($a_j$ *is active*) **do**
        $s_j \leftarrow passive$;
        add $(a_j, s_j)$ to *agent_view*;
    **elseif** ($x_j$ *is passive*)
        $s_j \leftarrow done!$;
        add $(a_j, s_j)$ to *agent_view*;
    **do check_agent_view**;
**end do**;
```

Algorithm 1. E-ABT algorithm (Part I)

completion time, i.e., the difference between their completion times is less than a pre-defined threshold time, $|t_1' - t_2'| < T_{threshold}$. In this case the algorithm will be terminated after a search through all tasks. If the QCAs reach an inconsistency (i.e., situation in which constraints are violated), they will enter the backtracking procedure until their completion time fulfills the condition again. In this step, QCAs try to allocate tasks with respect to constraints (6)-(8) in order to achieve a minimal completion time for carrying out the tasks involving traveling from source to destination $(t_{ij} | i = j)$.

Step 2 "Re-scheduling" The allocation of Step 1 does not result in an optimized sequence for handling of tasks, since the allocation has been done independent of the order of tasks and precedence relations of tasks. Depending on the source position, destination position and handling time related to a task defined by the supervisory system, the order in which the allocated tasks (both non-overlapping and overlapping tasks) are handled has a significant influence on the completion time.

After each QCA has allocated its tasks in Step 1, it tries to re-order the sequence of the task handling by solving locally a conventional "single machine scheduling model, minimizing maximum cost" [10] including precedence relationships constraint (9) between tasks. Each QCA performs re-scheduling and based on this calculates the final completion time (t_i'). In this way, traveling times between adjacent tasks $(t_{ij} | i \neq j)$ as given by the transition time matrix T in (4) are the key information required by QCAs for calculating the optimal sequence of tasks for themselves.

For the distributed QCSP, the two objectives as discussed in the previous section are pursued. We need to obtain an as close as possible equal completion time for all QCs. This ensures that the full capacities of QCs are used and the QCs work together (objective 1). To satisfy the minimal completion time objective for each individual QC (objective 2). Performing step 1 and 2 is done by QCAs sequentially, i.e., right after allocation (step 1), a QCA performs rescheduling to calculate its completion time t_i' (step 2) and again compares it with neighboring QCAs at the next time of performing the E-ABT (step 1).

So far, we have seen how the two objectives of distributed QCSP can be implemented by the 2 steps of E-ABT. In this case, QCAs must carry out a distinct distributed algorithm (E-ABT) to perform both above steps in order to strive for achieving the objectives.

The E-ABT algorithm is an extended version of ABT. E-ABT has the possibility to take into account assigning multiple values to a variable with minimal knowledge about neighbors of an agent and involving minimal communication. E-ABT constructs a *consistent_set* and an *inconsistent_set* of values of agents during the execution of the algorithm. It maintains these sets using a structure called an *agent_view*. The *consistent_set* refers to the set of values of variables that satisfy the problem constraints. Three main states are considered for each agent in the E-ABT: *active*, *passive*, and *done*!. The proposed algorithm is implemented on each QCA in exactly the same way. The steps of E-ABT are as follows:

1) Initialization (Algorithm 1)

At initialization, all of the overlapping and non-overlapping tasks are assigned to agents by the container terminal supervisory system. The tasks that are located between agents x_i and x_j and can be handled by either of them are in the overlapping set $\mathbb{D}_{ij}^{\mathrm{ov}}$. After each agent has received a list of all of its tasks from the terminal supervisory system, each agent tries to schedule all the non-overlapping tasks ($\mathbb{D}_i^{\mathrm{nv}}$). After that all the agents will be initialized with sequenced overlapping tasks by calling *re-schedule_values* with tag numbers related to their ship bay number by the supervisory layer. The *initialize* procedure finishes and the agents are in the state *active*. The *agent_view* stores the names, values, domains, and functional relationships of the agents in their environment.

2) Backtrack and Re-Scheduling Procedures

Some procedures used by the algorithm are introduced first. These procedures handle sending of packets that by agents in order to inform or ask another agent about its actions. Backtracking will be done for allocated tasks in case of inconsistency and save the inconsistent solutions in a subset to prevent happening such these solutions during search again. In

```
• procedure check_agent_view
// check agent time lower than both neighbors
if ($(t_i \le t_j) \wedge (t_i < t_k)$ ) do                                  (i)
    if ( $a_i$is $active$ ) do
        if ( $\exists\, d \in \mathbb{D}_{ij}^{ov} \mid d \notin \mathbb{T}_i, \mathbb{T}_j$ ) do  // first select from right then left
            do select_value ($d_i \in \mathbb{D}_{ij}^{ov}$);
            send ( run!, $(a_i, s_i, d_i, t_i)$ ) to $a_j$;
            do re-schedule_values;
        elseif ($\nexists\, d \in \mathbb{D}ij$ov | $d \in \mathbb{T}j$) do       // no value exists
            send ( no_value!, $(a_i, s_i, d_i, t_i)$ ) to the neighbors;
            $s_i \leftarrow passive$;       //$a_i$ become $passive$
        end do;
    // if all values were selected by their near agents
    elseif ($a_i$ is $passive$ $\wedge$ all neighbors are $passive$)
        if ( $\exists\, d \in \mathbb{D}_{ij}^{ov} \mid d \in \mathbb{T}_j$ ) do       // first select from left then right
            do backtrack( $d_j \in \mathbb{D}_{ij}^{ov}$ );
        elseif ( $(\nexists\, d \in \mathbb{D}_{ij}^{ov} \mid d \in \mathbb{T}_j) \wedge$ all $x_j$ are $passive$)
            send ( no_value!, $(a_i, s_i, d_i, t_i)$ ) to all $a_j$;
            $s_i \leftarrow done!$;
        end do;
    elseif ($a_i$ is $passive$ $\wedge$ $a_j$ is not $passive$)
        if ( $a_j$ is $active$ ) do       //first select value from right then left
            send ( run!, $(a_i, s_i, d_i, t_i)$ ) to $a_j$;
        else if ( $a_j$ is $done$ )
            send ( no_value!, $(a_i, s_i, d_i, t_i)$ ) to neighbors $a_j$;
            $s_i \leftarrow passive$;
        end do;
    end do;
// check agent time lower than one and higher than another neighbor
elseif ( $(t_i \ge t_j) \wedge (t_i < t_k) \wedge (a_i$ is $active)$ )              (ii)
    if ($a_j$ is $active$) do
        send ( run!, $(a_i, s_i, d_i, t_i)$ ) to $a_j$;
    elseif ($a_j$ is not $active$)
        do select_value ($d \in \mathbb{D}_{ij}^{ov}$);
        send ( run!, $(a_i, s_i, d_i, t_i)$ ) to $a_j$;
        do re-schedule_values;
    end do;
// check agent time be greater than its neighbors
elseif ( $(t_i > t_j) \wedge (t_i > t_k) \wedge (a_i$ is $active)$ )               (iii)
    if ( $(t_k > t_j) \wedge (a_j$ is $active)$ ) do   // first right then left
        send ( run!, $(a_i, s_i, d_i, t_i)$ ) to $a_j$;
    elseif ($a_j$is not $active$)
        do select_value ($d \in \mathbb{D}_{ij}^{ov}$);
        send ( run!, $(a_i, s_i, d_i, t_i)$ ) to $a_j$;
        do re-schedule_values;
    end do;
// check if agent is in passive state
elseif ($a_i$ is $passive$)                                                   (iv)
    if ($a_j$ is $active$) do       // first right then left
        send ( run!, $(a_i, s_i, d_i, t_i)$ ) to $a_j$;
    // check if neighbors have no possible value and check consistency
    elseif ( $(\nexists\, d_i \in \mathbb{D}_{ij}^{ov})$ )
        // check sub_solution consistency with $a_j$ (first left then right)
        if ( $|t_i - t_j| > T_{threshold}$ ) do
            if ($a_j$ is $done!$) do
                $s_j \leftarrow passive$;
            end do;
        //request the most appropriate task from neighbor
        do backtrack($d_j \in \mathbb{D}_{ij}^{ov}$);
        elseif (this sub-solution is consistent)
            send ( no_value!, $(a_i, s_i, d_i, t_i)$ ) to the neighbors;
            $s_i \leftarrow done!$;
        end do;
    end do;
// check if agent is in $done!$ state
elseif ($a_i$ $is$ $done!$)                                                  (v)
    if ($a_j$ is $active$) do       // first left then right
        send ( run!, $(a_i, s_i, d_i, t_i)$ ) to $a_j$;
    elseif ($a_j$ is $passive$ ) do       // first left then right
        send ( run!, $(a_i, s_i, d_i, t_i)$ ) to $a_j$;
    elseif (all neighbors are $done!$)
        consistent_subset ← this_solution;
        broadcast terminate algorithm;
    end do;
end do;
```

Algorithm 3. E-ABT algorithm (Part III)

```
• procedure backtrack($d_j \in \mathbb{D}_{ij}^{ov}$)
    inconsistent_subset ← this_solution;
    if ($a_j$ is $passive$) do       //check if neighbor $a_j$ can remove the value
        if ($\exists\, d_j \in \mathbb{D}_{ij}^{ov}$) do
            remove this_solution from consistent_subset;
            send ( request_value?, $(a_i, s_i, d_i, t_i)$ , $d_j$ ) to neighbor $a_j$;
        elseif ($\nexists\, d_j \in \mathbb{D}_{ij}^{ov}$)
            send ( no_value!, $(a_i, s_i, d_i, t_i)$ ) to the neighbors;
            $s_i \leftarrow done!$;
        end do;
end do;

• procedure select_value($d \in \mathbb{D}_{ij}^{ov}$)
    if ($\exists\, d \in \mathbb{D}_{ij}^{ov}$) do       //select a most appropriate value;
        add $(a_i, d_j)$ to agent_view;
    elseif ($\nexists\, d \in \mathbb{D}_{ij}^{ov}$)
        send ( no_value!, $(a_i, s_i, d_i, t_i)$ ) to the neighbors;
        $s_i$ ← passive;
end do;

• procedure re-schedule_values
    1. construct single machine scheduling model for the allocated tasks;
    2. connect $x_i$ to CPLEX;
    3. minimize the cost using Branch and Bound;
    4. calculate new completion time $t_{i(new)}$;
    5. $t_i \leftarrow t_{i(new)}$;       // update completion time
end do;
```

Algorithm 2. E-ABT algorithm (Part II)

order to re-schedule the allocated tasks (in Step 2 of the distributed QCSP) in each step, the *re-schedule_values* procedure is defined. As soon as the *re-schedule_values* procedure is called, the single machine scheduling problem is solved by each QCA for the tasks allocated so far using CPLEX, [11], which employs a Branch and Bound search. Hence, each QCA can calculate the sequence of handling the allocated tasks to decrease its completion time and applying precedence relationship constraint.

3) Checking agent_view (Algorithm 3)

The task assignment for each agent is made by the *check_agent_view* procedure using data from neighboring agents. In this procedure, agents will decide on a task assignment with the knowledge of themselves or information from the *agent_view* of their neighbors (step 1). After an agent has assigned each task and updated its *agent_view*, each agent tries to optimize its own problem to reach its minimum completion time $\boldsymbol{t_i}$ (step 2) and compares this minimum completion time with its neighbors' $(\boldsymbol{t_j}, \boldsymbol{t_k})$ to ensure that all tasks will be completed by the QCAs at nearly the same time (step 1). In the *active* state, agent $\boldsymbol{x_i}$ (note that each agent $\boldsymbol{a_i}$ has only one variable and therefore $\boldsymbol{a_i \equiv x_i}$) compares his $\boldsymbol{t_i}$ with the value of its neighbors $(\boldsymbol{x_j}, \boldsymbol{x_k})$. It may then allocate a task from $\overline{\boldsymbol{D}}_{i,j}$ or $\overline{\boldsymbol{D}}_{i,k}$. After assigning the tasks, an agent sends a *no_value!* message to its neighbors and changes its state to *passive*. When all agents enter the *passive* state, no more tasks exist for allocation.

The agents will then check the *sub_solution* consistency and do a *backtrack* if necessary by sending a *request_value?* message to their neighbors and receiving an *add_value?* message from their adjacent agents. The *backtrack* steps will be continued until all the agents obtain agreement. After the *backtrack* has completed the agents enter the *done*! State. In this state nothing more has to be done, unless a neighbor is forced to move back again into the *passive* state. When all the agents are in the *done*! state, the algorithm terminates. The ABT algorithm as proposed originally allocates only one value to each variable. E-ABT is proposed for allocating multiple values to each variable, at minimal communication. Although the main features of ABT and E-ABT (i.e., selecting value, backtracking value, and check agent view and consistency) are the same, there are differences regarding how those features can be implemented. In ABT, agents try to find appropriate values according to the problem constraint(s) using a mixture of selecting values a so-called hyper-resolution rule and backtracking. On the other hand, in E-ABT, due to the use of the initializing procedure, agents try to select values in the first phase (*active*), then try to backtrack values in the second phase (*passive*) if there is any inconsistency, and then complete the algorithm when all agents are in the *done!* state. Since in our case all values relate to at most two agents, there is no need to adopt the hyper-resolution strategy; the filtering algorithm is sufficient. In E-ABT, the overlapping tasks for two neighboring QCAs are first prioritized by the supervisor according to its position regarding to its agent neighbors. This can help agents to decide faster on which value is better to be assigned in each iteration and prevent QCA from selecting irrelevant tasks. This algorithm is therefore categorized as an incomplete algorithm.

III. SIMULATION EXPERIMENTS

In this section we illustrate via simulation experiments the potential of the approach proposed. The simulation is carried out using Java and the JADE middleware [12], [13] for simulating agent entities, interactions, and behaviors. Each agent can solve the single machine scheduling problem for its allocated tasks, i.e., constructing the transition time matrix T, via a connection between Java and the ILOG CP Solver library. The simulations have been carried out on a DELL 1320 Core 2 Duo 2.2GHz with 3 GB RAM. We consider a setup of 4 quay cranes with equal distribution of tasks among them. That means, in our simulation we consider the same number of tasks for each QC. We can define for each quay crane a different overlapping index $R_{i,j}$ (the overlapping index indicates how much the neighbor QCs i and j interact with each other, defined as the ratio of overlapping tasks between two QCs to all tasks (overlapping and non-overlapping) for a given QC); this parameter varies between 0 and 1. For the sake of simplicity, we consider $R_{1,2} = R_{2,3} = R_{3,4} = R$. In the experiments, we consider 4 scenarios: $R = 0$, $R = 1/4$, $R = 1/2$, and $R = 1$. The number of tasks considered varies between 10 and 1000.

Fig. 2 shows the results of the simulations. The figure illustrates that by increasing the number of tasks to be scheduled, the processing time increases, almost at an exponential rate. The figure also illustrates that when considering a constant number of tasks, but increasing the overlapping ratio, the processing time also increases. With more overlapping tasks, E-ABT has to spend more time on the backtracking phase in the "allocation" step. Therefore, in both steps there is a large amount of computations.

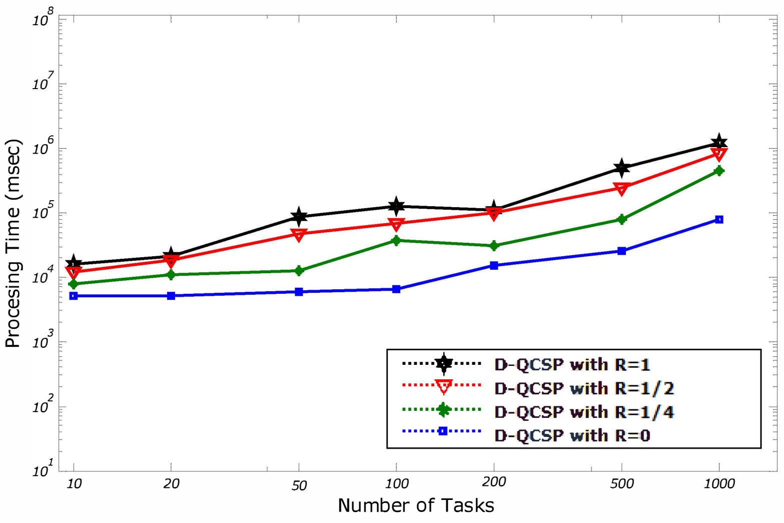

Fig. 2. The computation time for distributed QCSP for different *R*

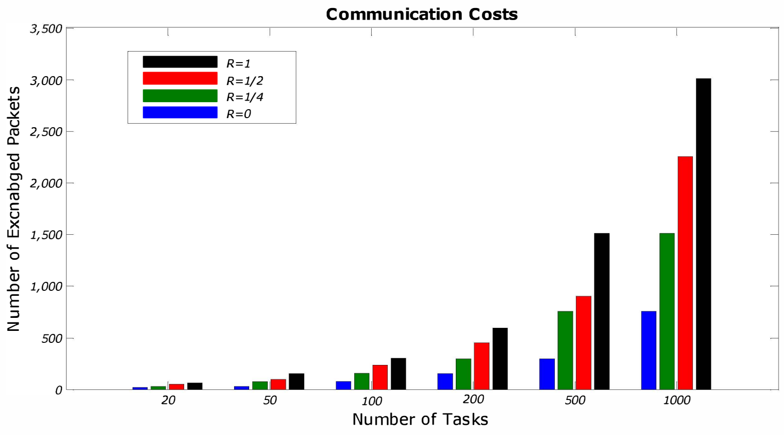

Fig. 3. Number of exchanged packets for varying number of tasks and different overlapping ratio (*R*)

Communication between agents is one of the important aspects when using agent-based implementations. With E-ABT we propose an algorithm that can decrease the number of exchanged packets considerably. The most important parameter that affects the communication is the overlapping tasks ratio R. The higher the overlapping tasks ratio, the more scheduling of overlapping tasks needs to be coordinated. Communication costs are shown for the different scenarios considered in Fig. 3. Fig. 4 illustrates the result when comparing a centralized QCSP version with the distributed version proposed. We observe that by increasing the amount of tasks in each scenario the execution performance of D-QCSP improves in comparison to QCSP. With a lower number of tasks to be scheduled the QCSP performs better.

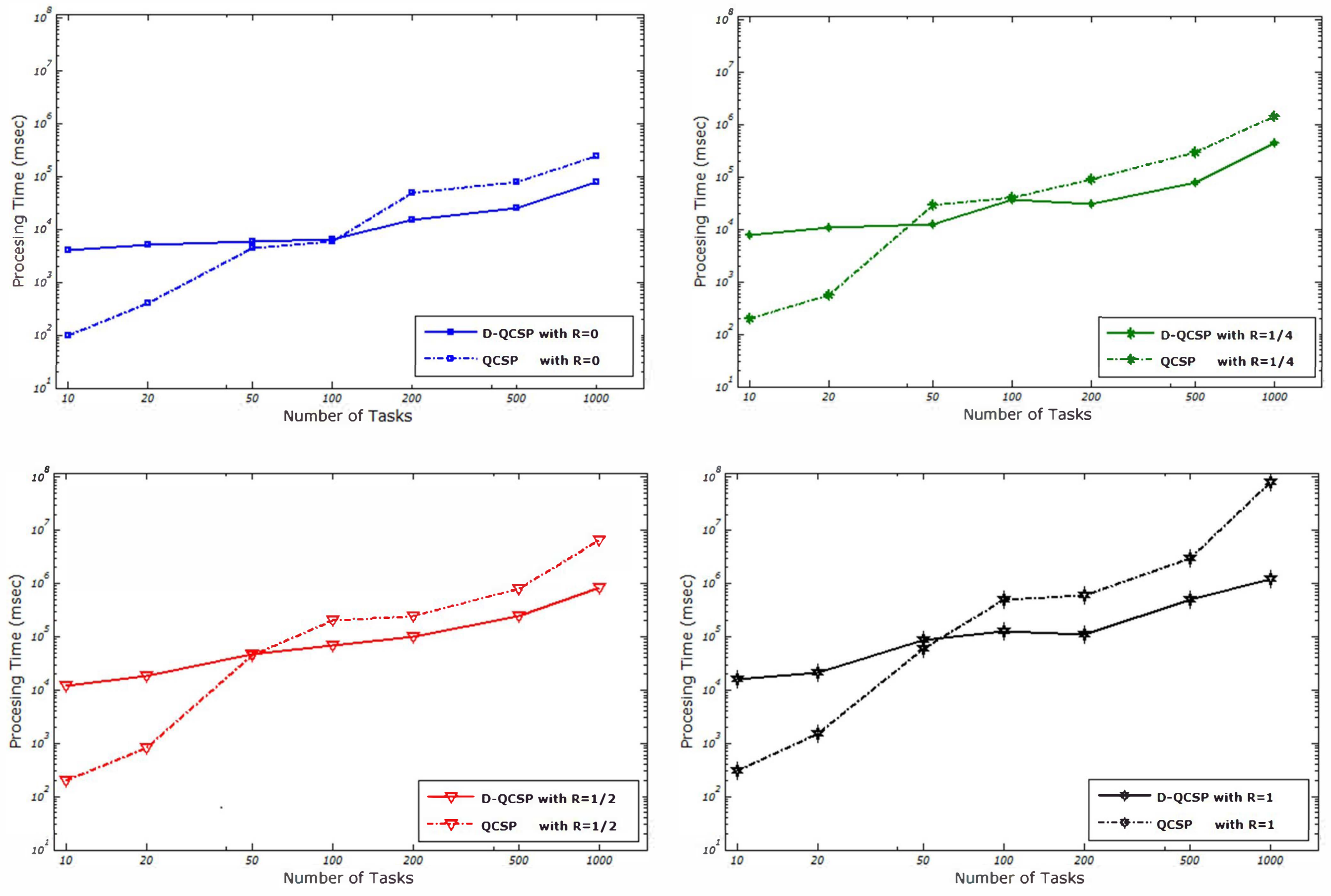

Fig. 4. Comparisons between the QCSP and D-QCSP for each scenario (different *R*).

IV. Conclusions and Future Research

In this paper, we have proposed a distributed quay crane scheduling problem with tasks overlapping constraints among multiple quay cranes processing loading/unloading containers of a common containership. We proposed an approach in which each agent makes its own local decisions and controls its own allocated area based on solving a distributed constraint optimization problem. Future work will consider a MAS architecture for the complete container terminal.

Acknowledgements

This research is partially supported by the VENI project ``Intelligent multi-agent control for flexible coordination of transport hubs" (project 11210) of the Dutch Technology Foundation STW, a subdivision of The Netherlands Organisation for Scientific Research (NWO).

References

[1] United Nations Conference on Trade and Development (UNCTAD), Review of Maritime Transport, pp. 40, New York and Geneva 2013.

[2] Alphaliner Weekly Newsletter Vol. 2011 Issue 04, pp. 1, website: http://www.alphaliner.com/.

[3] K.H. Kim and Y.M. Park. "A crane scheduling method for port container terminals," European Journal of Operational Research, 156:752–768, 2004.

[4] G. Weiss, "Multiagent Systems - A Modern Approach to Distributed Modern Approach to Artificial Intelligence," The MIT Press, 1999.

[5] M. Rebollo, V. Julian, C. Carrascosa, and V. Botti, "A MAS Approach for Port Container Terminal Management," in Proceedings of the 3rd Iberoamerican workshop on DAI-MAS. Atiaia, Sao Paulo, Brasil, 2001.

[6] P. Davidsson, L. Henesey, L. Ramstedt, J. Törnquist, F. Wernstedt. "An analysis of Agent-Based Approaches to Transport Logistics," Transportation Research Part C: Emerging technologies 13.4, 255-271, 2005.

[7] I.F.A. Vis and R. de Koster, "Transshipment of containers at a container terminal: an overview," European Journal of Operational Research, Vol. 147, pp. 1-16, 2003.

[8] A. Petcu, "A Class of Algorithms for Distributed Constraint Optimization: Volume 194 Frontiers in Artificial Intelligence and Applications (Dissertations in Artificial Intelligence)," IOS Press, 2009.

[9] A. Meisels, "Distributed Search by Constrained Agents (Algorithms, Performance, Communication) - Advanced Information and Knowledge Processing," Springer, 2008.

[10] M. Yokoo, E. Durfee, T. Ishida and K. Kuwabara, "Distributed Constraint Satisfaction Problem: Formalization and Algorithms," IEEE Transactions on Knowledge and Data Engineering, Vol. 10, No. 5, Sep-Oct 1998.

[11] J. Blazewicz, K. Ecker, E. Pesch, G. Schmidt, and J. Weglarz, Handbook on scheduling: from theory to applications. Springer, 2007.

[12] IBM ILOG CPLEX Optimisation Studio V12.2 – Reference Manual.

[13] JADE — Java Agent Development Framework, [Online]. Available: http://jade.tilab.com.

[14] F.L. Bellifemine, G. Caire, and D. Greenwood, "Developing multi-agent systems with JADE," Wiley, 2007.