Distributed Quay Crane Scheduling

using Extended Asynchronous Backtracking

Authors Name/s per 1st Affiliation *(Author)*
line 1 (of *Affiliation*): dept. name of organization
line 2: name of organization, acronyms acceptable
line 3: City, Country
line 4: e-mail address if desired

Authors Name/s per 2nd Affiliation *(Author)*
line 1 (of *Affiliation*): dept. name of organization
line 2: name of organization, acronyms acceptable
line 3: City, Country
line 4: e-mail address if desired

***Abstract*— This paper presents a distributed multi-agent solution for a quay cane scheduling problem called Distributed Quay Crane Scheduling Problem (D-QCSP). The objective of this work is mainly to increase the container throughput of a container port by minimizing the amount of time necessary to load into and discharge from a ship. Distributed problem solving has the potential to significantly reduce the overall problem's makespan in the case of a high amount of tasks. The proposed multi-agent system (MAS) is implemented using the Java Agent Development Environment (JADE), which is fully compliant with the Foundation of Intelligent Physical Agents (FIPA). The MAS architecture can be used as a part of container terminal automation system. First, the scheduling problem is formulated in a centralized fashion using an efficient Optimization Programming Language (OPL) and then solved using ILOG CP Optimizer. Second, the QCSP is formulated as a Distributed Constraint Problem (DCOP) in which each crane is considered as an agent. A new negotiation algorithm using the FIPA-Contract-Net Interaction Protocol (IP) called Extended Asynchronous BackTracking (Extended ABT) algorithm is proposed to solve the DCOP. The FIPA-Contract-Net is one of the protocols introduced by FIPA capable of implementing complex algorithms for negotiation among agents asynchronously. Also among many distributed algorithms, ABT could be a best choice because of its consistency and simplicity. Agents implement the distributed algorithm continuously in order to solve the DCOP dynamically. In this research we provide an experimental implementation of the proposed MAS embedding commercial tools for further industrial usage.**

Keywords-component; Multi-Agent Systems (MAS), Distributed Quay Crane Scheduling Problem (D-QCSP), Distributed Constraint Programming (DCOP), Extended Asynchronous BackTracking (Extended ABT), Container Terminal.

I. INTRODUCTION

Every year millions of TEU (Twenty-foot Equivalent Unit steel container) are handled by container terminals. Maritime container terminals all over the world are the most important part for transshipment and intermodal container transfers, Fig. 1 shows that by the end of 2011 almost 30 millions 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 burgeoning 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 process of containers to/from ship is done as quickly as possible.

Commented [Z1]:
IZ: Please revise it with latest stats and information

A typical modern automated container terminal (like The European Combined Terminal (ECT) in Rotterdam, one of the few automated ports in the world) has several stages for container management. The loading and unloading process of containers can be divided into different subprocesses [3]. First, containers have to be taken off from ships by Quay Cranes (QC) with the arrival of ships at the port. Next, the containers are conveyed from QCs by vehicles (usually by Automated Guided Vehicles (AGVs) at the most automated ports) which travel between the ship and the stack at the port yard. The stacking process can also be done by Automated Stacking Cranes (ASCs) automatically, which have their own subprocesses. After a certain storage period, the containers are retrieved from the stack by another ASC and transported by other vehicles for transshipment to barges, vessels, trucks or trains. This process can also be executed reversely, to load containers onto ships.

One of the problems that has to be addressed by port automation systems is the Quay Crane Scheduling Problem (QCSP). This problem consists of determining a handling sequence of tasks (container holds) for QCs assigned to a containership considering interference between QCs. In other word, after assigning containers to be handled by each QC by port supervisory system, tasks can be divided into two main groups. One group refers to the tasks that have to be handled by one QC, while other group consists of the tasks that will be handled by two adjacent cranes. In this study, we assume an area called "overlapping area", contains tasks of the second group which we will discuss later. A large amount of works have been

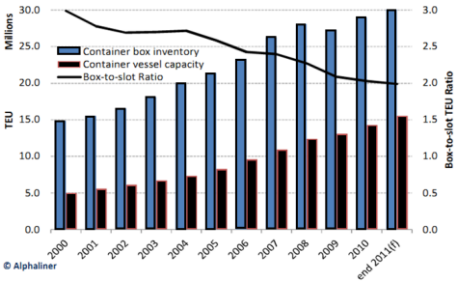


Fig. 1 - Container box inventory-to-containership fleet ratio 2000-2011
(Source: Alphaliner Weekly Newsletter Vol. 2011 Issue 10)

done to optimize such problem in different aspects [4], [5], [6], [7] introduced in Section III.

In the past, a large deal of centralized systems has been proposed, simulated and implemented. These days many busy ports in the world are dealing with large amounts of demands and transportations, which require more rapid technology to optimize the flow of container transshipment. In this case, the lack of new distributed technology as a management and decision maker is being sensed especially in automated ports. Using a distributed approach, the whole optimization problem is divided into sub-problems which can be handled by a specific agent. This approach can reduce the amount of optimization computations significantly in comparison with a centralized approach. Also, in a hierarchical distributed architecture, information safety and security management will be guaranteed.

Distributed Artificial Intelligence (DAI) is the study, construction, and application of Multi-Agent Systems (MAS), that is, systems in which several interacting, intelligent agents pursue some set of goals or perform some set of tasks [8]. Behavioral flexibility and rationality are achieved by an agent on the basis of processes such as problem solving, planning, decision making, and learning.

Implementing a MAS in recent years for logistics and port automation systems seems to be promising [9], [10], [11], [12]. Today these systems are not simply a research topic, but are also beginning to become an important subject of academic teaching, and industrial, and commercial applications. Several standards have been proposed for designing, programming, and implementing MAS. The main Foundation for Intelligent Physical Agents (FIPA) was first established in 1996 as an international non-profit association for developing a set of standards relating to software agent technology [13]. FIPA was reincorporated in mid-2005 as a standards committee of the IEEE Computer Society, lending credibility to the use of FIPA as standards for industrial and commercial multi-agent system applications.

The entity platform used by agents in the proposed MAS design is provided by the Java Agent Development Environment (JADE) [14], [15]. JADE is a well-known middleware, FIPA standards compliant, and fully implemented in Java language. This platform enables agents executing, communicating with other agents by its advanced libraries [13]. JADE messages adhere to FIPA-ACL standards. The communication between agents is done by means of asynchronous messages, which are based upon a FIPA standard, called FIPA-ACL Also FIPA has proposed several Interaction Protocols (IPs) which we will discuss later. The FIPA-Contract-Net as a well-defined IP [16] typically is a way to carry out a complex conversation. This protocol has the ability to provide a platform to coordinate interactions between agents by asynchronous algorithms like Asynchronous BackTracking (ABT) [17]. By means of ABT, we can decompose the main problem into sub-problems, this is done by assigning each task in two separated pre-defined area called overlapping or non-overlapping area which is defined by supervisory system to each crane. The non-overlapping area consists of tasks that are fixed for each quay crane agent; on the other hand, the tasks which are placed in overlapping area did no assign to their neighbor cranes. The procedure for allocating the overlapping tasks to each crane is regarding to makespan of each crane. As we will see a centralized scheme in Section III.AIII, the objective function consists of a weighted sum function, that means we try to minimize overall time and time of each crane simultaneously. In the distributed scheme, each agent plans to assign overlapping tasks step wisely, minimizes its crane makespan, and communicate its states step by step to other agents. In this way, negotiations among agents have to be done by an asynchronous algorithm. Generally, in an asynchronous procedure, all agents are active at any time, having a high degree of parallelism based on their local knowledge without any global control.

This paper consists of a novel approach for implementing a cooperative-based MAS for solving a QCSP distributedly. This study particularly refers to a modified version of the ABT algorithm based on FIPA-Contract Net IP for performing coordination among agents in order to solve a problem co-operatively. Previously, studies are mainly limited to solve QCSP as a centralized approach. Task assignment for each quay crane was performed by a single processor and reaching solutions was taking too long with respect to high amount of tasks. The main advantage of using distributed approach is reducing the overall time for solving QCSP efficiently by mean of using multiple processors. We will study more in Section VI.

This paper is organized as follows. An overview MAS and the proposed architecture is given in Section II. The original mathematical model of QCSP, its formulation as mixed-integer optimization problem enabling solving by CPLEX, and the formulation enabling solving by the ABT algorithm is presented in Section III. In Section V, the quay crane agent structure is described and the proposed algorithm based on the FIPA protocol is shown step by step.. Section VII is devoted to conclusions of the present work with discussion about analysis of this work and future works.

II. AGENTS AND MULTI AGENT SYSTEMS (MAS)

A. Overview and Definitions

Today, researches in the area of Distributed Artificial Intelligence (DAI) have been focused on intelligent agents as a part of MAS in an increasingly wide variety of applications. An intelligent agent is a real or virtual entity, able to act on itself and on its environment, generally populated by other agents. It has a partial representation of its environment to perform its actions.

With the above assumption, a MAS would be an autonomous system which many agents in it can form a community together and their act may depend on each as well as their act can be autonomously within their own decisions. Due to the MAS possibilities to handle the complex problems flexibility, it attracts researcher's attention. So far, MAS suitable to be exploited in two ways: as an approach to construct flexible, robust and extensible hardware/software systems; and also appropriate as a modeling approach. In this design according to MAS technology, allows us to apportion the problem into sub-problems. Each sub-problem is allocated for a specific agent.

B. Proposed MAS Architecture

Fig. 2 is shown the proposed MAS architecture for a yard-crane-relay system which all automated container throughout the word adopted this type of container flow system (like ECT terminal of Rotterdam and the CTA terminal in Hamburg) [18]. In such system, Container Ship Agent (CSA) is examined allocating berth space for vessels problem in container terminal at the sea side. Quay cranes are handling container from/to berthed ship to/from Automated Guided Vehicles (AGVs). Quay cranes scheduling process, and sequence of handling containers are being managed by Quay Crane Agent (QCA) which is critical resource in port container terminal. AGVs are being used to transfer containers between Quay Crane (QC) and Automated Stack Crane (ASC) within a scheduling time managed by Vehicle Traffic Agent (VTA). Storage space may be pre-assigned in stack to have efficient loading/discharging operation. Management and assignment of containers in stack is used by Stack agent (SA) with communication to adjacent Stack Crane Agent (SCA) for handling containers from ship side to transshipment side. This is done usually after certain storage period. At the transshipment side, after retrieving containers from the stack by SCA, another Vehicle Traffic Agent (VTA) manages other vehicles like bargs, vessels, trucks or trains to transship containers. To design the system as MAS architecture, the system has been divided into its main tasks as shown in Fig. 2. These agents are mainly characterized by their independence from the rest of the system elements. Agents are able to coordinate and to communicate with each other in order to make decision for their controlled area. This architecture consists of several Quay Crane Agents (QCAs) that each crane will be controlled and managed by a QCA [3], [11].

Many researchers have been concentrating on berth, quay cranes, transporting and traffic management, stacking, and transshipment planning; however, recent trends toward automating in maritime logistics, like emerging of mega-containerships and automated vehicles have created some gaps. Lack of container and workforce management, fleet and real-time data availability, scheduling and process automation, terminal safety and security are the examples of these gaps. These issues are become basically due to the conventional terminal management system. These gaps will shrink by implementation of terminal automation technology in two key focus points; terminal efficiency and terminal safety/security. Focusing primarily on accuracy, availability of data, and intelligence of field devices to reach an upgraded terminal control system interface and algorithms can also make a significant impact on terminal efficiency. Also focusing on alarms, interlocks, and uncertainties, which play a meaningful role in mitigating risks and safeguarding against major occurrences, bring more facilities to optimize the performance of the system. In the proposed hierarchical MAS architecture, the above two main issues for container terminal automation system is considered. Firstly, the agents control and optimize their own environment. In this way, they can improve terminal efficiency as a distributed manner with new algorithms and their intelligences. Secondly, in order to improve safety and security of the terminal automation system, we introduce three essential layers from automatic control point of view in decision making and planning named: *Supervisory*, *Scheduling*, and *Operational*. These layers assist system to have a safe and secure data communication and storage, also user/agent accessibility can be.

1- At the *Supervisory layer*, the flow of operational performances and the way of handling containers are defined by incoming information of arrival containerships. The decision time horizon covers long-term periods, i.e. from several months up to several years. This layer also provides sets of constraints under which the decisions at the lower layers. These constraints are forced agents at scheduling layer to control its environment at operational layer in order to meet supervisory layer goals. These goals are typically include time management for transship containers via handling equipment from ships to barges, trucks, and train or vice versa.

2- At the *Scheduling layer*, it is decided which type of information that is received from the supervisory and the operational layers, is used in order to sequence of choices have to be made. For example, which order should be used

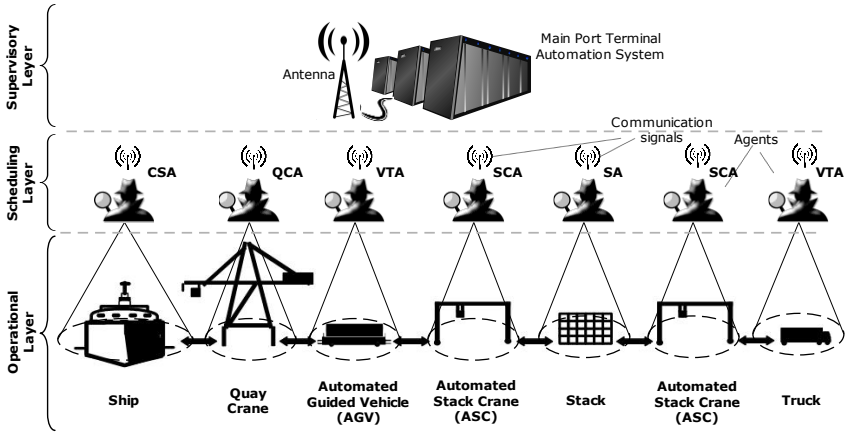


Fig. 2 – Proposed MAS architecture for port container automation system

for handling a set of containers from ship to berth by quay cranes sequentially? Or which ways of storing containers in a couple of a stack should be selected? The time horizon of this level for making decisions covers a day to months.

3- At the *Operational layer*, various equipments and vehicles should be used in order to satisfy overall required strategies for automation system like reduce dwell time of containerships at yard. Also, some sort of detailed problems like, where certain containers should be stored or handled, are solved.

In the following sections, we will study what potential QCAs have for solving a quay crane scheduling problem cooperatively in a distributed manner. In the rest of this paper, our focus is on the scheduling level as we discussed in this section. The interactions between QCAs and the port automation system in the supervisory level, QCAs and other adjacent agents in the scheduling level, and QCAs and their pair quay cranes in the operational level will be shown.

III. QUAY CRANE SCHEDULING PROBLEM

The problem of scheduling quay cranes known as *Quay Crane Scheduling Problem (QCSP)* was first addressed by [4] who provided a mixed integer programming (MIP) formulation for it, and then this problem has been studied in many different settings. The authors also proposed a branch-and-bound algorithm for this problem in [5]. A mixed-integer model, an exact algorithm and a heuristic for the QCSP was studied in [6]. QCSP with spatial non-crossing constraints (i.e., crane arms cannot be crossed over each other simultaneously) was investigated in [7], [19]. In [20], the QCSP with non-interference constraints was studied. The interference between quay cranes in that quay cranes cannot cross over each other because they are on the same track. In practice only one QC can move to another hold until it completes the current hold one.

In this paper, we focus on a QCSP for a given berth schedule by giving the MIP formulation first.This is a centralized formulation. We will solve this mathematical formulation by CPLEX for all cranes as a centralized approach. Since we intend to use distributed approach, the problem is decomposed into its sub-problems. This is done by QCAs and finally, implementation will be performed as a distributed constraint programming with a proposed negotiation algorithm which enables agents to solve the QCSP distributively in a cooperation manner. In fact, each QCA has ability to solve partial QCSP by using CPLEX CP Solver, also it has ability to negotiate with other QCA by using proposed algorithm based on ABT algorithm. This procedure will describe in Section VI.

A. Problem Description

The aim of this formulation of QCSP is to determine sequence of unloading and loading operations of a QC will carry out regarding to minimize completion time of ship operation. Fig. 3, shows the stowage plan for a ship, a collection of slots that are located at nearest neighbor to each other are usually containers of the same group (the same group containers usually are referring to inbound/outbound containers with the same loading/destination port, of the same size, and to be discharged from/loaded to the same ship). In the stowage plan for loading, a cluster is defined to be a collection of adjacent slots into which containers of the same group are planned to be loaded. In the stowage plan for discharging, a cluster implies a collection of adjacent slots in which inbound containers of the same group are stacked.

This plan (Fig. 3) consists of four cross-sectional views, each corresponding to a ship-bay and labeled with an odd number from 1 to 7. Each small square represents a slot. Shaded squares correspond to slots that containers must be discharged from or loaded onto in this container terminals. The shaded pattern in each slot represents a specific group of containers to be loaded into or picked up from the corresponding slots. Fig. 3 shows that four groups of containers should be discharged from five clusters of slots and then four groups of containers should be loaded into five clusters of slots.

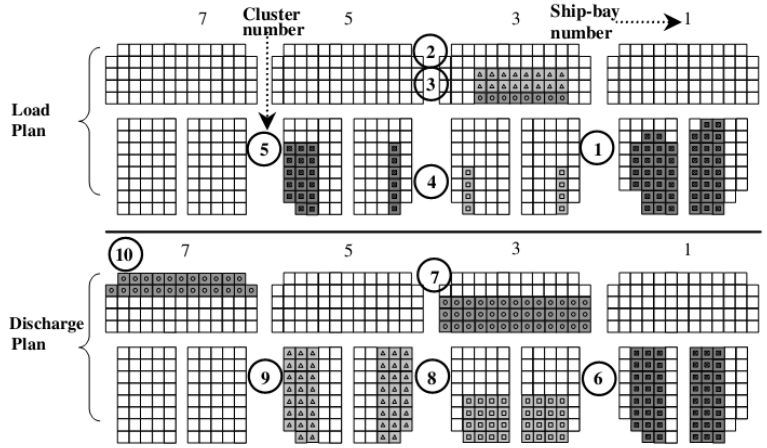


Fig. 3 - A partial example of a stowage plan [6]

Commented [Z2]:
Note:
I would tell that, this formulation originally proposed by Kim and Park that I have not accessed to their article when I wrote it, and later I received from you.
I found [21] article for Sammarra et al. that they used Kim and Park formulation and unfortunately their descriptions are incomplete or somehow incorrect.
I re-check it and try to describe it more from original one.

Commented [Z3]:
This description is from Kim and Park formulation [6]

Commented [Z4]:
Note:
I would not like this plan for QCSP if you are agree with me.
Because:

1. This plan consists of three main problems instead of just QCSP. The problems are:
   a.Stacking/ Storage allocation Problem
   b.AGV Scheduling problem
   c.QCSP

2.This plan has many uncertainties and scheduling will not be occurred in the predicted time. Since we decrease the main three problems into only one. So, this scheduling will not be a precise scheduling in my opinion although their formulation is right and detailed.

Formatted: Condensed by 0.05 pt

Quay Crane Schedule

QC 1 (operation time: 09:00 ~ 12:00)							QC 2 (operation time: 09:00 ~ 12:00)						
Operation sequence	Cluster number	Location of task	Type of task	Number of containers	Start time	Finish time	Operation sequence	Cluster number	Location of task	Type of task	Number of containers	Start time	Finish time
1	6	1 Hold*	D**	47	09:00	09:47	**1**	7	3 Deck	D	39	09:00	09:39
2	1	1 Hold	L**	42	09:47	10:29	**2**	9	5 Hold	D	46	09:41	10:26
3	8	3 Hold	D	32	10:31	11:03	**3**	5	5 Hold	L	23	10:26	10:49
4	4	3 Hold	L	8	11:03	11:11	**4**	10	7 Deck	D	24	10:51	11:14
5	3	3 Deck	L	8	11:11	11:19							
6	2	3 Deck	L	16	11:19	11:35							

* The hold of ship-bay 1.
** D (discharging), L (loading).

Fig. 4 – An example of QCs scheduling [6]

This paper defines a ''task'' as a discharging or loading operation for a cluster. This paper assumes that once a QC starts to load (or discharge) containers into (from) a cluster of slots, it continues to do so until all the slots in the cluster become filled (empty). Therefore, this paper considers handling work for a cluster to be a task.

However, the QC scheduling problem has several unique characteristics that are different from those of a typical m-parallel machine problem. For example, when discharging and loading operations must be performed at the same ship-bay, the dis-charging operation must precede the loading operation. When a discharging operation is performed in a ship-bay, tasks on a deck must be performed before tasks in the hold of the same ship-bay are performed. Also, the loading operation in a hold must precede the loading operation on the deck of the same ship-bay. Thus, there are precedence relationships among clusters, relationships that must be observed during a ship operation. Also, it should be noted that QCs travel on the same track. Thus, certain pairs of tasks cannot be performed simultaneously when the locations of the two clusters corresponding to the tasks are too close to each other, because two adjacent QCs must be apart from each other by at least one ship-bay so that they can simultaneously perform their tasks without interference. Also, if containers for any two tasks must be picked up at or delivered to the same location in a yard, the two tasks may not be performed simultaneously, because doing so will cause interference among yard cranes that transfer containers corresponding to the two tasks.

Fig. 4 illustrates a QC schedule that shows the number of containers in each cluster, the sequence of tasks to be performed, and the time schedule for performing the tasks.. lll

If we just assume each group task (cluster) as ~~just~~ one task (container hold to be handled by a crane), we can use this formulation with some modification as a formulation of our problem which we will try to solve it by OPL with CPLEX as centralized one and by DCOP formalization as distributed one.

These modifications are limited to defining overlapping and non-overlapping sets that indicate which tasks will hold by which quay crane/cranes, and defining some new constraints over those sets to ban handling a task by another cranes.

~~A.~~*B. Notation of Mathematical Model for single-agent QCSP*

The mathematical model of QCSP introduced in [21] as a Mixed-Integer Model (MIP) based on the developed model in [6], including the modifications report in [22].

The main constraints of the scheduling operation can be described as below:

1. Each QC can operate after its earliest available time.
2. QCs are on the same track and thus cannot cross over each other.
3. Some tasks must be performed before others.
4. There are some tasks that cannot be performed simultaneously.

A set of tasks (Here tasks are referred to containers to be handled) $\Omega = \{1, \dots, n\}$ and a set of crane $Q = \{1, \dots, q\}$ is given. The complete set of tasks is $\bar{\Omega} = \Omega \cup \{0, T\}$, where 0 and T represent *first* and *last* ~~tasks performed~~ by states of each crane. For example, if task i is the first task of QC k, $X_{0i}^{k} = 1$. Also, when task j is being the last task of QC k, $X_{jT}^{k} = 1$. p_i is defined as the relative processing time of task $i \in \bar{\Omega}$; we know also $p_0 = p_T = 0$ clearly. Furthermore, $l_i \in \mathbb{Z}^+$represents the location of task $i \in \Omega$. Set $\Phi = \{(i,j) | i,j \in \Omega\}$ expressed the precedence relationships between pairs of tasks. Ψ is the set of pair tasks which cannot be performed at the same time, ~~clearly $\Phi \subset \Psi$~~.

a) Indices

i, j , Tasks to be performed

Commented [Z5]:
Note:
The definition of task is quite different of my assumption either other QCSP literature.
Fortunately as far as I revise the formulation and constraints again, I found that we can use this definition also and will adapt of our definition in OPL.
This def. consider task is a set of group containers that adjacent to each other like as Fig. 3.
If we consider that each "container hold" be a task (same as our def. in next section), and we define adjacent holds in Φ and Ψ sets, we will see that the number of tasks are increased and

Formatted: Condensed by 0.05 pt

Commented [Z6]:
My last opinion

Commented [Z7]:
RN: So, tasks 0 & T Are done by multiple cranes?

IZ: added

Commented [Z8]:
RN: physical location?

IZ: No, this is the location expressed by ship bay number and mostly regarding to the position of the task on port.
~~Yes I mean physical location on ship. Is there any location unless physical location?~~

Commented [Z9]:
RN: what do you mean when do you have such relationship?

IZ: this set Φ is defined whether task i should be handled before task j or not. This is useful when we want to prioritize some tasks over some other tasks.

Commented [Z10]:
RN: where is that the case?

IZ: On the other hand set Ψ defines whether two pairs of tasks can be handled simultaneously or not. This is useful when we want to emphasize that for e.g. two tasks should be handled by one crane and they cannot be handled at the same time.

Commented [Z11]:
RN: Correct?

IZ: I don't think so this is correct!

k, QCs where $k = 1, \dots, K$

b) Problem Data

p_i The time required to perform task i

r_k The earliest available time of QC k

l_i The location of task (expressed by the ship bay number)

l_k^0 The starting position of QC k

l_k^T The final position of QC k

t_{ij} The travel time of a QC from location (l_i) of task i to location (l_j) of task j

M A sufficient large constant

α_1 The weight of the makespan (the maximum completion time)

α_2 The weight for total cranes completion time

c) Sets of indices

Ω The set of all tasks

Ψ The set of pairs of tasks cannot to be performed simultaneously

Φ The set of ordered pairs of tasks with precedence relationship between them

d) Decision variables

X_{ij}^k 1 if QC k performs task j right after task i; 0 otherwise

Y_k The completion time of QC k

D_i The completion time of task i

Z_{ij} 1 if task j starts later than the completion time of task i; 0 otherwise

W Time at which all tasks are completed

e) Objective

$$minmize\ \alpha_1 W + \alpha_2 \sum_{k=1}^{K} Y_k \tag{1}$$

f) Cosntraints:

$$Y_k \le W \qquad \forall k = 1, \dots, K, \tag{2}$$

$$\sum_{j \in \Omega} X_{0j}^k = 1 \qquad \forall k = 1, \dots, K, \tag{3}$$

$$\sum_{i \in \Omega} X_{iT}^k = 1 \qquad \forall k = 1, \dots, K, \tag{4}$$

$$\sum_{k} \sum_{i \in \Omega} X_{ij}^k = 1 \qquad \forall j \in \Omega, \tag{5}$$

$$\sum_{j \in \Omega} X_{ij}^k - \sum_{j \in \Omega} X_{ji}^k = 0 \qquad \forall i \in \Omega,\ \forall k = 1, \dots, K, \tag{6}$$

$$D_i + t_{ij} + p_j - D_j \le M(1 - X_{ij}^k) \qquad \forall i, j \in \Omega,\ \forall k = 1, \dots, K, \tag{7}$$

$$D_i + p_j \le D_j \qquad \forall (i, j) \in \Phi, \tag{8}$$

$$D_i - D_j + p_j \le M(1 - Z_{ij}) \qquad \forall i,j \in \Omega, \qquad (9)$$

$$Z_{ij} + Z_{ji} = 1 \qquad \forall (i,j) \in \Psi, \qquad (10)$$

$$\sum_{v=1}^{k} \sum_{u \in \Omega} X_{uj}^{v} - \sum_{v=1}^{k} \sum_{u \in \Omega} X_{ui}^{v} \le M(Z_{ij} + Z_{ji}) \qquad \forall i,j \in \Omega,\ l_i < l_j,\ \forall k = 1, \dots, K, \qquad (11)$$

$$D_j + t_{jT}^{k} - Y_k \le M\left(1 - X_{jT}^{k}\right) \qquad \forall j \in \Omega, \forall k = 1, \dots, K, \qquad (12)$$

$$r_k - D_j + t_{0j}^{k} + p_j \le M\left(1 - X_{0j}^{k}\right) \qquad \forall j \in \Omega, \forall k = 1, \dots, K, \qquad (13)$$

$$X_{ij}^{k}, Z_{ij} = 0 \text{ or } 1 \qquad \forall i,j \in \Omega, \forall k = 1, \dots, K, \qquad (14)$$

$$Y_k, D_i \ge 0 \qquad \forall i \in \Omega, \forall k = 1, \dots, K, \qquad (15)$$

In this model the objective function (1) is a weighted summation of the makespan (2) and sum of the crane completion times. Minimizing the ship completion time, minimizes the makespan and maximizes their productivity, since the idle time of each crane is kept as little as possible. Note that, the problem under consideration is a multiobjective optimization problem. The weighted coefficients can be assumed $\alpha_1 \gg \alpha_2$ according to [6], since the primary objective is minimizing the makespan. The constraints (3), (4) and (6) are the classical routing constraints; indicate sequence of performing tasks by each crane. Constraints (3) and (4) respectively select the first and last tasks for each QC. Constraint (5) ensures that each task is performed by one and only one crane. Constraint (6) is a flow balance constraint, guaranteeing that tasks are performed in well-defined sequences. Constraints (7) and (8) calculate the completion times. Constraint (7) simultaneously determines the completion time for each task and eliminates sub-tour. When required, constraint (8) denotes that task i should be completed before task j. Constraint (9) defines z_{ij} variable and prevent crane interference. Constraint (10) states that tasks i and j cannot be handled at the same time if $(i,j) \in \Psi$. Constraint (11) avoids interferences cranes jobs and the overtaking between cranes. The correct completion times for the last and the first task on each crane are computed through (12) and (13). Finally, constraints (14) and (15) indicate natural numbers that variables have to be taken.

Commented [Z12]:
RN: Is there a set that define the overlapping tasks between quay cranes?

IZ: added in th following section.

~~IZ: No, this formulation is a simpler version of our formulation that we implement with OPL later in this section.~~
~~Maybe we can consider this set or maybe we can index the existing sets. E.g Ω could be Ω_i for each crane i~~
~~Furthermore it may not be as easily as we think.~~
~~In this formulation, they assume all the tasks are shared with all the cranes~~
~~Also they did not consider cranes crossing over constraints.~~

MY MODIFICATION

~~The "tasks (T)" are referred to each container hold to be handled by a quay crane, the set of all tasks can be written as $\Omega = \{1,2,\dots,n\}$, the total number of tasks is n). We can define also a set of m "group tasks (GT)", $\Omega = \{\Omega_1, \Omega_2, \dots, \Omega_m\}$ which Ω_i $(1 \le i \le m)$ is a set of GT with n_i tasks, which its tasks are adjacent to each other and should be handled consecutively. We can easily show that $\sum_{i=1}^{m} n_i = n_1 + n_2 + \cdots + n_m = n$. A set of crane $Q = \{1, \dots, K\}$ is given. We define decision variable X_{ij}^{k}, which will be 1 if QC k performs GT j right after GT i; 0 otherwise. In the original QCSP problem proposed by Kim and Park [6], the X variable is $m \times m$ matrix for each crane for m group tasks (GTs) which 0 and T position state of cranes are appended to that. For each crane k we will have:~~

$$X_{ij}^{k} = \begin{array}{c|cccc} & 0 & 1 & \cdots & m & T \\ \hline 0 & X_{00} & X_{01} & \cdots & X_{0m} & X_{0T} \\ 1 & X_{10} & X_{11} & \cdots & X_{1m} & X_{1T} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ m & X_{m0} & X_{m1} & \cdots & X_{mm} & X_{mT} \\ T & X_{T0} & X_{T1} & \cdots & X_{Tm} & X_{TT} \end{array}$$

~~With our modification, we can re-write the above formulation as follows:~~

$$X_{ij}^{k} = \begin{array}{c|cccc} & \Omega_0 & \Omega_1 & \cdots & \Omega_m & \Omega_T \\ \hline \Omega_0 & X_{\Omega_0\Omega_0} & X_{\Omega_0\Omega_1} & \cdots & X_{\Omega_0\Omega_m} & X_{\Omega_0\Omega_T} \\ \Omega_1 & X_{\Omega_1\Omega_0} & X_{\Omega_1\Omega_1} & \cdots & X_{\Omega_1\Omega_m} & X_{\Omega_1\Omega_T} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \Omega_m & X_{\Omega_m\Omega_0} & X_{\Omega_m\Omega_1} & \cdots & X_{\Omega_m\Omega_m} & X_{\Omega_m\Omega_T} \\ \Omega_T & X_{\Omega_T\Omega_0} & X_{\Omega_T\Omega_1} & \cdots & X_{\Omega_T\Omega_m} & X_{\Omega_T\Omega_T} \end{array}$$

~~That each $X_{\Omega_i\Omega_j}$ is a block of matrix $n_i \times n_j$. Obviously, $\Omega_0 = 0$ and $\Omega_T = T$ are scalar regarding to initial and final state of crane k. Note that we can write each set sequentially as follows:~~

~~$\Omega_1 = \{c_1, \dots, d_1\}, \quad c_1 = 1, \quad d_1 = n_1,$~~

~~$\Omega_2 = \{c_2, \dots, d_2\}, \quad c_2 = d_1 + 1 \quad d_2 = d_1 + n_2,$~~

~~$\vdots$~~

~~$\Omega_m = \{c_m, \dots, d_m\}, \quad c_m = d_{m-1} + 1, \quad d_m = d_{m-1} + n_m,$~~

~~Where, the value of c_{in} and d_{in} can be derive from a recurrence relation, $c_i = d_{i-1} + 1$ and $d_i = d_{i-1} + n_i$. So, $c_i = n_1 + n_2 + \cdots$~~

The mathematical model of QCSP introduced in [21] as a Mixed-Integer Model (MIP) based on the developed model in [6], including the modifications report in [22].

The main constraints of the scheduling operation can be described as below:

1. Each QC can operate after its earliest available time.
2. QCs are on the same track and thus cannot cross over each other (cross over constraint / non-interference).
3. Some tasks must be performed before others (precedence relationship).
4. There are some tasks that cannot be performed simultaneously (non-preemptive).
5. Some tasks must be performed by one and only one QC (non-overlapping tasks).
6. Some tasks must be performed by one of two adjacent QCs (overlapping tasks).
7. Some tasks must not be performed by some QCs.
8. Spatial constraints for QCs did not seen in this formulation.

A set of tasks (Here tasks are referred to containers to be handled) $\Omega = \{1, \dots, n\}$ and a set of crane $Q = \{1, \dots, q\}$ is given. The complete set of tasks is $\bar{\Omega} = \Omega \cup \{0, T\}$, where 0 and T represent *first* and *last* states of each crane. For example, if task i is the first task of QC k, $X_{0i}^k = 1$. Also, when task j is being the last task of QC k, $X_{jT}^k = 1$. p_i is defined as the relative processing time of task $i \in \bar{\Omega}$; we know also $p_0 = p_T = 0$ clearly. Furthermore, $l_i \in \mathbb{Z}^+$ represents the location of task $i \in \Omega$. Set $\Phi = \{(i,j) | i,j \in \Omega\}$ expressed the precedence relationships between pairs of tasks. Ψ is the set of pair tasks which cannot be performed at the same time,.

We can define set of quay crane allocations Δ that consists of allocation of all tasks regarding to the cranes. The set $\Delta = \{\Delta_1, \dots, \Delta_K\}$ is defined regarding to the number or quay cranes. For example, Δ_n indicates which tasks should be handled by QCn. Clearly $\Delta_n \subseteq \Omega \subseteq \Delta$. Note that, we can share a task to maximum two adjacent cranes which indicates that the position of the tasks is between the two adjacent cranes and can be handled by one of them. For example, if we define task $1 \in \Delta_1$, this means that task 1 should be handled just by QC1. And if we define task $5 \in \Delta_1, \Delta_2 \{\sout{\Delta_1, \Delta_2}\}$, this means that task 5 should be performed only by one of QC1 or QC2. Note that, in the both example, we implicitly define that the tasks will not be performed by other QCs.

With respect to the QC allocation set Δ, we can define overlapping set $\bar{\Delta}$ and non-overlapping set $\tilde{\Delta}$. $\bar{\Delta} = \{\bar{\Delta}_{1,2}, \bar{\Delta}_{2,3}, \dots, \bar{\Delta}_{k-1,k}\}$, where $\bar{\Delta}_{n,m} = \{\Delta_n \cap \Delta_m\}$ and $m - n = 1$, $m = \{2,3, \dots, k\}, n = \{1,2, \dots, k-1\}$. Also, $\tilde{\Delta} = \{\tilde{\Delta}_1, \tilde{\Delta}_2, \dots, \tilde{\Delta}_k\}$, where $\tilde{\Delta}_n = \left\{\left\{\{\bar{\Delta}_{n-1,n}\}^c \cap \Delta_n\right\} \cup \left\{\{\bar{\Delta}_{n,n+1}\}^c \cap \Delta_n\right\}\right\}$ ~~$\tilde{\Delta}_n = \left\{\left\{\{\bar{\Delta}_{n,n-1} \cap \Delta_n\}^c \cap \Delta_n\right\} \cup \left\{\Delta_n \cap \{\Delta_n \cap \bar{\Delta}_{n,n+1}\}^c\right\}\right\}$~~, $n =$ and $\{\dots\}^c$ means complement of set $\{\dots\}$.

~~Now instead of using set Ω which indicate all the tasks, we can use both set Δ sets Δ_n which indicate all tasks for specific QC n.~~

a) Indices

i, j, Tasks to be performed

k, QCs where $k = 1, \dots, K$

b) Problem Data

p_i The time required to perform task i

r_k The earliest available time of QC k

l_i The location of task (expressed by the ship bay number)

l_k^0 The starting position of QC k

l_k^T The final position of QC k

t_{ij} The travel time of a QC from location (l_i) of task i to location (l_j) of task j

M A sufficiently large constant

α_1 The weight of the makespan (the maximum completion time)

α_2 The weight for total cranes completion time

c) Sets of indices

Commented [Z13]: This is my previous formulation that at last doesn't work and I back again to the original formulation. If we want to use it, we have to propose some new integer programming constraint.
This may have another function that does not use here.

Commented [Z14]: Note: Modified

Commented [Z15]: Note:
We will do exactly the same when proposing OPL model.

Commented [Z16]: Modified

Ω The set of all ~~group~~ tasks

~~Ω_i The set of each group tasks that should be performed consecutively $(1 \leq i \leq m)$~~

Φ The set of ordered pairs of tasks with precedence relationship between them

Δ The set of QC task allocations

Δ_n The subset of QC task allocations for QCn

X_{ij}^k 1 if QC k performs task j right after task i; 0 otherwise

Y_k The completion time of QC k

D_i The completion time of task i

Z_{ij} 1 if task j starts later than the completion time of task i; 0 otherwise

W Time at which all tasks are completed

~~g~~)e) *Objective*

$$minmize\ \alpha_1 W + \alpha_2 \sum_{k=1}^{K} Y_k \qquad (16)$$

~~h~~)f) *Cosntraints:*

$$Y_k \leq W \qquad \forall k = 1, \dots, K, \qquad (17)$$

$$\sum_{j \in \Delta_k} X_{0j}^k = 1 \qquad \forall k = 1, \dots, K, \qquad (18)$$

$$\sum_{i \in \Delta_k} X_{iT}^k = 1 \qquad \forall k = 1, \dots, K, \qquad (19)$$

$$\sum_{k} \sum_{i \in \Omega} X_{ij}^k = 1 \qquad \forall j \in \Omega, \qquad (20)$$

$$\sum_{j \in \Delta_k} X_{ij}^k - \sum_{j \in \Delta_k} X_{ji}^k = 0 \qquad \forall i \in \Delta_n, \quad n = 1, \dots, K, \quad \forall k = 1, \dots, K, \qquad (21)$$

$$D_i + t_{ij} + p_j - D_j \leq M(1 - X_{ij}^k) \qquad \forall i, j \in \Delta_k, \ \forall k = 1, \dots, K, \qquad (22)$$

$$D_i + p_j \leq D_j \qquad \forall (i,j) \in \Phi, \qquad (23)$$

$$D_i - D_j + p_j \leq M(1 - Z_{ij}) \qquad \forall i, j \in \Delta_k, \ \forall k = 1, \dots, K, \qquad (24)$$

$$Z_{ij} + Z_{ji} = 1 \qquad \forall (i,j) \in \Psi, \qquad (25)$$

$$\sum_{v=1}^{k} \sum_{u \in \Delta_n} X_{uj}^v - \sum_{v=1}^{k} \sum_{u \in \Delta_n} X_{ui}^v \leq M(Z_{ij} + Z_{ji}) \qquad \forall i, j \in \Delta_n, \ l_i < l_j, \ \forall n = 1, \dots, K, \ \forall k = 1, \dots, K, \qquad (26)$$

$$D_j + t_{jT}^k - Y_k \leq M(1 - X_{jT}^k) \qquad \forall j \in \Delta_k, \forall k = 1, \dots, K, \qquad (27)$$

$$r_k - D_j + t_{0j}^k + p_j \leq M(1 - X_{0j}^k) \qquad \forall j \in \Delta_k, \forall k = 1, \dots, K, \qquad (28)$$

$$X_{ij}^k, Z_{ij} = 0 \text{ or } 1 \qquad \forall i, j \in \Omega, \forall k = 1, \dots, K, \qquad (29)$$

$$Y_k, D_i \geq 0 \qquad \forall i \in \Omega, \forall k = 1, \dots, K, \qquad (30)$$

In this model the objective function (16) is a weighted summation of the makespan (17) and sum of the crane completion times. Minimizing the ship completion time, minimizes the makespan and maximizes their productivity, since the idle time of

Commented [Z17]:
Note:
Since in the modification constraints we use sets Δ_k contain some of tasks instead of set Ω contains all tasks, ~~the performance of search and finding solution will increase and finding solution will be more rapid.~~
~~In other word, searching through Δ_x, will eliminate many other possible selections through Ω that are irrelevant.~~
~~The search performance will decrease since the tasks of search space within the set of $\Delta = \{\Delta_1, \dots, \Delta_x\}$ is larger than all tasks of set Ω which was the search space. So, the time require to reach solution naturally will be increased.~~

In some constraints, the proposed QC task allocation set $\Delta_n \subseteq \Omega$ can decrease search time and increase optimization performance in comparison with Kim and Park formulation (e.g. (18), (19), (22), (24), (27), (28)), while in some other constraints since $\Omega \subseteq \Delta$ and Δ may have more tasks than Ω the search performance may decrease (e.g. (21), (26)).

Commented [Z18]:
Note:
We will use "initial transition time" in OPL model in 4th scenario by using "append" function, same as this constraint.

Commented [Z19]:
Note:
This constraint exactly same as function "alternative" in OPL model.
In D-QCSP, since we select each overlapping task in each step by each agent (QCA) and inform the assignment to neighbor QCAs, we will not assign a task more than one time same as this constraint.

Commented [Z20]:
Note:
We will use "noOverlap" function in OPL model exactly same as this constraint to avoid performing two sequential tasks simultaneously.
The D-QCSP also we use the OPL model in every iteration and use this constraint. (The OPL model for D-QCSP actually is more simple than the OPL used in the next section and used just for a single crane to derive the correct sequencing of the tasks that should perform by each QCA)

Commented [Z21]:
Note:
In D-QCSP, by defining the string form of tasks between two adjacent cranes regarding to the position and distance of the tasks, we will do same as this constraint and won't permit adjacent QCs to cross over each other.

each crane is kept as little as possible. Note that, the problem under consideration is a multiobjective optimization problem. The weighted coefficients can be assumed $\alpha_1 \gg \alpha_2$ according to [6], since the primary objective is minimizing the makespan. The constraints (18), (19) and (21) are the classical routing constraints; indicate sequence of performing tasks upon each QC task allocation set Δ_k. Constraints (18) and (19) respectively select the first and last tasks for each QC. Constraint (20) ensures that each task within set Ω is performed by one and only one crane. Constraint (21) is a flow balance constraint, guaranteeing that tasks are performed in well-defined sequences by each QC. Constraints (22) and (23) calculate the completion times. Constraint (22) simultaneously determines the completion time for each task and eliminates sub-tour. When required, constraint (23) denotes that task i should be completed before task j. Constraint (24) defines z_{ij} variable upon each pair tasks performed by a QC and prevent crane interference. Constraint (25) states that tasks i and j cannot be handled at the same time if $(i,j) \in \Psi$. Constraint (26) avoids interferences cranes jobs and the overtaking between two adjacent cranes. Suppose that different tasks $i,j \in \{\Delta_{k_1}, \Delta_{k_2}\}$ which $l_i < l_j$ are performed simultaneously. This means that $Z_{ij} + Z_{ji} = 0$. Note that both QCs and tasks are ordered in an increasing order of their relative locations in the direction of increasing ship-bay number. Now imagine, QCk_1 performs task j and QCk_2 performs task i, where $k_1 < k_2$. However, in such a case for $k = k_1$ and $\Delta_n = \Delta_{k_1} or\ \Delta_{k_2}$, $\sum_{v=1}^{k_1}\sum_{u\in\Delta_{k_1}} X_{uj}^{v} - \sum_{v=1}^{k_1}\sum_{u\in\Delta_{k_1}} X_{ui}^{v} \le M(Z_{ij} + Z_{ji})$, which cannot be allowed by this constraint. The correct completion times for the last and the first task on each crane are computed through (27) and (28). Finally, constraints (29) and (30) indicate natural numbers that variables have to be taken.

B.C. Model Transformation Description

We can use the MIP model properties like as "Decision variables", "Objective function", "Constraints", and some other useful data for transforming the problem into the CPLEX solver. CPLEX CP Optimizer is a Constraint Programming (CP) system designed for an easy-to-use "model and run" approach. To solve QCSP, a model transformation is needed in order to use CPLEX as a scheduling problem solver by using CP optimizer. A major benefit of the CPLEX CP Optimizer approach to scheduling is that no enumeration of time is required. That means we just use *intervals*, which have duration in general, defining start and end time is not mandatory, so by using particular constraints and defining general relationships between them, we can optimize the overall problem regarding to objectives. This means that relatively few decision variables are needed for using an OPL scheduler model in comparison with a Mathematical Programming (MP) approach that would require variables for each time interval of a discretized model. Therefore, models can be formulated and solved efficiently, regardless of the relevant unit of time to describe a scheduling problem [23]. CPLEX CP Optimizer addresses scheduling problems by assigning start and end times to intervals during which, for example, a task occurs, while effectively managing resource constraints over time and alternative modes to perform a task.

The QCSP problem is depicted in Fig. 5. There is a containership on one side which should load/unload with containers and the container carrier tracks i.e. AGV tracks on the other side when an unloaded container place on them in a specific place. In this paper, we assume each crane has overlapped area with its adjacent crane. The overlapped area consists of several container columns (holds) which the nearby cranes can handle them without preemption. Handling the overlapping tasks depends on each crane duty and amount of tasks. We can define an index R which shows the ratio between overlapped tasks and all tasks as follows:

$$R_{i,j} = \frac{overlapping\ tasks\ for\ a\ given\ QCi\ and\ QCj}{all\ tasks\ for\ a\ given\ QCi} \qquad (31)$$

Overlapping index $R_{i,j}$ indicates how much the neighbor QCs i and j have interact with each other. Note that, $R_{i,j} \neq R_{j,i}$, since the former indicates the ratio of the overlapping tasks between cranes i and j to all tasks of crane i, and the latter one indicates the ratio of the overlapping tasks between cranes i and j to all tasks of crane j. Moreover, each crane regarding to the number of its neighbors may have one or two overlapping index (R). This parameter varies between 0 and 1, so bigger R shows higher interactions between two adjacent QCs. We will study how varying R could affect optimizing the problem as a distributed scheduling problem co-operatively among cranes. The distributed QCSP optimization is performed via CPLEX CP Optimizer, which provides access to Optimization Programming Language (OPL) IDE to build scheduling problem model. OPL provides an efficient combinatorial optimization that simplifies the formulation of model and produce substantially simpler and shorter code [24].

Commented [Z22]:
RN: R_j? / On both side of a crane?

IZ: I have modified the formulation of overlapping index without effect of rest of article **(except simulation section)** This is true since I was assuming the same overlapping tasks for each quay crane (same overlapping tasks in their overlapping area). Even I wrote code in OPL model, I defined same number of overlapping tasks for each cranes.
Now with this new formulation we can adjust them manually and consider different number of overlapping tasks for different quay crane and both different quay crane's sides. (Our solution model is generalized)

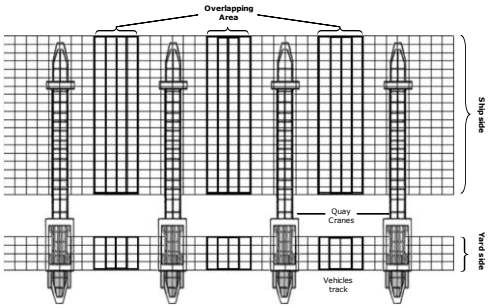


Fig. 5 – Terminal layout shows ship, quay cranes, and overlapped area of containers

~~C.~~D. *Problem Solving Description*

1) Step 1: Transforming the mathematical crane scheduling problem model into a OPL model solving by CPLEX Scheduler

Usually in constraint programming (CP) and specially in scheduling programming (SP) we are dealing with mathematical formulation. On the other hand, when we want to use IBM ILOG we are facing with different types of programming language referred to LP, MP, CP or CLP in Scheduler. In this case we should transform our mathematical model to an optimization concept and then convert it to syntax and codes where defined for the specific environment of optimization as the "optimization programming language". Regarding to this brief introduction, we proposed the mathematical model of crane scheduling problem for unloading containers from ships to port as follows:

We can assume each crane has its own task window which can do its assigned particular job (See Fig. 6). Each adjacent crane task window could have disjunctive (non-overlapping) or overlapping. In the case of overlapping windows, we can define a variable $R_{i,j}$ as in (31) which indicates ratio of overlapping tasks between QCi and QCj to all tasks of QCi (width of overlapped columns).

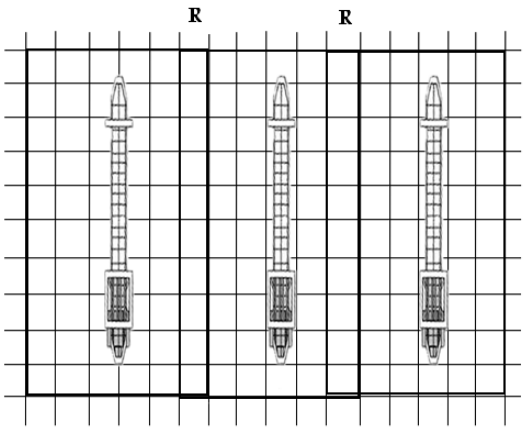


Fig. 6 – Cranes situation and their task windows

2) Step 2: Describing scheduling problem model with Scheduler

a) 1st scenario

First of all, we consider a crane with a window which has several container places. Each container place has its own *LocationID* number (same as l_i which expressed by ship bay number). Each task related to two arbitrary *LocationID* (l_i and $\acute{l}_i$) (See Fig. 7). For example we know task 1 is connecting to carry container from place3(l_i) to place75($\acute{l}_i$). The $\acute{l}_i$ is pseudo position for task i, indicates the destination position of task i.

Commented [Z23]:

IZ: I have brought this section from the "Report #2" which I had sent for you several months ago.
This section consists of modeling, programming, and results for a centralized QCSP with CPLEX.
I have obtained this procedure for the D-QCSP.
When we use this procedure in D-QCSP, the problem have decomposed to smaller one in comparison with centralized one. Therefore the differences can be described as follows:
- In D-QCSP we consider just one quay crane, while in centralized QCSP more than one QCSP should be considered
- In D-QCSP we are dealing with more less tasks for an individual quay crane to solve than centralized QCSP, so solving such a problem can be done concurrently and can take less time dramatically.

RN: But the performance will go down and communication is needed?

IZ: Why performance goes down?
Like as many other distributed solving problems, unfortunately communication cost is inevitable.

Commented [Z24]:
RN: You didn't use this in (1)-(15)

IZ: This is from my own word. Task window consists of overlapping and non-overlapping tasks.

Commented [Z25]:
Note Note Note:

Only this model (1st scenario) is used in our Extended ABT algorithm in D-QCSP section for scheduling all the allocated tasks for each QCA.
That means, each QCA tries to allocate tasks from the string ordered tasks. After that, each QCA tries to schedule its allocated tasks to find which sequence is better for handling assigned tasks.
For example, during each iteration each QCA has a set of assigned tasks, e.g. QCA1 has {1,2,6}, so he should find the best sequence to handling them according to each task l_i and $\acute{l}_i$ and their transition time matrix T.
In other word, by the prior knowledge, the QCA1 construct a T matrix with the allocated tasks such as:

$$T = \begin{bmatrix} t_{11} & t_{12} & t_{16} \\ t_{21} & t_{22} & t_{26} \\ t_{61} & t_{62} & t_{66} \end{bmatrix}$$

After that, with the proposed scheduling algorithm in this scenario the QCA1 connect to CPLEX from JAVA (by CPLEX api) and find the best way to handle the allocated tasks by its constructed OPL model in each iteration.

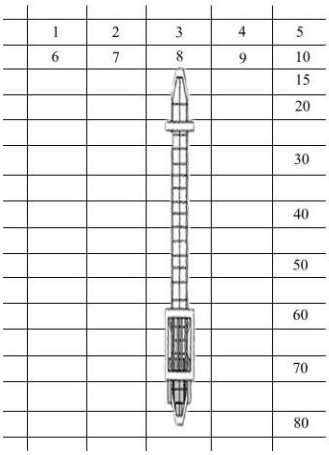


Fig. 7 – Depicting a typical quay crane with its task window (consists of its overlapping and non-overlapping tasks) and LocationID defined by ship bay number

We know the distance between each place and its traveling time to do it before. For simplicity task for each crane is defined:

$$\text{task } i = < location\ i\ (l_i), location\ i'(\acute{l}_i) > \quad (32)$$

Where, location i refers to source of container and location i' refers to destination of container which to be handled. For example, a container hold in place 6 should be handled to place 71, then containers in hold 4 should be handled to place 68 and so on. We can formulate the traveling time (t_{ij}) between adjacent task i and task j as follows:

$$t_{ij} = \begin{cases} \text{traveling time of empty crane from } \acute{l}_i \text{ to } l_j & \text{if } i \neq j \\ \text{loading time task } i \text{ at location } i\ (l_i) + & \\ \text{traveling time of loaded container from } l_i \text{ to } \acute{l}_i + & \text{if } i = j \\ \text{unloading time of container at } \acute{l}_i + & \end{cases} \quad (33)$$

Where, location j is referring to the next container place that is to be handled. Now for our model which has n tasks, we can define a $n \times n$ transition time matrix $T_{n\times n}$:

$$T = \begin{bmatrix} t_{11} & t_{12} & \cdots & t_{1n} \\ t_{21} & t_{22} & & \\ \vdots & & \ddots & \vdots \\ t_{n1} & & \cdots & t_{nn} \end{bmatrix} \quad (34)$$

Where, in general $T_{n\times n}$ is an asymmetric square matrix and its diagonal are non-zero. ~~(very small numbers indicate loading time of their relevant task).~~

Note, the loading time of container at l_i, traveling time of loaded container l_i to $\acute{l}_i$, and unloading time of container at $\acute{l}_i$, all the processing time of task i, and all can be assumed as actions for tasdenoted in section III.Bk and are identical to the time required to perform task i (p_i denoted in section III.B), defined in section III.B. In addition, the definition of traveling time of empty crane from task i to task j ($\acute{l}_i$ to l_j), is exactly the same as the definition of the QCSP explained in section III.B for t_{ij} and is assumed as elements of matrix T. Regarding to the above assumptions we can re-write (33) as follows:

$$t_{ij} = \begin{cases} t_{ij} & if\ i \neq j \\ p_i & if\ i = j \end{cases} \quad (35)$$

In the first simple simulation, 4 tasks with 1 machine are defined as follow. Also a 4×4 2-dimensional transition time matrix is considered. A tuple for transition between source and destination with duration from transition time matrix is defined in order to using "noOverlap" function. The optimization problem model is introduced in Fig. 8 as an OPL model.

Commented [Z26]:
RN: Why do you not explicitly include this in the formulation (1)-(15)?

IZ: added

Commented [Z28]:
RN: Integrate with (1)-(15)?
(use same fashion)

IZ: added

Commented [Z27]:
RN: Is this possible $i = j$?

IZ: changed

```
/*********************************************
 * OPL 12.2 Model
 * Author: Zabet
 * Creation Date: 2011/9/20 at 14:54:30
 *********************************************/
using CP;

int n = 4; // n tasks
int m = 1; // m machines
int T[1..n][1..n] = [ // T[i][j] gives duration of task i to task j
  [40,  50, 30, 40],
  [100, 50, 90, 80],
  [ 80, 90, 20, 70],
  [ 50, 45, 30, 45]
];

tuple triplet {
  int source;
  int destination;
  int distance;
};
{triplet} transitions = ...;
dvar interval task [i in 1..n] size T[i][i];
dvar interval alloc[i in 1..n][j in 1..m] optional;

dvar sequence resource
   in    all(i in 1..n) task[i] // Allocated task
   types all(i in 1..n) i;          // Task type (position)

minimize max(i in 1..n) endOf(task[i]);
subject to {
  noOverlap(resource, transitions, 1); // Unary resource with transition time M
}
```

Fig. 8 - OPL model of 1st scenario

In this scenario, we use generalized version of "Single Machine Scheduling Model, Minimizing Maximum Cost (The Maximum Lateness)" with transitional matrix for all tasks, i.e. in our formulation, the cost here shows the "transition time" of each task.

In this formulation, we do not see cross over constraints, precedence relationships between tasks, the initial and final states of QC in comparison with our modification (16)-(30). Also, all tasks must be performed by one crane (we have not overlapping tasks). So, the mathematical formulation will be:

Objective

$$minmize \ \sum W \tag{36}$$

Cosntraints:

$$\sum_{i \in \Omega} X_{ji} = 1 \qquad \forall j \in \Omega, \tag{37}$$

$$\sum_{j \in \Omega} X_{ij} - \sum_{j \in \Omega} X_{ji} = 0 \qquad \forall i \in \Omega, \tag{38}$$

Commented [Z29]:
Note
The diagonal of T matrix is changed with the OPL results.

Commented [Z30]:
RN: ?

IZ:
`transitions = {<i, j, T[i][j]> | i,j in 1..n};`
We can use "..." in OPL model for more legibility of model instead of using data.
So, the data of model is included in a .dat file. Even the T[n][n] was in the file.

Commented [Z31]:
RN: What does this mean?

IZ:
According to the last definition of T_{ij} we broke the transition time into T_{ij} *and* T_{ii}. That means, according to move the crane from its source to another destination. The crane first should try to load the task from its source T_{ii} in location i, then it should move the task from location i to location j, using the transition time T_{ij} from matrix T[1..n][1..n].
With the new definition of T_{ij} which is defined in the previous page the loading time is considered into T_{ij} $i \neq j$
And so T_{ij} $i = j$ (on diagonal) all will be Zero.

Commented [Z32]:
RN: ?

This is a OPL function which calculate the latest time of interval task[i].
It is exactly the makespan of completion time.

Commented [Z33]:
RN: ?

This is a OPL function. It gives a sequence (here is resource which is a sequence of tasks), and a tuple (here is triple tuple as "source, destination, distance" here is transition).
Then this function try to make all the feasible consecutive compound of sequences "source+distance+destination".
Using the constraint noOverlap, you can constrain that the interval sequence variable passed defines a chain of non-overlapping intervals that are present in the solution. If a set of transition tuples is specified, it defines the minimal time that must elapse between two intervals in the chain.

Commented [Z34]:
RN: What is the mathematical formulation of the problem being solved here?

IZ:
When I was writing OPL, my attention was toward various kind of "scheduling problems". I tried to find a way among various scheduling algorithm to solve our problem which have the same constraints. Before doing that, I tried to formulate rough problems which have less constraints and more simple. So, I started with single crane with some tasks with no precedence and non-preemptive in 1st scenario), then using 2 cranes with some tasks which are shared among all cranes with no cross over constraints, no precedence, and non-preemption (in 2nd scenario). After that, in 3rd scenario we assume overlapping and non-overlapping tasks which can be performed by specific QC(s). Finally in 4th scenario, we include initial and final states for QCs scheduling to reach more précises model to a real problem.
Note that, in none of them, we didn't considered cross-over constraints.
(Please see Table 1)

$$D_i + t_{ij} + p_j - D_j \leq M(1 - X_{ij}) \qquad \forall i,j \in \Omega, \qquad (39)$$

$$D_i + p_j \leq D_j \qquad \forall (i,j) \in \Phi, \qquad (40)$$

$$X_{ij} = 0 \; or \; 1 \qquad \forall i,j \in \Omega, \qquad (41)$$

$$D_i \geq 0 \qquad \forall i \in \Omega, \qquad (42)$$

Result:

The sequences of doing tasks with one machine same as the following Gantt chart. As we can see, the 1st, 3rd, 4th and 2nd tasks are allocated sequentially to the machine. The space between them is referred to transition time between tasks. Referring to the defined transition time matrix $t_{4\times4}$, we can obviously see that $t_{13} = 30$, $t_{34} = 70$, and $t_{42} = 45$. Also, the processing time (p_i) for all tasks are considered as $p_1 = t_{11} = 40$, $p_2 = t_{22} = 50$, $p_3 = t_{33} = 20$, and $p_4 = t_{44} = 45$. And the minimum completion time is concluded as sum of the above transition time i.e. $t_{11} + t_{13} + t_{33} + t_{34} + t_{44} + t_{42} + t_{22} = 300$ illustrated in Fig. 9 and Fig. 10.

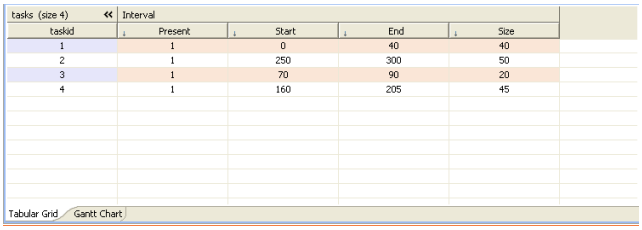


Fig. 9 – Result of the 1st scenario, indicates scheduling tasks (start time, stop time, and duration of tasks) for one machine

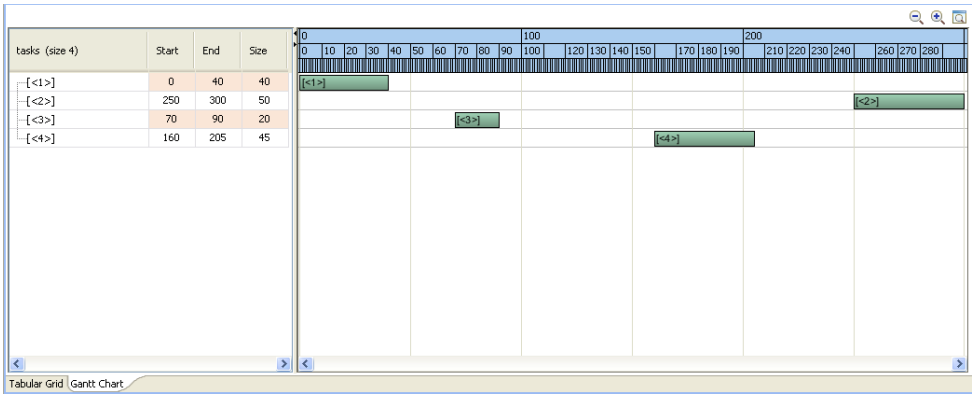


Fig. 10 – Gantt chart of the 1st scenario result

b) 2nd scenario

In the second simulation, 4 tasks for 2 machines are defined. We know each task should be done just with one machine once. So another useful function of scheduler i.e. "alternative" was used. Note that, using those function needs to see their usages in various problems and sometimes it is hard to get their concepts just with "OPL reference manual", due to lack of descriptions and examples. In this case, we assume each task allocates to all the machines but one of the machines can handle it. This case is known as a famous optimization problem called *"Flexible Job Shop scheduling Problem with one stage"*. The problem model is depicted in Fig. 11. Note that in this scenario, all tasks are shared with all machines, same as Kim and Park formulation [6].

In this scenario, we have no cross-over and precedence relationship constraints. Also, we have not consider initial and final state of QCs. The mathematical formulation according to our modification (16)-(30) will be:

Objective

Commented [Z35]: Updated

Commented [Z36]: Updated

Commented [Z37]: RN: if you change objective function, (e.g. priorities, ???ation) coefficients (for 1 or 2 tasks) do you get a different solution?

IZ: Sorry, but I cannot get your comment! Could you please explain it more?

$$minmize\ \alpha_1 W + \alpha_2 \sum_{k=1}^{K} Y_k \quad (43)$$

Cosntraints:

$$Y_k \le W \qquad \forall k = 1, \dots, K, \quad (44)$$

$$\sum_k \sum_{i \in \Omega} X_{ij}^k = 1 \qquad \forall j \in \Omega, \quad (45)$$

$$\sum_{j \in \Omega} X_{ij}^k - \sum_{j \in \Omega} X_{ji}^k = 0 \qquad \forall i \in \Omega,\ \forall k = 1, \dots, K, \quad (46)$$

$$D_i + t_{ij} + p_j - D_j \le M(1 - X_{ij}^k) \qquad \forall i, j \in \Omega,\ \forall k = 1, \dots, K, \quad (47)$$

$$X_{ij}^k = 0\ or\ 1 \qquad \forall i, j \in \Omega, \forall k = 1, \dots, K, \quad (48)$$

$$Y_k, D_i \ge 0 \qquad \forall i \in \Omega, \forall k = 1, \dots, K, \quad (49)$$

```
/*********************************************
 * OPL 12.2 Model
 * Author: Zabet
 * Creation Date: 2011/9/23 at 02:23:10
 *********************************************/
using CP;

int n = 4; // n tasks
int m = 2; // m machines
int T[1..n][1..n] = ...; // T[i][j] gives duration of task i to task j
tuple triplet {
  int source;
  int destination;
  int distance;
};
{triplet} transitions = ...;

dvar interval task [i in 1..n] size T[i][i];
dvar interval alloc[i in 1..n][j in 1..m] optional;

dvar sequence resource[j in 1..m]
   in    all(i in 1..n) alloc[i][j] // Allocated task
   types all(i in 1..n) i;          // Task type (position)

minimize makespan;
subject to {
  forall (i in 1..n)
    alternative(task[i], all(j in 1..m) alloc[i][j]); // Resource allocation
  forall (j in 1..m)
    noOverlap(resource[j], transitions, 1); // Unary resource with transition time M
}
```

Fig. 11 – OPL model of 2nd scenario

Result:

In the following table, we can obviously see, the effects of using alternative leads to some intervals are not presented in the result. The 2nd and 4th tasks are allocated to the first machine and the 1st and 3rd tasks are allocated to the second machine.

Index 0 (size 4)	Index 1 (size 2)	Interval			
		Present	Start	End	Size
1	1	0	0	0	0
1	2	1	0	1	1
2	1	1	49	51	2
2	2	0	0	0	0
3	1	0	0	0	0
3	2	1	31	34	3
4	1	1	0	4	4
4	2	0	0	0	0

Fig. 12 - The 2nd scenario result, allocation of tasks to machines plus start time, stop time, and duration of each task

Also, in the defined sequence we see the intervals and transition time between them for each machine. In this scenario, completion time is reduced to 51 because of two cranes handle the tasks depicted in Fig. 12 and Fig. 13.

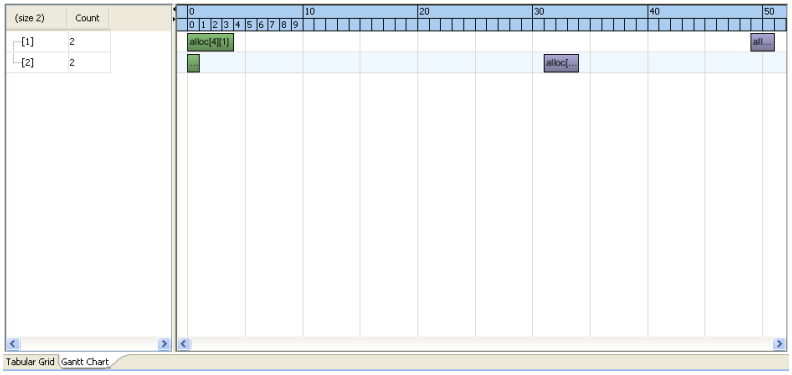


Fig. 13 - Gantt chart of the 2nd scenario result

c) 3rd scenario

In the third simulation, as far as we know our transition matrix between task *i* and task *j* are non-symmetric in our real problem, because some tasks must be assigned ~~may define~~ to some machine and some others must be assigned ~~may define~~ to other part of machine (i.e. some tasks aren't defined never to some machine). Regarding to this fact, our model should be more complex. Also from now, our problem becomes a customized optimization problem which is more problematic and needs to define a special model. In this case, we should change the entire n-dimensional array (except of transition time matrix) to tuple (tuple is a data structures in OPL which can be constructed cluster together closely related data). Also, we should modify the defined constraints over the above new changes.

In this case, we assume 7 tasks for 3 machines and the transition time matrix T:

```
T = [
      [10, 50, 30, 40, 50, 30, 40],
      [90, 20, 85, 80, 90, 80, 60],
      [80, 90, 30, 70, 80, 90, 70],
      [50, 45, 30, 40, 50, 45, 30],
      [80, 65, 30, 50, 50, 85, 70],
      [60, 85, 50, 45, 90, 60, 60],
      [50, 45, 90, 70, 80, 75, 70]]
```

(50)

And, we assume our problem is same as Fig. 6 with overlapping and non-overlapping tasks, so we have 3 cranes, task no. 1,2,3,4 are allocated for machine 1, task no. 3,4,5 are allocated to machine 2, and task no. 5,6,7 are for machine 3. According to our modification (16)-(30), we can define a;; tasks to QCs as: $\{1,2,3,4\} \in \Delta_1$, $\{3,4,5\} \in \Delta_2$, $\{5,6,7\} \in \Delta_3$, for non-overlapping tasks: $\{1,2\} \in \bar{\Delta}_1$, $\{6,7\} \in \bar{\Delta}_3$. Also, overlapping tasks are defined: $\{3,4\} \in \bar{\bar{\Delta}}_{1,2}$, $\{5\} \in \bar{\bar{\Delta}}_{2,3}$.

The mathematical formulation according to our modification (16)-(30) will be:

Commented [Z38]:
RN: how?

IZ: By using "alternative" function some intervals are not presented in results (gray with number 0)

Commented [Z39]:
RN: Be assigned?

added

Commented [Z40]:
RN: Be assigned?

added

Commented [Z41]:
RN: In which sense?

IZ: In the sense that we should remove some tasks from some machines, i.e. all of QCs cannot use entire of transition matrix T for defining their scheduling path any more.

Commented [Z42]:
RN: Why?

IZ: I meant that, we add some constraints to our prev. model.

Objective

$$minmize\ \alpha_1 W + \alpha_2 \sum_{k=1}^{K} Y_k \quad (51)$$

Cosntraints:

$$Y_k \le W \qquad \forall k = 1, \dots, K, \quad (52)$$

$$\sum_{k} \sum_{i \in \Omega} X_{ij}^{k} = 1 \qquad \forall j \in \Omega, \quad (53)$$

$$\sum_{j \in \Delta_k} X_{ij}^{k} - \sum_{j \in \Delta_k} X_{ji}^{k} = 0 \qquad \forall i \in \Delta_n, \quad n = 1, \dots, K, \quad \forall k = 1, \dots, K, \quad (54)$$

$$D_i + t_{ij} + p_j - D_j \le M(1 - X_{ij}^{k}) \qquad \forall i, j \in \Delta_k, \quad \forall k = 1, \dots, K, \quad (55)$$

$$D_i + p_j \le D_j \qquad \forall (i, j) \in \Phi, \quad (56)$$

$$X_{ij}^{k} = 0\ or\ 1 \qquad \forall i, j \in \Omega, \forall k = 1, \dots, K, \quad (57)$$

$$Y_k, D_i \ge 0 \qquad \forall i \in \Omega, \forall k = 1, \dots, K, \quad (58)$$

Results:

In the following table, we can see the solution for 7 tasks with 3 machines with the constraints which is mentioned above. The present intervals are depicted in Fig. 16 and Fig. 17.

With some modification to our OPL model, the tuple was included same as follows:

```
tuple Modes{  //Task allocation to machines
      int modeid; //arbitrary task number for each machine
      int machine;
      int taskid;
  };
```

Fig. 14 – Using tuple in 3rd scenario to define which tasks must be performed by which machines.

In Fig. 14, the tuple (Modes) contains three elements. "modeid" refers to a sequential arbitrary number for tasks of each crane, "machine" indicates the machine number, and "taskid" refers to id number (ship bay number) of the task. I try to depict the example code in Fig. 15.

```
{Modes} modes = {
      <1,1,1>,<2,1,2>,<3,1,3>,<4,1,4>,
      <1,2,3>,<2,2,4>,<3,2,5>,
      <1,3,5>,<2,3,6>,<3,3,7>
  };
```

Fig. 15 – the instance "mode" from tuple "Modes" for the 3rd scenario

In Fig. 16, with

modes (size 10) | Interval

modeid	machine	taskid	Present	Start	End	Size
1	1	1	1	190	200	10
2	1	2	1	0	20	20
3	1	3	0	0	0	0
4	1	4	1	100	140	40
1	2	3	1	130	160	30
2	2	4	0	0	0	0
3	2	5	1	0	50	50
1	3	5	0	0	0	0
2	3	6	1	0	60	60
3	3	7	1	145	215	70

Tabular Grid | Gantt Chart

Fig. 16 - The 3rd scenario result, allocation of tasks to machines plus start time, stop time, and duration of each task

Commented [Z43]:
RN: Provide more explanation on Fig?

IZ: added

Commented [Z44]:
Updated

Also, the sequence of tasks are depicted as the following Gantt chart in Fig. 17. We obviously see that tasks no. 1,2 and 4 are allocated for machine 1. Tasks no. 3 and 5 are allocated to machine 2. Tasks no. 6 and 7 are for machine 3.

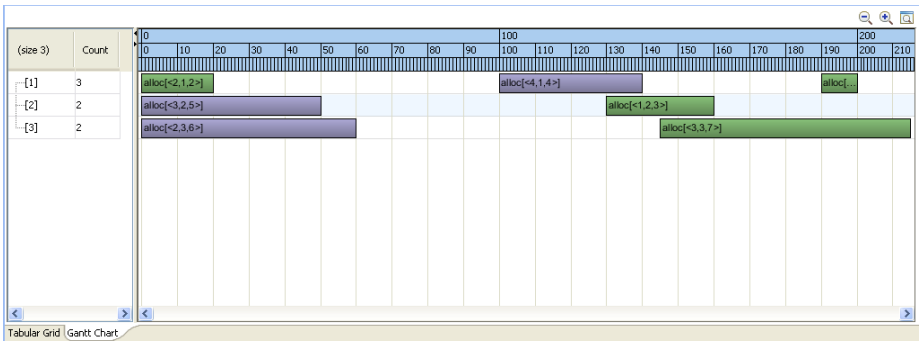


Fig. 17 - Gantt chart of the 3rd scenario result with makespan 215

d) 4th scenario

In the next simulation, we include "initial transition time" for each crane according to our prior knowledge about initial state of each crane before doing tasks. This is more important when a crane wants to perform the first previous simulation task when initial transition time to perform it became high. In this case, crane should perform another strategy to do tasks with a new order. In the re-defined model we use "append" function which append the initial intervals to the previous intervals and then solve the problem.

The mathematical formulation according to our modification (16)-(30) will be:

Objective

$$minmize\ \alpha_1 W + \alpha_2 \sum_{k=1}^{K} Y_k \qquad (59)$$

Cosntraints:

$$Y_k \le W \qquad \forall k = 1, \dots, K, \qquad (60)$$

$$\sum_{j \in \Delta_k} X_{0j}^k = 1 \qquad \forall k = 1, \dots, K, \qquad (61)$$

$$\sum_{i \in \Delta_k} X_{iT}^k = 1 \qquad \forall k = 1, \dots, K, \qquad (62)$$

$$\sum_{k} \sum_{i \in \Omega} X_{ij}^k = 1 \qquad \forall j \in \Omega, \qquad (63)$$

$$\sum_{j \in \Delta_k} X_{ij}^k - \sum_{j \in \Delta_k} X_{ji}^k = 0 \qquad \forall i \in \Delta_n, \quad n = 1, \dots, K, \quad \forall k = 1, \dots, K, \qquad (64)$$

$$D_i + t_{ij} + p_j - D_j \le M(1 - X_{ij}^k) \qquad \forall i, j \in \Delta_k, \ \forall k = 1, \dots, K, \qquad (65)$$

$$D_i + p_j \le D_j \qquad \forall (i, j) \in \Phi, \qquad (66)$$

$$D_j + t_{jT}^k - Y_k \le M\left(1 - X_{jT}^k\right) \qquad \forall j \in \Delta_k, \forall k = 1, \dots, K, \qquad (67)$$

$$r_k - D_j + t_{0j}^k + p_j \le M\left(1 - X_{0j}^k\right) \qquad \forall j \in \Delta_k, \forall k = 1, \dots, K, \qquad (68)$$

$$X_{ij}^k = 0 \text{ or } 1 \qquad \forall i, j \in \Omega, \forall k = 1, \dots, K, \qquad (69)$$

$$Y_k, D_i \ge 0 \qquad \forall i \in \Omega, \forall k = 1, \dots, K, \qquad (70)$$

Commented [Z45]: Updated

Commented [Z46]: RN: What are the objective function values? Is one solution better than another?

How do the different scenarios that you consider ??? to each other? -> make a table

IZ: Added in Table 1

Commented [Z47]: RN: What do you mean with that?

IZ: initial / final transition time regarding to the time need to move a crane from its initial state to the first task / from the last task to the final state of crane

Commented [Z48]: RN: ?

IZ: It is a function to concatenate intervals to an existing OPL model.

e) 5th Scenario

In this scenario by defining new sequences and constraints related to precedence relationship and consequently non-crossing constraints between adjacent quay cranes, we reach to the final model same as the Modified QCSP (16)e(30) model with all the constraints in the original QCSP (1)-(15) which is proposed by Kim and Park in [6], including the overlapping and non-overlapping definition of tasks. The latest model is depicted in Fig. 18. Our simulation for the centralized (single agent) QCSP will be based on this model and then compare with the proposed D-QCSP.

```
using CP;

int n = 100;       // n tasks
int m = 4;         // m machines
int T[1..n][1..n] = ...; // T[i][i] gives duration of task i

tuple Tasks{
  key int taskid;
};
{Tasks} tasks = ...;

tuple Modes{       //Task allocation to machines
      int modeid; //arbitrary task number for each machine
      int machine;
      int taskid;
};
{Modes} modes = ...;

tuple Triplet {
  int source;
  int destination;
  int distance;
};

{Triplet} transitions = {<i, j, T[i][j]> | i in 0..n+1, j in 0..n+1};

/*********************************************************************/
// intervals of all tasks with its duration, (no start, no end time) (all must
present in solution)
dvar interval tasks_itvs      [t in tasks]      size T[t.taskid][t.taskid];

// intervals of all modes (optional: some of them will be present in solution)
// (no start time, no end time, no duration)
dvar interval alloc_itvs      [i in modes]      optional;

dvar sequence resource_seq [j in 1..m] in
                  all    (i in modes : i.machine==j) alloc_itvs[i] // Allocated task
                  types all(t in modes : t.machine==j) t.modeid;  // Task type

dvar sequence crossover_seq_L [m1 in modes] //[t in tasks][k in 1..m]
                  in    append (
                        all(m2 in modes :
                                    (m2.machine ==    m1.machine-1)
                              &&    (m2.taskid  >     m1.taskid)
                                          ) alloc_itvs[m2],

                              alloc_itvs[m1]);  // Allocated task

dvar sequence crossover_seq_R [m1 in modes] //[t in tasks][k in 1..m]
                  in    append (
                        all(m2 in modes :
                                    (m2.machine ==    m1.machine+1)
                              &&    (m2.taskid  <     m1.taskid)
                                          ) alloc_itvs[m2],
```

```
                              alloc_itvs[m1]);  // Allocated task

constraint ctPrec1[modes][modes];
constraint ctPrec2[modes][modes];
constraint ctAlt[tasks];
constraint ctOvlp[1..m];

//dexpr int makespan = sum (j in 1..m) max(i in modes : i.machine==j)
endOf(task[i]);
dexpr int makespan = max(t in tasks) endOf(tasks_itvs[t]);
/********************************************************************/
minimize makespan;
subject to {

      forall (t in tasks){
            ctAlt[t]    :
            // Resource allocation
            // A "Vertical" constraint
            // An alternative constraint between an interval decision variable "a"
            //    and a set of interval decision variables B states that interval a
            //    is executed if and only if exactly "one" of the
            //    members of B is executed.
            // In that case, the two tasks are "synchronized"
            //    (= have the same start,stop, and duration).
            alternative(tasks_itvs[t],
                        all(i in modes : i.taskid == t.taskid ) alloc_itvs[i]);
            }
      forall (j in 1..m)
            ctOvlp[j]   :
            // noOverlap (<sequenceName> [,M]);
            // Unary resource with transition time M
            //    (M is an optional transition matrix (in the form of a tuple set)
            //      that can be used to maintain a minimal distance between the
            //    end of one interval and the start of  the next interval in the
sequence.)
            // The resource sequence connect to transition time triplet using its
"Type"
            noOverlap(resource_seq [j], transitions, 1);


      forall (md in modes){
            forall (nd in modes     :           (nd.machine==md.machine+1)
                                          &&    (nd.taskid<md.taskid))
                  ctPrec1[md][nd]:
                  noOverlap( append(      alloc_itvs[md],   alloc_itvs[nd]));

            forall (ne in modes     :     (ne.machine==md.machine-1)
                                          &&    (ne.taskid>md.taskid))
                  ctPrec2[md][ne]:
                  noOverlap( append(      alloc_itvs[md],   alloc_itvs[ne]));
      }

      forall (t in tasks)
            forall (m in modes){
                  noOverlap (crossover_seq_L[m]);
                  noOverlap (crossover_seq_R[m]);
            }
  }
```

Fig. 18 – The latest model with all the constraints same as Modified QCSP (16)-(30)

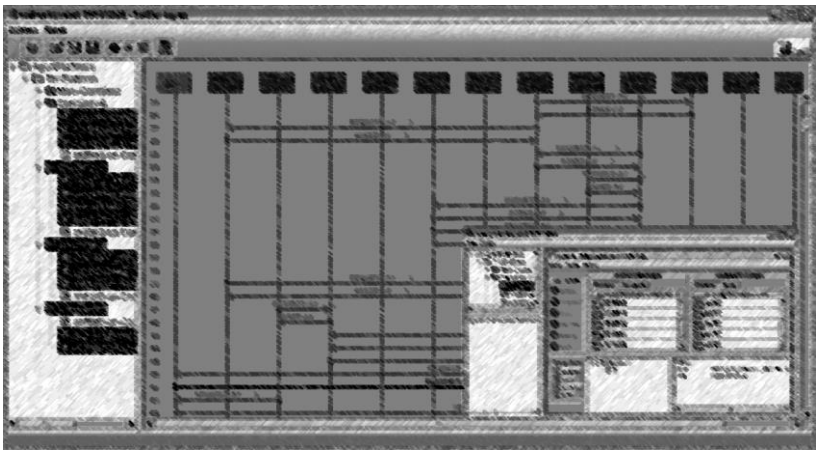
Until now, we conclude the previous models and scenario within Table 1.

Constraints / Scenarios	Non-preemption	Overlapping and Non-overlapping tasks	Initial and final states	Precedence relationship	Non-Crossing over (Non-Interference)	Procedure
QCSP (1)-(15)	𝕐	ℕ	𝕐	𝕐	𝕐	MILP
Modified QCSP (16)-(30)	𝕐	𝕐	𝕐	𝕐	𝕐	
1st	𝕐	ℕ	ℕ	ℕ	ℕ	OPL Model
2nd	𝕐	ℕ	ℕ	ℕ	ℕ	
3rd	𝕐	𝕐	ℕ	ℕ	ℕ	
4th	𝕐	𝕐	𝕐	ℕ	ℕ	
5th	𝕐	𝕐	𝕐	𝕐	𝕐	
D-QCSP	𝕐	𝕐	𝕐	ℕ	𝕐	Distributed (Multi-Agent)

Table 1 – Comparisons of the properties of the four scenarios, QCSP and our modified QCSP formulation

~~*Last scenario*~~

~~In the final simulation, I intend to implement the problem as a distributed one with respect to each crane assume as an agent. They can solve their own problem and find an optimized and the shortest sequence for performing their tasks. Also, in order to have overlapped tasks with their adjacent crane, they may execute communication to them to make optimized decisions. This notion could be implemented as an agent platform called Java Agent Development Environment (JADE), please see Fig. 15.~~



~~Fig. 15 – An overview of Java Agent Development Environment (JADE) and communication process among agents~~

Commented [Z49]:
RN: Should this be in the next section?

IZ: Yes this is my previous report and this section for information.
I remove it.

Commented [Z50]:
RN: What does a local solution provide?

IZ: As I explained into one of previous comment, In distributed scheme we will only use the 1st scenario for each agent after allocating tasks in each iteration.

IV. Extended Asynchronous BackTracking

A. Introduction

Now, let us assume that each quay crane has a certain agent called QCA, and each QCA has non-overlapped and overlapped tasks (Fig. 5). In order to coordinate among agents to reach the solution of the quay crane scheduling problem, we need to propose an algorithm which enables agents to perform their tasks in a distributed cooperation manner. We will first introduce algorithms for solving scheduling problems in a distributed way. In this section, an asynchronous search algorithm is presented to aid reaching a solution of the QCSP by communicating among agents.

The problems that have been addressed by distributed search algorithms can be categorized into three main classes: path-finding problems, constraint satisfaction problems, and two-player games. In the Constraint Satisfaction Problem (CSP), we usually involve to find a goal that satisfies the given conditions (constraints) which this class of problem originally developed

for single-agent problem solving. We can assume QCSP as a CSP and try to search for solution with a distributed structure like the proposed MAS.

Distributed search algorithms for MAS are distinguished to work either synchronously or asynchronously [25]. In a *synchronous* algorithm, each agent waits for the messages from its peer agents, and only after having received them, it starts solving and computing, and then sending out messages even before having received any confirmation message from its peers. In this case, as incoming messages arrive, they assimilate them into their computation, and if needed, they will send out updated messages of their own. In contrast to that, *asynchronous* algorithms have the potential advantage that an agent will not be waiting for messages from its peers. On the other hand, this algorithm model ensures agents to perform their computations based on their *most up to date* information. In asynchronous algorithms, execution order can be highly flexible and arbitrary. Furthermore, computational and recognition abilities to perform local performances for each agent can be small enough, and each step of the algorithm not require the global knowledge of the problem.

B. Distributed Constraint Optimization

Distributed Constraint Optimization Problems (DCOP or DisCSP) have been considered as an important research area for multi-agent systems. DCOP can model many distributed real-world systems. The challenge is to find a way to solve them with fully distributed algorithms efficiently.

In the literature, a number of formulations have arisen including the distributed partial valued constraint satisfaction problem (DPCSP) [26], distributed valued constraint satisfaction problem [27], and the distributed constraint optimization problem (DCOP) [28]. These problems can be solved either to optimally or near optimally by a number of powerful algorithms which have been introduced. For instance, Distributed Depth-first Branch and Bound (DDBB) and Distributed Iterative Deepening (DID) [28], Synchronous Branch and Bound (SBB) and Iterative Distributed Breakout (IDB) [26], and the Asynchronous Distributed Optimization (Adopt) algorithm [29].

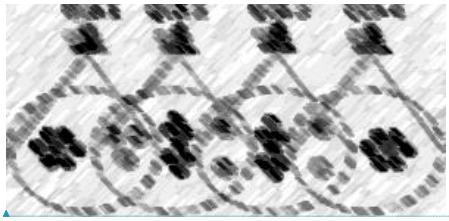
A DCOP has been formalized in [30] as a $tuple\ \langle \mathcal{A}, \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$, where $\mathcal{A} = \{a_1, \dots, a_i\}$ is a set of i agents, $\mathcal{X} = \{x_1, \dots, x_n\}$ is a set of variables that each agent a_j is assigned one or more variables along with the functions on their variables. $\mathcal{D} = \{D_1, \dots, D_n\}$, is a set of values domains whose are taken by variables, where D_i is a finite set of values to which variable x_i may be assigned. $\mathcal{C} = \{c_1, \dots, c_m\}$ is a set of binary/integer constraints that specify the combinations of values allowed for the ~~two~~ multiple variables ~~tasks~~ they involve. For example, a constraint $c_{ij} \in \mathcal{C}$ between two variables x_i and x_j is a subset of the Cartesian product $c_{ij} \subseteq D_i \times D_j$. The problem is to find an assignment $A^* = \{d_1, \dots, d_n | d_i \in D_i\}$, such that the global cost $\mathbb{C}$, is minimized. Note that, d_i refers to the values from domain D_i of variable x_i. In this paper, for our formulated problem, $\mathbb{C}$ for a set of allocation A can be defined as follows:

$$\mathbb{C}(A) = \sum_{i=1}^{m} c_i(A) \tag{71}$$

Since DCOP is NP-complete in general, a trial-and-error exploration of alternatives is inevitable, also it is solved iff the following conditions are satisfied:

$\forall i, \forall x_j$ where belongs (x_j, i), the value of x_j is assigned to d_j, and $\forall l, \forall p_k$ where known (p_k, l), p_k is true under the assignment $x_j = d_j$

DCOP is distributed constraint optimization which a group of agents must choose their values (tasks) for a set of variables (quay cranes) distributedly such that the cost for a set of constraints over the variables should be optimized (minimized or maximized depends on constraints definition). DCOP is a framework for describing a problem in terms of constraints that are known by distinct agents. The constraints of the distributed problem are described on some variables with predefined domains which have to be assigned to the same values by different agents. In this case, the multi-agent system should execute search action over the possible values which can be taken.



~~Fig. 4 A schematic shown each QCA with their non-overlapping (black dots) and overlapping (circles) tasks.~~

~~Fig. 4 shows an example of overlapping and non-overlapping area between QCAs. In this example, all of the overlapping tasks have sequential numbers. Each QCA tries to assign the overlapping tasks until no task is remaining. In this way, QCAs~~

Commented [Z51]:
RN: Rephrase?

IZ: Let me please describe it more, since, I had a trouble to digest this formulation before, even we can obviously see that this formulation is described not correctly in Wikipedia.
Also, there are several definitions were proposed.
The simple version of DisCSP is CSP. In the CSP we have just $tuple\ \langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$ without $\mathcal{A}$.
Now in DisCSP/DCOP we have $tuple\ \langle \mathcal{A}, \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$ [in some literature 5-tuple was proposed: $tuple\ \langle \mathcal{A}, \mathcal{X}, \mathcal{D}, \mathcal{C}, \varphi \rangle$, where $\varphi: \mathcal{X} \to A$ is a function assigning variables from variables $\mathcal{X}$ to agents $\mathcal{A}$ from A]. This means that the problem (variables, domain, and values) is the same. But in DCOP, we have agents which can allocate one or more variables. Variables usually refers to a real entity. E.g. in our problem, each variable is one quay cranes. Now we define each quay crane (variable) has one corresponding agent. We can also define for e.g. two or more quay cranes (values) have one agent. So, in our problem each agent has just one variable (quay crane) so their numbers are the same.

Commented [Z52]:
RN: Tasks? On variables?

IZ: variables = quay cranes
Values = tasks/container holds
Each variable has its own domain which can has overlapping with another variable's domain or not.
Each domain consists of one or more values that may assigned by another variable's domain.

Formatted: Condensed by 0.05 pt

Formatted: Font: (Default) Times New Roman, 10 pt, Not Italic, Complex Script Font: Times New Roman, 10 pt, Not Italic, Not Small caps

will adopt ABT algorithm and start the algorithm from the highest agent i.e. QCA1. We can assume upper-bound M and lower-bound m for doing all overlapping tasks:

These indexes can restrict the algorithm and can omit many search branches in order to reach to a near optimal solution within a reasonable time and efforts. Since we know number of whole overlapping tasks $\bar{n} = n - \tilde{n}$, we can derive the minimum and maximum mean time for each crane. Each crane completion time must be within minimum and maximum mean time $T_{min} < T_i < T_{max}$ where $i = 1,2,3, or\ 4$. This fact can be important when all QCAs try to assign near tasks to their selves. Where, i is referred to the number of QCAs. In our example $i = 4$. In this case, we can guarantee that each QCA designs limited number of tasks which don't exceed their time from a reasonable time, also tasks are assigned to cranes near equally.

C. ABT Algorithm

1) Filtering Algorithm

Consider an agent that knows its constraints, its domain of variables, and the domain of variables of its neighbor agents. He can use this information to determine whether any values of its variable's domain are possible or impossible. That is, if there is a value which violates one or more constraints, regardless of what value other neighbors use, then that value will obviously not be part of the solution and will be omitted. All other agents do the same. This idea is implemented by filtering algorithm [31] shown in

Fig. 19. Every agent i executes the FILTERING procedure which leads it to do a REVISE with every one of its neighbors variables x_j. If the agent does succeed in eliminating one or more values from its domain by filtering them, it then tells its neighbor its new domain. On the other side, when an agent receives new domains from its neighbors it again executes REVISE for those neighbors. This procedure repeats until all agents finish their allocations and computations and finally no more messages are sent.

• FILTERING ( )

1. **for** $j \in$ {neighbors of i} // i is this agent.
2. **do** REVISE (x_i, x_j)

• HANDLE_NEW_DOMAIN (j, D')

1. $D_j \leftarrow D'$
2. REVISE (x_i, x_j)

• REVISE (x_i, x_j)

1. $old\_domain \leftarrow D_i$
2. **for** $v_i \in D_i$
3. **do if** there is no $v_i \in D_j$ consistent with v_i
4. **then** $D_i \leftarrow D - v_i$
5. **if** $old\_domain \neq D_i$
6. **then** $\forall_{k \in \{\text{neighbors of } i\}}\ k.$HANDLE_NEW_DOMAIN (i, D_i)

Fig. 19 – Filtering algorithm

The filtering algorithm is guaranteed to stop if there are only a finite number of values in all the agent domains and the algorithm only sends a message when a value is eliminated from a domain. Furthermore, this algorithm is not complete and might stop at a point where one or more of the agents still have many values in its domain. Thus, it can stop before finding a solution. Also, filtering algorithm cannot detect problems that do not have a solution reliably. Because the REVISE function considers only the agent's own value and one other agent's value at a time, it might fail/get stuck at first step of some problem (e.g. graph coloring problem with three agents and two color) [32].

2) Hyper-Resolution Based Consistency Algorithm

To overcome the deficiencies of filtering algorithm, we need to consider the agent's value against not only the values of one of its neighbors, but also the values of all possible combination of the values of all other neighbor agents at the same time. More formally, we say that if an agent can find a suitable value regardless of the values of k of its neighbors then it can be consistent wth anything they might do. If all the agents within a problem meet this, we have achieved *k-consistency*. And if the above holds for *all* $j \leq k$, then we say the problem is *strongly* *k-consistency*. Note that, filtering algorithm guaranties 2-

consistency. Now, we need to extend filtering procedure to more than one neighbor. This can be done by using *hyper-resolution rule*.

Definition 1 - Hyper-Resolution Rule

$$\frac{\begin{array}{l} A_1 \vee A_2 \vee \ldots \vee A_m \\ \neg(A_1 \wedge A_{11} \wedge \ldots) \\ \neg(A_2 \wedge A_{21} \wedge \ldots) \\ \vdots \\ \neg(A_m \wedge A_{m1} \wedge \ldots) \end{array}}{\neg(A_{11} \wedge \ldots \wedge A_{21} \wedge \ldots \wedge A_{m1} \wedge \ldots)} \quad (72)$$

The hyper-resolution idea, is that the agents domains can be represented as an "or" statement like the first one in the rule. The constraints are represented as negations like the one in the second line of hyperrresolution rule (see Definition 1) For example, in a simple graph coloring problem; if agent x_1 can be either gray, black, or white then we write $x_1 = \text{gray} \vee x_1 = \text{black} \vee x_1 = \text{white}$. The constraints for example, if x_1 and x_2 cannote both be white we would write that as $\neg(x_1 = \text{white} \wedge x_2 = \text{white})$. Once we have represented both the domain and the constraints in this logical form then we can use the hyper-resolution rule to generate new constraints which are added to the agent's knowledge and communicated to the other agents. Each one of these new constraints is called a "*nogood*".

We can use the hyper-resolution rule to implement a distributed constraint satisfaction algorithm that, unlike the filtering algorithm, will detect all cases where there is no solution and will find a solution when there is one. The algorithm is basically the same as the filtering algorithm but we modify the REVISE function to instead use the hyper-resolution to generate new nogoods. These nogoods are then sent to the other agents. When an agent receives nogoods from other agents it adds them to its set of nogoods. The process continues until no new nogoods can be derived by any agent or until a contradiction is reached. When a contradiction is found it means that no solution exists to this problem.

In practice, the hyper-resolution algorithm is very slow because there is often little guidance as to which nogoods should be derived first. That is, at any one time each agent can derive a large number of different nogoods. Most of these are useless to the other agents and there is no easy way for the agent to determine which nogoods might be most useful since that would require that it also know about the other agents' databases. As such, we find that in practice the hyper-resolution algorithm spends a lot of time deriving useless nogoods. This is especially true as the size of the nogoods - their number of terms - increases. Also, if the problem does have a solution then the algorithm only stops when all the agents prove to themselves that they cannot generate any new nogood, which can take a very long time. In summary, hyper-resolution in its raw form is impractical for most cases with the possible exception of highly over-constrained problems [32].

Commented [Z53]:
IZ: I have added this section for more clarification on filtering and hyper resolution from

~~1)~~3) *Asynchronous Backtracking algorithm*

The *Asynchronous BackTracking (ABT)* algorithm is a well-known algorithm for solving DCOPs as a distributed version of the formal backtracking algorithm [17]. In the backtracking algorithm, there are two possible steps, an assignment step and a backtrack step. In the first step, one agent starts to assign a task from the overlapping area, and check with other agents that no constraint is violated due to new assignment. If the first step was true, another task will be assigned and continue, or the algorithm will be terminated if all tasks are assigned. If a constraint is violated under the first step, backtracking will be executed as second step. In this step, the last wrong assignment is undone and the next available task from unassigned tasks domain is selected instead if there exists such a value. If there is no next task, another backtrack step is taken. Indeed, the backtrack algorithm is a depth-first search on the search tree of all possible tasks assignments to all agents.

The ABT algorithm can be categorized as depth-first search algorithm, under blind/exhaustive/brute-force searches algorithm. The principle idea of backtracking is to construct a tree of choices/solutions - called the state-space - on component at a time and evaluate such partially constructed candidates. The root of such tree represents the initial state before the search for a solution begins. During search, algorithm selects the first node at the first level and then gets through the first node in the second level until construct a branch of the tree. The nodes at each nth level represent the choices for the nth component. If a partially constructed solution can be developed further without violating the problem's constraints, it is done by taking the first remaining legitimate option for the next component. If there is no legitimate option for the next component, no alternatives for any remaining component need to be considered. In this case the algorithm backtracks (step backward) to its parent node to replace the last component of the partially constructed solution with its next option.

Asynchronous backtracking algorithm is used to solve problems for which a sequence of objects is to be selected from a set such that the sequence satisfies some constraints. In our problem, a set of overlapping tasks should be selected by each quay crane regarding to their costs and constraints. Moreover, in our problem the branches of the search tree are limited and the branch could be deep, so the backtracking algorithm could be the best solution among other exhaustive search algorithms.

Commented [Z54]:
RN: Why is backtracking needed?

IZ: added

The backtrack procedure is responsible for generating and sending new nogoods. As we saw, it is only called when there is no way for the agent to set is value so that no constraints are violated. This means that someone else must change their

value. That someone else is the agent with the lowest priority in the nogood. Backtrack first uses the hyper-resolution rule (or similar) to generate one or more nogoods. Each one of these nogoods is then sent to the agent in that nogood that has the lowest priority. Also, if the hyper-resolution generated a contradiction, represented as an empty set then this means that no solution exists and thus the algorithm is terminated.

Notice that, since the nogoods are always sent to the lowest priority agent and the agents have linear priorities, the nogoods always flow from the high priority agents to the low priority agents. Namely, the highest priority agent will never receive a nogood, the second highest priority agent will only receive nogoods from the highest priority agent, and so on. This suggests to us that the lower priority agents will end up doing more work [32].

Commented [Z55]:
IZ: I have added this section for more clarification on filtering and hyper resolution from

In a DCOP, agents can eliminate a number of hypotheses by using filtering algorithm or the hyper-resolution-based consistency algorithm [33]. Both of the algorithms are examples of a general class of algorithms called *consistency algorithms*. In hyper-resolution-based rule, all constraints are represented as a "nogood", which is prohibited combination of variable values. The "nogood" is generated by an agent to inform other agents about a taken variable from its domain and consequently eliminates non-possible solutions. In ABT algorithm, which is using hyper-resolution rule to find solution, the generation procedure of a new "nogood" message is basically identical to the hyper-resolution rule. In other words, a new "nogood" is generated only when the constraints that are not satisfied in the current situation. During the search, if consistency is not achieved, then agents using hyper-resolution rule to generate new message, which express no more available task exists to assign, and exchanging them to higher priority agent. Therefore, as long as, a single selection of values would become fatal in large-scale problems, such an exhaustive search can be virtually impossible for large number of agents.

This model assumes the existence of a reliable underlying communication structure among the Quay Crane Agents (QCAs), so the topology of the physical communication network plays an important role here. We assume the agents have other adjacent agents addresses and can send messages to them. Also the delay in delivering a message is finite.

The basic idea of the backtracking algorithm is to construct a partial solution first, which an agent is assigned to a subset of tasks that satisfies all the original constraints within the subset, and then to expand the partial solution to complete solution by adding new tasks, assigning new tasks will iterate until the partial solutions become complete. In our QCSP, there are initial non-overlapping tasks between cranes, and they are assigned to each QCA firstly. Then, the iteration to allocate remaining overlapping tasks is being started until all the tasks are allocated to all QCAs.

In ABT, agents are acting asynchronously and concurrently based on their local knowledge of their environment, they communicate with each other until they reach a complete solution. For example, each QCA has its own knowledge about number of the overlapping and non-overlapping tasks, number of its adjacent QCAs and so on. On the other hand, In order to avoid task assigning violations, the agents priority order is determined based on alphabetic order of their names and each agent tries to find a task in the sence of ABT algorithm which satisfies the constraints with the variables of higher priority agents. In other word, in QCSP, when a QCA assigns tasks for itself, the agent is strongly committed to the selected task, i.e., the selected task will never be changed unless an exhaustive search is being performed by a lower priority QCAs.

• **when received** (**ok?**, (x_j, d_j)) **do**

1. add (x_j, d_j) to *Agent_View*;
2. **check_agent_view** ; **end_ do**;

• **when received** (**Nogood** , x_j, nogood) **do**

1. add Nogood to Nogood list;
2. **when** *Nogood* contains an agent x_k that is not its neighbor **do**
3. request x_k to add x_i as a neighbor,
4. and add (x_k, d_k) to *Agent_ View* ; **end_ do**;
5. *old_ value* ← *current_value*; **check_ agent_view**;
6. **when** *old_value* = *current_value* **do**
7. send (**ok?,** (x_j, *current_value*)) to x_j; **end_ do**; **end_ do**;

procedure **check_agent_view**

1. **when** *Agent_View* and *current_value* are not consistent **do**
2. **if** no value in D_i is consistent with *Agent_View*
3. **then backtrack** ;
4. **else** select $d \in D_i$ where *Agent_View* and d are consistent;
5. *current _value* ← d;
6. send (**ok?**, (x_i, d) to *low_priority_neighbor*s ; **end_ if** ; **end_ do**;

procedure **backtrack**

1. *nogood* ← *inconsistent_subset*; //using hyper-resolution rule for checking inconsistency
2. **when** *nogood* is an empty set **do**
3. broadcast to other agents that there is no solution;
4. terminate this algorithm; **end_ do**;
5. select (x_j, d_j) where x_i has the lowest priority in Nogood;
6. send (**Nogood** , x_i, *nogood*) to x_j ;
7. remove (x_j, d_j) from Ag ent_ V iew ; **end_ do**;
8. *check_agent_view*

• **when received** (**ok?**, (x_j, d_j)) **do**

1. add (x_j, d_j) to *Agent_View*;
2. **check_agent_view** ; **end_ do**;

• **when received** (**Nogood** , x_j, nogood) **do**

1. add Nogood to Nogood list;
2. **when** *Nogood* contains an agent x_k that is not its neighbor **do**
3. request x_k to add x_i as a neighbor,
4. and add (x_k, d_k) to *Agent_ View* ; **end_ do**;
5. *old_ value* ← *current_value*; **check_ agent_view**;
6. **when** *old_value* = *current_value* **do**
7. send (**ok?,** (x_j, *current_value*)) to x_j; **end_ do**; **end_ do**;

procedure **check_agent_view**

1. **when** *Agent_View* and *current_value* are not consistent **do**
2. **if** no value in D_i is consistent with *Agent_View*
3. **then backtrack** ;
4. **else** select $d \in D_i$ where *Agent_View* and d are consistent;
5. *current _value* ← d;
6. send (**ok?**, (x_i, d) to *low_priority_neighbor*s ; **end_ if** ; **end_ do**;

procedure **backtrack**

1. *nogood* ← *inconsistent_subset*;

2. **when** *nogood* is an empty set **do**

3. broadcast to other agents that there is no solution;

4. terminate this algorithm; **end_ do**;

5. select (x_j, d_j) where x_i has the lowest priority in Nogood;

6. send (**Nogood** , x_i, *nogood*) to x_j ;

7. remove (x_j, d_j) from Ag ent_ V iew ; **end_ do**;

8. *check_agent_view*

Fig. 20 – ABT algorithm

D. Extended ABT Algorithm

In this paper, we present a cooperative based DCOP protocol, called Extended Asynchronous BackTracking (Extended ABT) that allows the agents to make their local decisions for their overlapping tasks as the distributed problem solving. The formal ABT algorithm was proposed to allocate just one value to each variable according to the problem constraints. The Extended ABT is introduced to allocate several values to each variable with as less communication as possible that can guarantee the whole constraints. This algorithm is an incomplete algorithm, so it can reduce the time of search as efficient as possible by identifying the problem substructures at its first step (initializing step).

If we want to explain about similarities and differences between single-agent (centralized) QCSP and multi-agent (distributed) QCSP; as in (1 the objective of centralized QCSP was described, in D-QCSP we have the same objective introduced in **Error! Reference source not found.**). The objective of D-QCSP clearly minimized the completion time all a nd each of the quay cranes as in (2). In this way, after assigning the non-overlapping tasks by supervisory, in each iterations quay cranes try to allocate overlapping tasks cooperatively. We say cooperatively, which means that, each quay crane agent (QCA) tries to allocate overlapping tasks and check its completion time in each iteration until its neighbor QCAs have no less completion time than a predefined (threshold) time. We will discuss more about it later. Like as constraints (3), (4), and (6 our proposed algorithm ensures all the tasks are sequenced before performing by a crane, also like as constraints (5), Extended ABT algorithm indicates that quay crane scheduling is a non-preemptive (i.e. each task is performed by one and only one quay crane). The proposed algorithm computes completion times of each crane as soon as assigning new task and/or backtracking existing task at each iteration somehow different from constraints (7), (9), (10), (12), and (13). More explanations regarding to calculating completion time is described in the algorithm description section. In addition, the proposed algorithm and the used OPL model guarantee the scheduling will prevent interference. The non-interference constraints for the centralized QCSP is introduced in (11).

The Extended ABT, has some similarities and some differences in comparison with the original ABT algorithm. For example, although the main features of both algorithms i.e. selecting value, backtracking value, and check agent view and consistency are the same, there are differences between them on how those feature can be run. In the original ABT, agents try to find appropriate value according to the problem constraint(s) with the mixture of selecting value with hyper-resolution rule and backtracking. On the other hand, in Extended ABT, thanks to the initializing step, agents try to select values in the first phase, then they try to backtrack values in the second phase if there is any inconsistency, so since, all the values related to maximum two variables (agents), there is no need to run hyper-resolution algorithm and the filtering algorithm is enough for our purpose. In extended ABT, we first prioritize the overlapping tasks for their both neighbor QCAs by supervisor according to its position regarding to its agent neighbors. This can help agents to decide more fast which value is better to be assigned in each step and prevent QCA from selecting irrelevant tasks. So, this algorithm is distinguished as an incomplete algorithm. In this paper, we present a distributed algorithm to solve a DCOP near optimally with incomplete search.

The near optimality can be guaranteed by sequencing the tasks between adjacent agents at the first step and before start the algorithm. Sequencing the tasks can be done by each non-overlapping task position regarding to its neighbor QCAs, also each overlapping task is enumerated regarding to its position and the entire overlapping tasks will be formed as string. This can be done by supervisory layer where all the overlapping tasks have distances from the default positions of cranes who can handle them, all this is done in the initializing step of algorithm which is introduced later. Then each agent try to allocate the tasks from the near point of the constructed string (like rope competition but the difference is, each agent can have two ropes from two direction and also the agents try to pull the rope cooperatively not competitively i.e. they try to pull ropes by looking to the opponent agents tasks and time).

We assume that overlapping tasks (bold squares) in Fig. 5 according to its distance from center area (non-overlapping tasks) has a sequential/arbitrary number called "cost" which is referred to the task ship bay nunber. It means that, the cost of handling tasks near center area is less than handling tasks far away. The string is formed like as Fig. 21. In Fig. 21 – Left, we assume there is a string starts from QCA1, get along through the overlapping tasks between two agents and then finishes at QCA2. In Fig. 21 – Right, we assume that the left figure is opened and straightened. Then we can give to each task a unique sequential number from low to high which is the "cost" of each item from its related agent point of view. Note that, the lowest cost item from QCA1 point of view is the highest cost item from QCA2 point of view and vice versa.

Commented [Z56]:
RN: Can you include a short description of the formal ABT algorithm?

IZ: I mean exactly ABT, I also added more descriptions in prev. section. If additional information is needed tell me plz.

Commented [Z57]:
RN: Not introduced yet?

IZ: Added in IV.B-Distributed Constraint Optimization.

Commented [Z58]:
RN: multiple values to one variable?
OR multiple variables with one value??

IZ: According to my clarifications in IV.B-Distributed Constraint Optimization. This is correct

Commented [Z59]:
RN: What constraints?

IZ: Explained in next paragraph

Commented [Z60]:
IZ: Added

Commented [Z61]:
RN: Makes more explain in the text.

IZ: Added

Commented [Z62]:
RN: Include description of this (otherwise it is not clear for people who are no familiar with the topic)

IZ: I mean exactly ABT, I also added more descriptions in prev. section. If additional information is needed tell me plz.

Commented [Z63]:
RN: Introduce the DCOP shortly.

IZ: The DCOP section is changed and placed before this section

Commented [Z64]:
IZ: Added

There are two different ways to allocate the tasks by their two quay cranes; we can optionally define “hard constraint” or “soft constraint” on these costs as our problem constraints. If we force our problem to assign each task from lower cost to upper cost only sequentially, we can call it hard constraint. In other word, if an agent can assign a task if and only if the previous lower cost task was assigned before, we call it hard constraint. Otherwise, if there is not force on assigning tasks in a sequential manner, we can call it soft constraint.

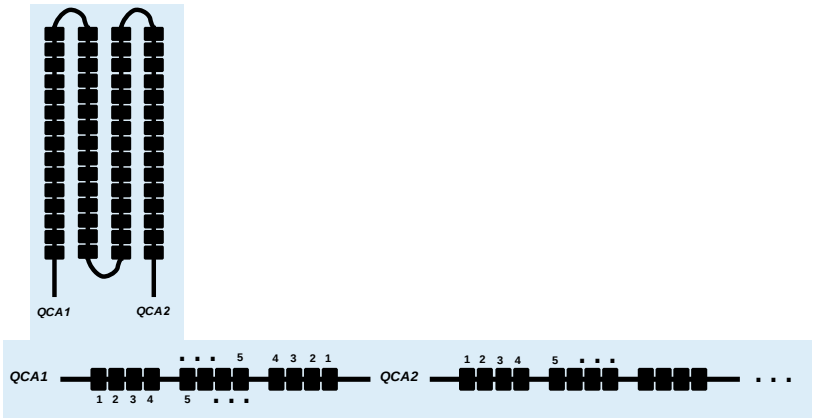


Fig. 21

Fig. 22 shows the above procedure and conversion of the problem. In Fig. 22 – a, there are two agents (QCAs), they have some non-overlapping (black circles) and five overlapping (white circles) tasks. In Fig. 22 – b, we try to depict the previous figure (Fig. 22 – a) and its tasks like as boxes that should put on top of each other. In addition, a string of overlapping tasks is formed and sequenced by some tag numbers. QCA1 try to allocate overlapping tasks from the left point of the string one by one, while, QCA2 try to allocate them vice versa. The completion time of each QCA (t_i) is shown and we can see obviously that t_1 with 9 tasks is greater than t_2 with 7 tasks. In Fig. 22 – c, each agent try to allocate the overlapping tasks and compare each completion time with another agent, until both agents have an equal (near equal) completion time, i.e. $t_1' \approx t_2'$. Alternatively, the difference of their completion time no more than a pre-defined constant (threshold time), i.e. $|t_1' - t_2'| < T_{threshold}$.

Commented [Z65]:
RN: How is the “rope/string” make?
Do you sort the tasks?
How is the ending time computed?

IZ: Added

Commented [Z66]:
IZ: we want to have the same time. But according to the problem, this may cannot be done many times. So we want to reduce differences as small as possible.

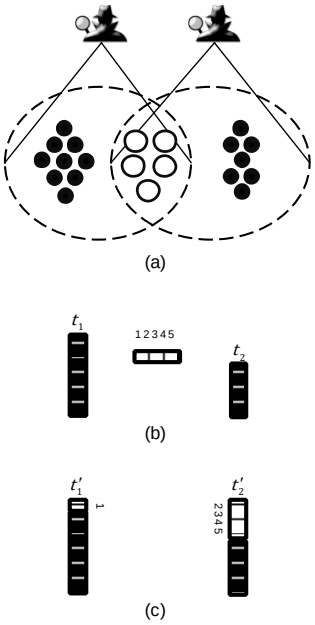


Fig. 22

In the worst case, we can consider:

1. travelling time of a crane from one task to another task
2. precedence relationship between tasks
3. a crane should carry to another position to unload a task as soon as he receive a task (pattern 2 of Fig. 23).

So, we have to model and then optimize/re-schedule the assignments of Fig. 22 – c by a powerful solver such as CPLEX.

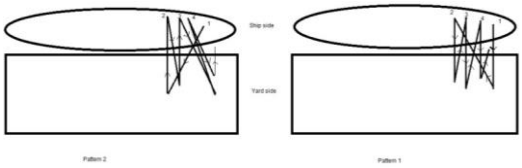


Fig. 23

In this way, the optimized assignment will be like as Fig. 24 where $t'_2 < t''_2$. ~~The agent~~ Agent can decide after allocating the tasks whether overlapping tasks should be run with non-overlapping tasks or not. It can be re-scheduled and categorized the same tasks (the tasks that are near to each other and can be handled with each other subsequently)

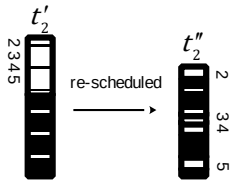


Fig. 24

Notes

Commented [Z67]:
RN: Why worst case?

IZ: I mean if we want to be critical and even consider these 3 items in our problem model

Commented [Z68]:
RN: What are the optimization goals?

IZ: Explained

Commented [Z69]:
RN: In which way?

IZ: Modeling and scheduling

Commented [Z70]:
RN: Which?

IZ: Typical agent. Each agent

Commented [Z71]:
RN: How?

IZ: By the proposed algorithm

Commented [Z72]:
RN: What do you mean?

IZ: After initialization step, by starting agents to allocate non-overlapping tasks, in each iteration, by assigning new non-overlapping task, the agent can re-schedule all the previous and the new task again. In this way, the new task may be handled in the intervening time of previous tasks in the new sequencing form.

Commented [Z73]:
RN: What does "Re-scheduling" mean here?

IZ: This is an example figure. I consider that after allocating non-overlapping tasks by supervisory and some overlapping tasks by its own agent to the agent, according to the transition matrix (20) which is defined by supervisory as a agent pre-knowledge. The agent can derive the OPL model for its own all tasks and then connect to the CPLEX and schedule it to reach an optimal sequencing (shows in the right picture) and calculate the new t''_2 completion time.

1. In our proposed algorithm, the neighbor agents just knows about each other. For example agent 2 knows about states (the three main states are introduced in the following paragraphs) of agent 1 and 3, and he doesn't know about agent 4 or 5. Therefore, this is one of the advantages of this algorithm that can solve the problem distributedly with minimum agent knowledge of the whole problem.
2. In our proposed algorithm, each agent just communicates with its neighbors. So, it means that we can control the amount of communicational cost efficiently.
3. In our algorithm, agents has a sequential identifier that assigned with the supervisory layer in advance.
4. In designing this algorithm, three steps are considered. The first one is initialization and sequencing the overlapping tasks, the second one is to start negotiating and allocating tasks by agents, and the third one is backtracking after allocating all the tasks to agents if necessary (reaching inconsistent solution).
5. In our algorithm, there is no pre-defined priority among agents (no leader no follower). This means that when the algorithm is started in allocation step (2nd step), the agent with the low identifier number (QCA1) should start, and when the algorithm is started the backtracking step (3rd step), the agent with the high identifier number should start it. During running the algorithm, just one of the agent is running and another neighbors should be waiting until receive run!() by the agent. Therefore, at any time each agent is running has, the highest priority (is master) and its neighbors are slaves. So, this can guarantee solving the algorithm robustly or stability of the algorithm in the sense of lacking each agent could be guaranteed. Assume if we want to pre-defined an agent as a leader of the team (like as ABT which agent 1 is leader and others are follower), when the leader gets into trouble, the whole of agents will get into trouble and cannot continue the algorithm (please be note that some ways have been proposed to overcome to this issue such as the lower priority agent will lead the team, that needs to consider to the algorithm in real problem solving that leads to complexity of the algorithm becomes high)
6. We can run this algorithm during the process of container handling by quay cranes (On-line optimization). With the on-line optimization, we can guarantee robust optimization of QCSP. In the sense of any uncertainties such as:
    a. Quay crane operator faults
    b. Slowness/fastness of handling the container
    c. Quay crane faults
    d. Shortage of pre-defined containers
    e. …

We will see the scheduling time will be different of doing the jobs. If this difference larger than a pre-defined constant, then the algorithm will be run with the remaining tasks that should be handled by cranes.

In this algorithm, three main phases for agents have been considered. These phases called *active*, *passive*, and *done!*. At first, during initializing the agents, all the agents have been considered as active. In this phase, agents try to allocate all the tasks with respect to the algorithm and try to reach a consistent set of values regarding to the problem constraints.

The Extended ABT works by constructing *consistent_set* and *inconsistent_set* of values and maintain these sets as a structure called *agent_view*. The *consistent_set* is referring to the set of values that belong to the variables which are comply the problem constraints, on the other hand, the *inconsistent_set* is the set of values that are against the problem nature. The agent value selection is stored in *agent_view* and after complete allocating or during selecting the values, agents can check the *agent_view* and decide that the set is consistent of not.

In the proposed algorithm, each agent has its own identifier (a sequential number regarding to its position), also agents have not prioritized over other agents. This means that, each agent has its own duty to making its decisions and no agent can govern it and optimize its solution (just some requests commands are allowed to reduce communications by its neighbors, which we will discuss later).

1) Initializatoion (Fig. 25)

On startup, all of the overlapping and non-overlapping tasks are assigned for each agent by the supervisory layer when the port automation system knows about incoming vessel. The overlapping tasks are divided into two sets, which are regarding to their position. The overlapping tasks which are located between agents x_i and x_j, will be within the set $\bar{D}_{i,j}$. After receiving all of the related tasks by each agent from the higher layer of port automation system, each agent tries to schedule all the non-overlapping tasks $\bar{D}$. In this way, the optimization model of each agent problem will be prepared by each agent in the OPL format. Then each agent will try to connect to CPLEX solver and then minimize the cost function using Branch and Bound search. After scheduling the exist non-overlapping tasks and calculate the completion time t_i by agent x_i, all the agent will be initialized sequenced overlapping tasks by the supervisory layer. The sequences have been ordered regarding to their location with respect to its neighbor agents and with valid tag numbers by the supervisory. The task ordering enables agents to allocate the overlapping tasks sequentially according their priority from each agent view. At last, the initialize procedure will be finished by resetting agent_view of QCAs and setting agent in active state. The agent_view stores the names, values, domains, and functional relationships of agents in their environment. By resetting the agent_view, all of the values and parameters for agent's neighbors will be set to zero. After initializing procedure, the first agent (QCA1) will be run by supervisory; in this case, all other agents are waiting to receive command from its neighbor. In this algorithm, just one of the agents is running at a time and other agents are waiting.

2) Checking agent_view (Fig. 26)

The decision of each agent is made by the *check_agent_view* procedure. In this procedure agent will decide with his knowledge himself or about *agent_view* of his neighbors and the logic. The *check_agent_view* procedure contains the algorithm to calculate the problem cost within each *agent_view*. In this way, after assigning each task by an agent and

Commented [Z74]:
RN: What is the state here?

IZ: Done

Commented [Z75]:
RN: Conflicts !

IZ: The identifier number not identified priority. It is just a sequence number. However, we know every algorithm has a start point which indicates where the algorithm process should be started.

Commented [Z76]:
RN: Not introduced yet !

IZ: Done

Commented [Z77]:
RN: Make sure that you always finish explain concept and then use that

Commented [Z78]:
RN: What is a good schedule?

IZ: In our distributed problem solving (D-QCSP), in initialization section, by arriving the data of tasks (non-overlapping and overlapping), each QCA tries to schedule their own (non-overlapping from $\bar{D}$ set) tasks [we can say this "pre-scheduling" section, this procedure can sequencing the tasks in a right order for handling by crane in minimum time]. After that, by assigning any task by each agent, the agent tries to connect to CPLEX and re-schedule its allocated tasks to reach to a right order sequencing. [We can name this section as "post-scheduling". This section can perform in several way:

1-By assigning new task in each step of the algorithm and perform iteratively until all the tasks are allocated to QCAs
2- By assigning all the tasks to all QCAs, after finishing the allocation, each QCA can perform scheduling by connecting to CPLEX
3- In the above two different procedure;
    a.we can assume the non-overlapping tasks are has the same order and the new assigning tasks can place between them,
    b.Or we can consider the previous scheduled sequence (non- or overlapping tasks) are fixed and by assigning a new task, the algorithm search the order and find a good place between two tasks.
    c.Or we can consider all the allocated tasks (new and previous) have to re-schedule. (Take much time and not necessary)

Commented [Z79]:
RN: Can you write down the formulation of the optimization problem that you give to CPLEX?

IZ: I have included the information in III.C problem solving description

Commented [Z80]:
RN: How determined?

IZ: The priority of any agent over its neighbors is determined due to its neighbor. In any step each agent or its neighbor can determine which agent should have the most priority for allocating task. The algorithm also approve that.

Commented [Z81]:
RN: What are the problem cost?

IZ: As we described, here the problem cost is to reach:
1- The minimum completion time to handling all the tasks by all cranes
2-The minimum completion time to handling tasks of each crane
...

updating the *agent_view*, each agent try to optimize its own problem to reach its minimum completion time t_i and also compare its t_i to the neighbors (t_{i-1}, t_{i+1}) to ensure that the tasks will be handled by QCAs at near same time $(t_1 \approx t_2 \approx \cdots \approx t_i \; for \; i \; agents)$. The purpose of that to ensure equalizing the allocated tasks among cranes, since, we don't want to handle most of the tasks by some cranes and handle other few tasks by other cranes. We try to find a solution that guarantees distribution of tasks among cranes as same as possible. For example in Fig. 31 we can obviously see that QCA3 has completion time t_3 and QCA4 has completion time t_4 which $t_3 \gg t_4$ so, QCA4 can try to allocate the selected task 12 by QCA3 to reach $t_3 \approx t_4$. This is important to note that, the algorithm try to reach an equalized solution, but sometimes there are not enough overlapping task for an agent to get it from its adjacent agent during backtracking, so it can not allocate further task and its completion time could have large difference in comparison to its adjacents. We can divide *check_agent_view* procedure into three main procedures related to the three main states i.e. $active$, $passive$, and $done$! In the $active$ state, agent i try to compare his t_i with its neighbors (x_{i-1}, x_{i+1}) and he may allocate task from $\bar{D}_{i-1,i}$ or $\bar{D}_{i,i+1}$. All the agents will try to allocate the entire tasks at the first state ($active$). After finishing assigning the agents send no_value! to their neighbors and enter their state into $passive$. By changing their state, they will check the $sub\_solution$ consistency and do $backtrack$ if necessary by sending $request\_value$? to their neighbor and receive add_value! from their adjacent agent. The $backtrack$ steps will be continued until all the agents receive agreements due to the problem consistency. After each agent take the $backtrack$ action if necessary, he enters into $done$! state which means there is no more thing to do, unless, its neighbor forces to change back again into $passive$ mode. When, all the agents reach to $done$! state, the consistent solution is reached and the algorithm will be terminated, unless, the supervisor force to restart algorithm to reach another consistent solution.

3) Other Packets and Procedures (Fig. 25 and Fig. 27)

There are some packets and procedures are introduced for handling the algorithm. They are consists of packets that send by agents in order to inform or ask another agent about their actions. Also, several procedures are introduced to handle some part of algorithm. In order to scheduling the allocated task in each step, the $schedule\_values$ is defined, this procedure can run not necessarily in the main thread, since, it has may take more time and may interrupt agent negotiation process. Until, scheduling the values/tasks by agent, we can estimate the completion time at time of assigning new task by $t_i \leftarrow t_i + t_{d_i}$. Whenever the scheduling has been done, we can substitute the real t_i with the estimation.

• procedure **initialize**

1. initialize all overlapping ($\bar{D}$) and non-overlapping ($\tilde{D}$) tasks;
2. initialize $T_{threshold}$;
3. reset *agent_view* for all $QCAs$;
4. $x_i \leftarrow active, \forall i \in QCAs$;

// scheduling the non-overlapping tasks

5. construct optimization model of non-overlapping tasks $\tilde{D}$;
6. connect x_i to CPLEX;
7. minimize the cost using Branch and Bound;
8. calculate new completion time $t_{i(new)}$;
9. $t_i \leftarrow t_{i(new)}$; // update completion time

// ordering overlapping tasks with valid tag numbers

// for all $QCAs$ based on their location w.r.t neighbor agents

10. Initialize sequences of all overlapping tasks;

• **when** received (**run!**, (x_j, s_j, d_j, t_j)) **do**

// update *agent view*

add (x_j, s_j, d_j, t_j) to *agent_view*;

do check_agent_view;

end_ do;

• **when** received (**request_value?**, (x_j, s_j, d_j, t_j) , d) **do**

// update *agent view*

add (x_j, s_j, d_j, t_j) to *agent_view*;

Commented [Z82]:
RN: What is this?

IZ: This is the threshold time that difference of adjacent agents completion time should not be more than this value. $|t_i - t_j| < T_{threshold}$

Commented [Z83]:
RN: What is the optimization problem solved by CPLEX? Give the mathematical formulation.

IZ: I have included the information in III.C problem solving description

Commented [Z84]:
RN: How?

IZ: After connecting to CPLEX and scheduling the tasks, the new completion time is calculated. Since the order of handling tasks affect directly on completion time, so after scheduling the tasks, and
I am now implementing it by Java

Commented [Z85]:
RN: What do you mean?

IZ: Before starting the algorithm all overlapping tasks should have tag numbers. These numbers indicate priority of each task regarding to each agent, so, each agent try to allocate tasks from the highest priority to the lowest.

Commented [Z86]:
RN: Received by who? Agent j?

IZ: The "received" procedure is an interrupt procedure which indicate when a packet is received that has for example "run!" function by the near agent. In this algorithm all the argument received from neighbor agent indicates by "j".

Commented [Z87]:
RN: What do these variables mean?

IZ:
x_j means agent j
s_j means state of agent j
d_j means all the allocated variables/tasks by agent j
t_j means the last calculated completion time of agent j

// check if this agent is $done!$ In backtrack procedure
if (x_i is $done!$) **do**
  $s_i \leftarrow passive$;
end do;
remove (x_i, d) the requested value (d) from *agent_view*;
send (**add_value!**, (x_i, s_i, d_i, t_i) , d) to x_j;
do schedule_values;
end_ do;

• **when** received (**add_value!**, (x_j, s_j, d_j, t_j) , d) **do**
// update *agent view*
add (x_j, s_j, d_j, t_j) to *agent_view*;
//add the requested value (d) to *agent_view*;
do select_value ($d \in \overline{D}_{i,j}$);
do check_agent_view;
do schedule_values;
end_ do;

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

Fig. 25 – Extended ABT algorithm (Part I)

• procedure **check_agent_view**
if $((t_i \leq t_{i-1}) \wedge (t_i < t_{i+1}))$ **do** **(i)**
    if (x_i is $active$) **do** (i-a)
        // first select non-selected value from right
        if ($\exists\, d \in \overline{D}_{i,i+1} | \; d \notin x_i, x_{i+1}$) **do** *(i-a-1)*
            do select_value ($d_i \in \overline{D}_{i,i+1}$);
send (**run!**, (x_i, s_i, d_i, t_i)) to x_{i+1};
            do schedule_values;
        // else select non-selected value from left
        elseif ($\exists\, d \in \overline{D}_{i-1,i} | \; d \notin x_{i-1}, x_i$) **do** *(i-a-2)*

do select_value ($d_i \in \overline{D}_{i-1,i}$);

\# **send** (**run!**, (x_i, s_i, d_i, t_i)) to x_{i-1};

do schedule_values;

// if no value exists to select neither from right nor from left

elseif ($\left(\nexists\, d_i \in \overline{D}_{i-1,i} \mid d \in x_{i-1}\right)$

$\wedge \left(\nexists\, d_i \in \overline{D}_{i,i+1} \mid d \in x_{i+1}\right)$) **do** *(i-a-3)*

send (**no_value!,** (x_i, s_i, d_i, t_i)) to the neighbors;

$s_i \leftarrow passive$; //x_i become $passive$

end do;

// if all values were selected by their near agents

elseif (x_i is $passive$ $\wedge$ all neighbors are $passive$) (i-b)

// first select non-selected value from left

if ($\exists\, d \in \overline{D}_{i-1,i} \mid d \in x_{i-1}$) **do** *(i-b-1)*

do backtrack($d_{i-1} \in \overline{D}_{i-1,i}$);

elseif ($\exists\, d \in \overline{D}_{i,i+1} \mid d \in x_{i-1}$) *(i-b-2)*

do backtrack($d_{i+1} \in \overline{D}_{i,i+1}$);

elseif ($\left(\nexists\, d_i \in \overline{D}_{i-1,i} \mid d \in x_{i-1}\right) \wedge \left(\nexists\, d_i \in \overline{D}_{i,i+1} \mid d \in x_{i+1}\right)$)

$\wedge$ all neighbors are $passive$) *(i-b-3)*

send (**no_value!,** (x_i, s_i, d_i, t_i)) to the neighbors;

$s_i \leftarrow done!$;

end do;

elseif (x_i is $passive$ $\wedge$ any $neighbor$ is not $passive$) (i-c)

if ($neighbor\ x_j$ is $active$) **do** //right then left *(i-c-1)*

send (**run!**, (x_i, s_i, d_i, t_i)) to $neighbor\ x_j$;

??? **else if** ($neighbor\ x_j$ is $done$) *(i-c-2)*

send (**no_value!,** (x_i, s_i, d_i, t_i)) to the neighbors;

\# $s_i \leftarrow passive$;

end do;

end do;

// check agent time within its neighbors

elseif ($(t_i < t_{i-1})$ $\wedge$ $(t_i \geq t_{i+1}) \wedge (x_i$ is $active)$) **(ii)**

if (x_{i+1} is $active$) **do** (ii-a)

send (**run!**, (x_i, s_i, d_i, t_i)) to x_{i+1};

elseif (x_{i+1} is not $active$) (ii-b)

do select_value ($d \in \overline{D}_{i,i+1}$);

send (**run!**, (x_i, s_i, d_i, t_i)) to x_{i+1};

do schedule_values;

end do;

elseif ($(t_i > t_{i-1}) \wedge (t_i < t_{i+1}) \wedge (x_i$ is $active)$) **(iii)**

if (x_{i-1} is $active$) **do** (iii-a)

send (**run!**, (x_i, s_i, d_i, t_i)) to x_{i-1};

elseif (x_{i-1} is not $active$) (iii-b)

do select_value ($d \in \overline{D}_{i-1,i}$);

send (**run!**, (x_i, s_i, d_i, t_i)) to x_{i-1};

do schedule_values;

end do;

// check agent time be greater than its neighbors

elseif ($(t_i > t_{i-1}) \wedge (t_i > t_{i+1}) \wedge (x_i$ is $active)$) **(iv)**

// check which agent time is greater

if ($(t_{i+1} > t_{i-1}) \wedge (x_{i-1}$ is $active)$) **do** (iv-a)

send (**run!**, (x_i, s_i, d_i, t_i)) to x_{i-1};

elseif ($(t_{i-1} > t_{i+1}) \wedge (x_{i+1}$ is $active)$) (iv-b)

send (**run!**, (x_i, s_i, d_i, t_i)) to x_{i+1};

// first select from right

elseif ($x_{i+1}$is not $active$) (iv-c)

do select_value ($d \in \overline{D}_{i,i+1}$);

send (**run!**, (x_i, s_i, d_i, t_i)) to x_{i+1};

do schedule_values;

// second select from left

elseif ($x_{i-1}$is not $active$) (iv-d)

do select_value ($d \in \overline{D}_{i-1,i}$);

send (**run!**, (x_i, s_i, d_i, t_i)) to x_{i-1};

do schedule_values;

end do;

elseif (x_i is $passive$) **(v)**

// check if still any neighbor agent is being in active state

if (x_{i+1} is $active$) **do** // first right then left (v-a)

send (**run!,** (x_i, s_i, d_i, t_i)) to x_{i+1};

elseif (x_{i-1} is $active$) **do** (v-b)

send (**run!,** (x_i, s_i, d_i, t_i)) to x_{i-1};

// check if all neighbors has no possible value and check problem's consistency

elseif ($\left(\nexists\, d_i \in \overline{D}_{i-1,i}\right) \wedge \left(\nexists\, d_i \in \overline{D}_{i,i+1}\right)$) (v-c)

// check *sub_solution* consistent with neighbor x_j – first *left* then *right*

if ($|t_i - t_j| > T_{threshold}$) **do** *(v-c-1)*

if (x_j is $done!$) **do**

$s_j \leftarrow passive$;

end do;

//request the most appropriate task from neighbor

do backtrack($d_j \in \overline{D}_{i,j}$);

elseif (this *sub-solution* is consistent) *(v-c-2)*

send (**no_value!,** (x_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 (x_i is $done!$) **(vi)**

// check if still any neighbor agent is being in active/passive state

if (x_{i-1} is $active$) **do** // first left then right (vi-a)

send (**run!,** (x_i, s_i, d_i, t_i)) to x_{i-1};

elseif (x_{i+1} is $active$) **do** (vi-b)

send (**run!,** (x_i, s_i, d_i, t_i)) to x_{i+1};

elseif (x_{i-1} is $passive$) **do** // first left then right (vi-c)

send (**run!,** (x_i, s_i, d_i, t_i)) to x_{i-1};

elseif (x_{i+1} is $passive$) **do** (vi-d)

send (**run!,** (x_i, s_i, d_i, t_i)) to x_{i+1};

elseif (all neighbors are $done$!) (vi-e)

consistent_subset ← *this_solution*;

broadcast terminate algorithm;

// agents may able to find another *consistent_solution*

// and compare with *this_solution*

//so they may restart algorithm again

end do;

end do;

Fig. 26 – Extended ABT algorithm (Part II)

• procedure **backtrack**($d_j \in \overline{D}_{i,j}$)

inconsistent_subset ← *this_solution*;

//check if neighbor x_j can remove the value

if (x_j is $passive$) **do**

if ($\exists\ d_j \in \overline{D}_{i,j}$) **do**

remove *this_solution* from *consistent_subset*;

send (**request_value?**, (x_i, s_i, d_i, t_i) , d_j) to neighbor x_j;

elseif ($\nexists\ d_j \in \overline{D}_{i,j}$)

send (**no_value!**, (x_i, s_i, d_i, t_i)) to the neighbors;

$s_i \leftarrow done!$;

end do;

end do;

• procedure **select_value**($d \in \overline{D}_{i,j}$)

//select a most appropriate value;

if ($\exists\ d \in \overline{D}_{i,j}$) **do**

add (x_i, d_j) to *agent_view*;

elseif ($\nexists\ d \in \overline{D}_{i,j}$)

send (**no_value!,** (x_i, s_i, d_i, t_i)) to the neighbors;

s_i ← passive;

end do;

// update an estimation of completion time

$t_i \leftarrow t_i + t_{d_i}$;

• procedure **schedule_values**

// start scheduling after allocating a task

// this procedure can be run not in main thread

// in order not to interrupt agent negotiations

construct optimization model of recent allocated tasks;

connect x_i to CPLEX;

minimize the cost using Branch and Bound;

calculate new completion time $t_{i(new)}$;

// update completion time

$t_i \leftarrow t_{i(new)}$;

Fig. 27 – Extended ABT algorithm (Part III)

E. Example of Extended ABT

I try to depict the proposed algorithm within a simple example. In this example, each black rectangle represents a non-overlapping tasks and each white rectangle represents an overlapping task. In this example, for simplicity we assume each task takes one unit time for handling (so the heights of rectangles are considered the same). Fig. 28 illustrates our example after initialization procedure. All the QCAs are initialized from supervisory layer and they scheduled their non-overlapping tasks and calculated their completion time t_i ($i = 1,2,3,4$). For example, $t_1 = 15, t_2 = 12, t_3 = 12, t_4 = 15$. Furthermore, all of the overlapping tasks are sequenced by supervisory regarding to their neighbor agents. For example $task$ 1 is more important than $task$ 2 for QCA1 and vice versa for QCA2. Alternatively, $task$ 3 to $task$ 10 are sequenced for QCA3, while, $task$ 10 to $task$ 3 are important for QCA3 regarding to the tasks position and completion time. Also, threshold time is assumed $T_{threshold} = 1$ for this example and all the agents are set in $active$ state.

The algorithm is started from the QCA1, since agents have no knowledge from their neighbors and consider all of their time and tasks are zero. So according to the algorithm QCA1 try to send run! to its neighbor including its knowledge. QCA2 will receive the run! from QCA1 and sent run! to QCA3, and so on until the last agent (QCA4) receives the run!. Now all of adjacent agents have their knowledge and can decide further decisions.

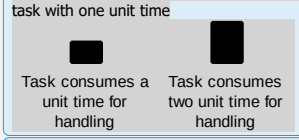
After that, QCA4 can compare its own completion time with its neighbor (QCA3). Since, $t_3 < t_4$ he will send run! to QCA3 for allocating a task. After receiving run! by QCA3, he compares time, so $t_2 \leq t_3 < t_4$ and he will assign an overlapping task with the less cost arbitrary (first from the right) and allocate the most appropriate one i.e. $task$ 11. Then, he compares time, so $t_3 < t_4$ and $t_2 < t_3$ and he will send run! to QCA2. QCA2 compares its time same as previous and then try to assign the most appropriate task i.e. $task$ 3. The algorithm will be running according to Fig. 30 until all the tasks are allocated and all the agents send no_value! to their neighbors and change their state from $active$ to $passive$ mode.

If we want to explain this problem in the sense of the described DCOP formalization in section B, We find that, the $tuple\ \langle \mathcal{A}, \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$ will be defined as; the set of agent is $\mathcal{A} = \{QCA1, QCA2, QCA3, QCA4\}$, the set of variables related to the quay cranes is $\mathcal{X} = \{QC1, QC2, QC3, QC4\}$ as we see, number of agents is the same as number of quay cranes in this problem, so each agent (QCA) defined by a related variable (Quay Crane), the set of domain of values/tasks for the agents/variables is $\mathcal{D} = \{D_1, D_2, D_3, D_4\}$ and as we know the domain consists of two different values/tasks i.e. two sub-domains; $\mathcal{D} = \{\bar{\mathcal{D}} \cup \overline{\mathcal{D}}\}$ where $\bar{\mathcal{D}} = \{\bar{d}_1, \bar{d}_2, \bar{d}_3, \bar{d}_4\}$ refers to the non-overlapping tasks and $\overline{\mathcal{D}} = \{\bar{d}_1, \bar{d}_2, \bar{d}_3, \bar{d}_4\}$ referes to the overlapping tasks for related agent/variable, for example $\bar{d}_2$ refers to all the overlapping tasks that can be assigned with QCA2. Please note that, we say "can be" since the overlapping tasks are in domain QCA2 and all of them will not mandatory be assigned by QCA2. For simplicity we can consider the $\overline{\mathcal{D}}$ is also consists of several sub-domains $\overline{\mathcal{D}} = \{\overline{D}_{1,2}, \overline{D}_{2,3}, \overline{D}_{3,4}\}$, these sub-domains are referring to the overlapping tasks between two adjacent agents. For example, $\overline{D}_{2,3}$ refers to the overlapping tasks between QCA2 and QCA3. From another point of view, we can write down: $\bar{d}_1 = \overline{D}_{1,2}$, $\bar{d}_2 = \{\overline{D}_{1,2} \cup \overline{D}_{2,3}\}$, $\bar{d}_3 = \{\overline{D}_{2,3} \cup \overline{D}_{3,4}\}$, and $\bar{d}_4 = \overline{D}_{3,4}$.

In this example shows in Fig. 28, $\bar{d}_1 = \{1,2\}, \bar{d}_2 = \{3,4,5,6,7,8,9,10\}, \bar{d}_3 = \{3,4,5,6,7,8,9,10,11,12\}, \bar{d}_4 = \{11,12\}$ also $\overline{D}_{1,2} = \{1,2\}, \overline{D}_{2,3} = \{3,4,5,6,7,8,9,10\}, \overline{D}_{3,4} = \{11,12\}$.

Commented [Z88]:
RN: What do you mean?

IZ: This is not critical. I meant, in depicting the example, if we want to be accurate to show the tasks, we can vary the heights of tasks in illustration. For example, task with handling time two unit time should be show two times larger than a task with one unit time



Commented [Z89]:
RN: Define which exactly you mean with "sequencing" in the sequencing of the paper

IZ: Regarding to comment Z15

Commented [Z90]:
RN: Consisting of what?

IZ: Its knowledge is the argument of the run! function introcuced in Fig. 20 – Extended ABT algorithm (Part II). This argument is almost common to all the functions i.e. (x_j, s_j, d_j, t_j)

Commented [Z91]:
RN: Like?

IZ: In this state i.e. active state, always there are two possible action can be taken by the agent who receive packet from its near agent. 1- check if there is a possible task that can be assigned and assigns it.
2- check if there is not possible assigning task, then pass its ball to its neighbor agent who can assign task.

Commented [Z92]:
RN: What?

IZ: After the above pre-processing action by all the agent and evaluation of their neighbor agents and exchange their knowledge.

Commented [Z93]:
RN: What does QCA3 send that?

Commented [Z94]:
RN: Who?

IZ: QCA3 (The agent of quay crane 3)

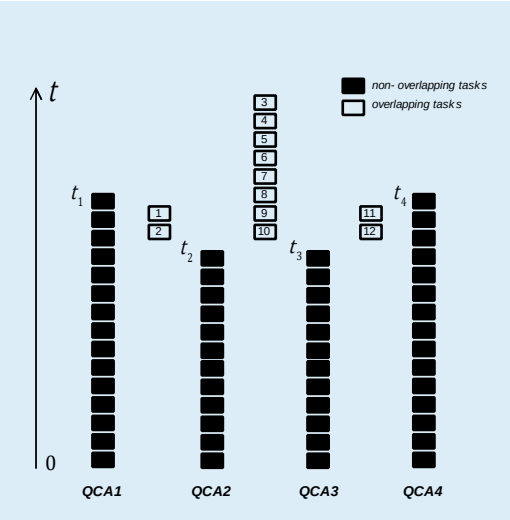


Fig. 28 – The illustration of the example after initialization by the supervisory layer and before the algorithm starts

The set of constraints $\mathcal{C} = \{c_1, c_2, c_3, c_4\}$ for the problem can be described as follow:

1- Difference of each adjacent agents should not more than a specific threshold time $|t_i - t_j| > T_{threshold}$ if possible. This constraint is checked by each agent in each iteration of the algorithm in the third step i.e. *backtracking step*. If this limitation violates by assigning new task, the agents try to backtrack the assigned tasks until reaching to the consistent solution.

2- Another constraint can be described as cost of task allocation to each quay crane agent. This cost referring to the distance of each task regarding to each quay crane and is regarding to the task ship bay number received from supervisory layer as we described before. We use hard constraint for this example and the string formation is depicted in Fig. 29.

In this problem, we are looking to optimal assignment $A^* = \{d_1, d_2, d_3, d_4 | d_i \in D_i\}$ through all possible assignments set which minimizes global problem cost **Error! Reference source not found.**.

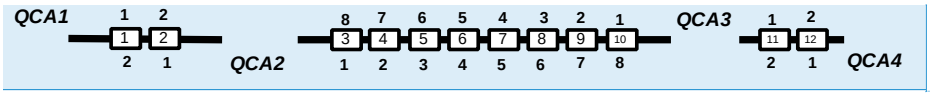
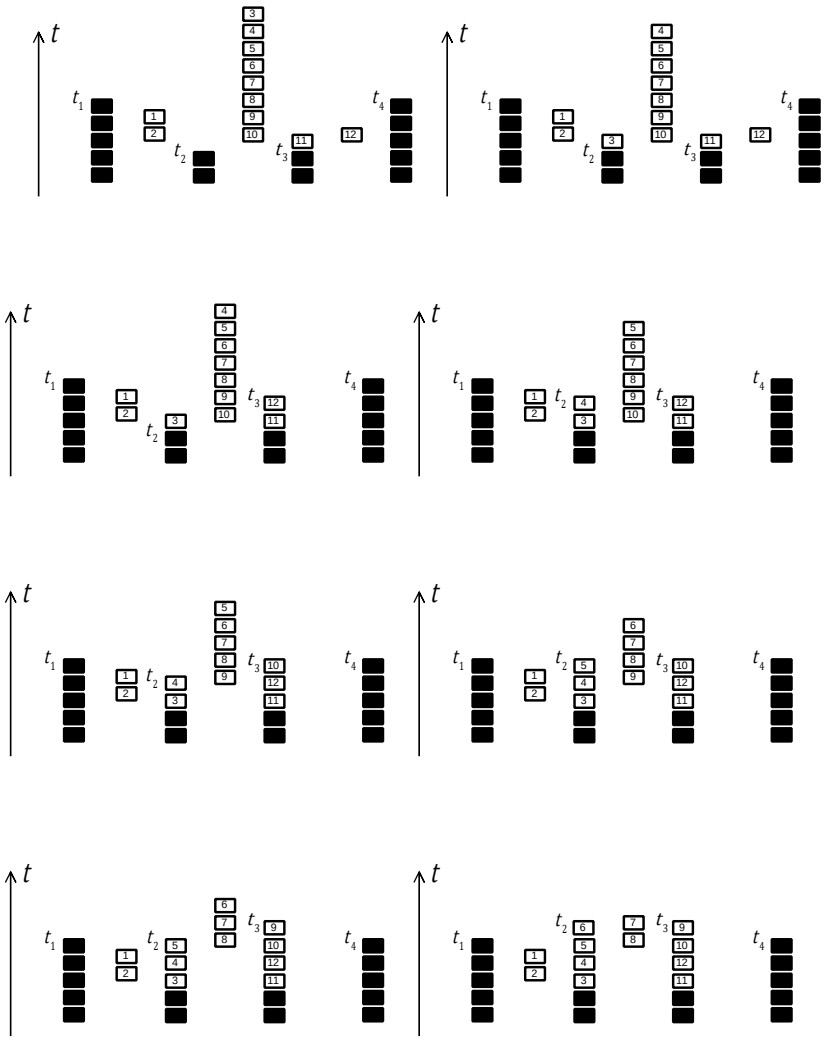


Fig. 29 – The string formation of the example, illustrated in Fig. 28

Commented [Z95]:
RN: How is the cost function/constraints/variables define?

IZ: Added

The following illustrations can explain how the algorithm can solve the example problem more precisely:


t
t1
t2
t3
t4
1
2
3
4
5
6
7
8
9
10
11
12

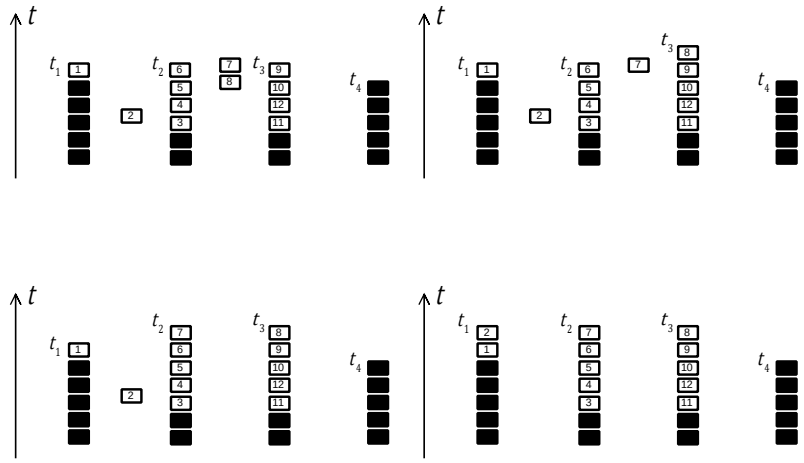


Fig. 30 – The algorithm is shown when agents in *active* state and finally send *no_value*! to their neighbors and change their state into *passive*

After all the agent states are changed into *passive*, the algorithm will be continued from the last agent i.e. QCA4. In this case, QCA4 compares his time with his neighbor $|t_4 - t_3| = 2 > T_{threshold} = 1$. Therefore, he will backtrack the allocation by sending *request_value*? to QCA3 (See Fig. 31). By receiving the *request_value*? by QCA3, he remove the requested value from his *agent_view* and then send *add_value*! to his neighbor to add the requested value to its *agent_view*. Both of the adjacent agents will schedule their allocated values after backtracking and calculate their new completion time. In this case, $t_4 = t_3$ and $|t_4 - t_3| = 0 < T_{threshold} = 1$, and this *sub_solution* between QCA3 and QCA4 is consistent. QCA4 send *no_value*! to QCA3 and set his state into *done*!. By receiving *no_value*! by QCA3, he check the *sub_solution* consistency with other neighbor i.e. QCA2. The new time are become $t_2 > t_3$ and $|t_2 - t_3| = 1 = T_{threshold}$. Therefore, he enters into *done*! state and send *no_value*! to QCA2, and he checks consistency and continue the above until the last agent enters *done*! and algorithm is terminated.

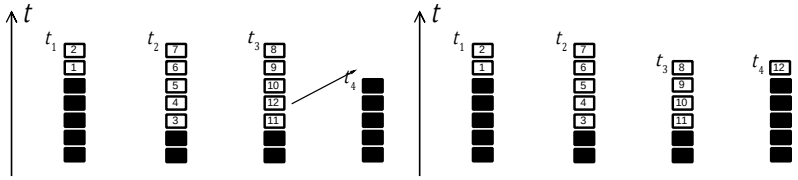


Fig. 31 – The algorithm is shown when all of the agents enter into *passive* state. QCA4 try to backtrack the allocation by request the most appropriate value from QCA3.

V. Implementation of Dynamic QCSP Approach of proposed MAS Structure

MAS models are oriented toward autonomy, cooperation and dynamic planning. The model is a dynamic distributed problem solver because as soon as port data is updated for example by arrival of a containership, the MAS model starts to solve the problem iteratively among distribution of agents. Our aim in this work is to construct the MAS framework to solve the D-QCSP with varying overlapping area. In this way, distributed solution is selected instead of traditional centralized one. In the conventional problem solving, a central processor usually gather information and solve complete problem instantly to reach solution. On the other hand in distributed manner, solving problem is done by decomposed the whole optimization problem into several parts called sub-problems. In this case, each agent (QCA) assigns to a specific sub-problem of D-QCSP

i.e. all non-overlapping tasks are assigned to QCAs first by supervisory layer. After the task pre-assignment procedure by the higher layer to QCAs, they start to communicate with each other to reach agreement over overlapping tasks.

In the implementation of MAS, selecting agent behavior among various behaviors is an important thing. An agent can execute several behaviors concurrently. According to the QCAs functions, *Finite State Machine (FSM)* behavior is selected mainly and other selected behaviors are playing role as sub-behavior such as; *One-shot* behavior for complete in one execution phase tasks, *Cyclic* behavior for never complete tasks, *Ticker* behavior for repetitive tasks in a specific period of time, *Composite* behavior include *Sequential* behavior and *Parallel* behavior for doing complex crane tasks.

A. *Implemented Agent Structure*

Each QCA has the structure presented in, can be described using agent-oriented programming paradigm. This 3-layered structure described below:

1) *Information Layer*

This layer has to have such pieces of knowledge regarding to agent task and its adjacent cooperated agents. This layer consists of *Data Base and Logger Agent (DBLA)* and *Sorting and Analysis* component, which update dynamically from supervisory system based on updated information of incoming vessels or handling containers.

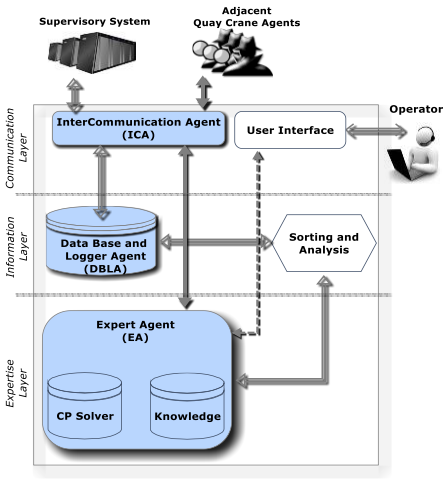


Fig. 32 – Quay Crane Agent (QCA) structure with 3-layers

2) *Expertise Layer*

This layer represents the intelligent part of the agent. It is based on optimizing the D-QCSP according to the knowledge which receives from *knowledge Layer* and *Communication Layer*. This layer consists of *Expert Agent (EA)* for computing data, which receives from *InterCommunication Agent (ICA)* and *Data Base and Logger Agent (DBLA)* after Sorting and Analysis it.

The information of QCA about its physical environment is embedded in *Knowledge* block mostly provided by supervisory of the port automation system within EA. The CP Solver engine is also embedded into this layer in order to solve the tasks ordering problem for a single machine (Fig. 8) ~~QCSP~~ dynamically. All decisions of QCAs are made in this layer by updating data of *knowledge* block. All of incoming data from supervisory and other QCAs sorted into knowledge block of EA for further decision making.

3) *Communication Layer*

The agent's communication is along the *physical environment*. The communications among agents is a very important issue, so reliable and robust communication platform should be guaranteed by this layer. There are various industrial communication protocols which can be used as the reliable basis for exchanging message among agents (e.g. communication network protocols used for process or industrial automation like as; MODBUS, Industrial wireless ETHERNET, or numerical control protocols). In this work, the transmission of information through the agent wireless network is based on *Transmission Control Protocol/ Internet Protocol (TCP/IP)* – a suite of protocols used in the design of most wireless technology through *Radio Frequency (RF)*. The structure of exchanging message is a set of key values written in FIPA-ACL. The content of the message is expressed in a content language, such as FIPA-SL or FIPA-KIF, and content expressions can be grounded by referenced ontologies [34]. Regarding on the protocols needed for the applications are implemented, types of messages are introduced as *FIPA-query*, *FIPA-Request*, *FIPA-Request-When*, *FIPA-Cancel-Meta-Protocol*, and *FIPA-Contract-Net.*

FIPA-Contract-Net can handle one of the most important tasks between the agents i.e. coordination of negotiation among agents. In this case, we have to has at least a pair of agents called *initiator* and *participant* agents. The initiator agent wishes to have some task performed by one or more other participant agents and further wishes to optimize a function that characterizes the task. This characteristic is commonly described as scheduling cost in our D-QCSP. By assigning new tasks, each QCA try to find a proper sequence to handle the allocated tasks as minimum time as possible. For example, in Fig. 30, QCA3 tries to allocate tasks both from both right and left hand sides. If he wants to handle its tasks by the assigning order, he will be moving more than enough and consequently, the completion time of QCA3 will be highly increase. So, after each iteration, an scheduling should be performed among the new and past allocated tasks to find an optimized way for handling tasks. This can be done by making an OPL model (Fig. 8) in each iteration by each QCA, then connect the agent to CP Solver of CPLEX, and finally receive a scheduled order of handling tasks and update its completion time. Until now, we describe an objective to reduce completion time of each QCAs in each iteration and receiving the latest completion time of each QCAs.

Commented [Z96]:
Note:
In our new simulation we didn't see FIPA-Contract-Net anymore.
We just use simple "FIPA-Request" protocol to communicate between agents.
1-In Extended ABT algorithm we assumed just two adjacent QCAs can communicate to each other.
2-In Extended ABT we assumed there is no master/slave config for agents, like Fig. 32 which has initiator (master) and participants (slaves).

Note:
We can also assumed this protocol for each pair agents who want to communicate to each other. In this case, initiator and participant can be each two adjacent agents. So this model also can be valid for our algorithm.

Commented [RN97]:
RN: Please elaborate on this. Make more clear what you mean.

IZ: added

Another objective that is important is, reaching to an equivalent completion time at the end of allocation as much as possible for all quay cranes. This is important to reach the minimum time as much as possible for the whole quay cranes by equalizing tasks and completion time to cranes, since we would not a crane working much more than another cranes. So, these issues are the objectives of the problem and all QCAs have to consider them meanwhile of their communications. ~~For a given task, any number of the participant QCAs may respond with a proposal; the rest must refuse. To implement the ABT algorithm which is introduced before, to handle the distributed algorithm by QCAs, the best protocol selection would be Contract-Net among others. In spite of some major disadvantages like enormous compexity in modern contract-net and amount of exchanged packets between agents, reliability, and ability to support complex negotiations between agents, and availability its library in JADE are the advantages of this protocol to become a best candidate to implemrent ABT algorithm. In Fig. 33 interaction between initiator and participant QCAs according to this protocol is shown.~~

In this study, we implemented Extended Asynchronous BackTracking (ABT) Algorithm under FIPA-Contact-Net Interaction Protocol (IP) introduced by FIPA standards. This protocol has been fully implemented by JADE middleware with its behavior and handlers for all successes and failures of message.

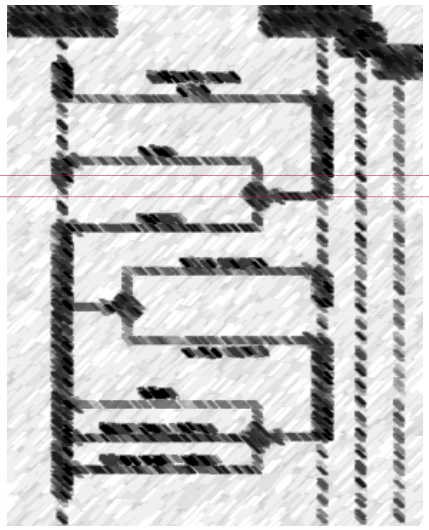


Fig. 33 – The FIPA-Contract-Net Interaction Protocol used for implementing the ABT algorithm for solving QCSP as a DCOP

Commented [RN98]:
RN: Give an example of a task

IZ: removed

Commented [RN99]:
RN: Give an example of a proposal.

IZ: removed

In this section we propose Extended ABT - a modified version of ABT - which enables QCAs to interact with each other to reach the problem minimization makespan (i.e. minimize overall scheduling of handling containers to/from ship). One of the important disadvantages of ABT is, in large-scale problem with a large-amount of agents, a single mistake would become fatal. This major failure can occurred when for example one or more agents are being absent or they miscalculate their assigned tasks and inform false data to other agents. In the D-QCSP, since there are usually three to seven QCAs are considered, that issue will not important. Also, in the case of having overlapping areas between adjacent crane agents, this algorithm would be implemented. On the other hand, each QCA has its own non-overlapped tasks which handle them by its own. In the proposed procedure which will be described later based on ABT algorithm, we find a solution for solving scheduling problems distributively. Although the centralized problem solving has the advantage of reaching the optimal solution, it takes hard to find solution of the problems in a reasonable time. In distributed problem solving we are dealing with breaking down the overall problem into sub-problems and then solving the sub-problems with agent to reach a near-optimal solution in a reasonable time in case of high amount of tasks. In the following algorithm, we modify the ABT algorithm for our crane scheduling problem which can be suitable implemented within FIPA-Contract-Net IP that detailed illustrates in Fig. 33. Also, in This algorithm can guarantee working agents together cooperatively and computations among them collectively to reach near-optimal solution

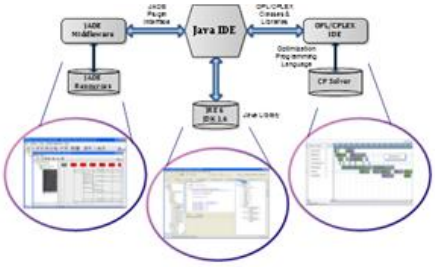


Fig. 34 – MAS implementation software design and Interactions among packages

.

VI. Simulation of Dynamic D-QCSP

The simulation is implemented in Java IDE with an agent-oriented programming approach. Java IDE has three connections to other resources and middleware. For running agents under IDE in a consistent environment like as JADE, the "JADE plugin interface" was used. That interface allows programmer to connect to the large library of JADE Middleware and

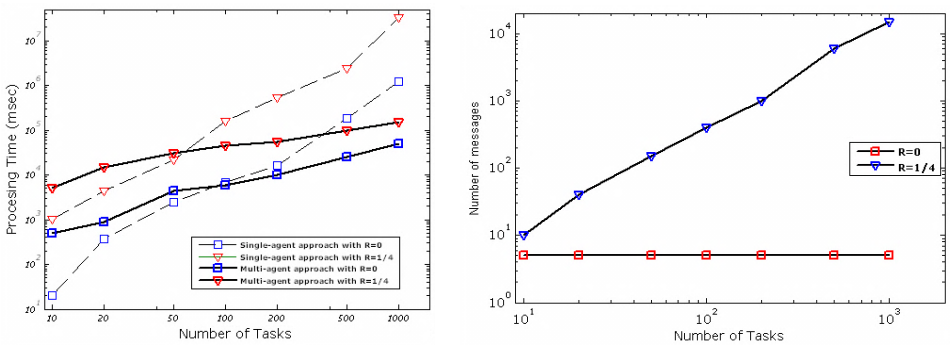


Fig. 35 - Left: Showing the execution performance with two methods and their comparison, Right: Showing communicational cost for MAS

its resources to control agents entity, interactions, mobility, behaviors, and so on. The JADE agent platform, provides high performance of a distributed agent system implemented with the Java language. In particular, its communication architecture offers flexible and efficient messaging, also its leveraging state-of-the-art distributed object technology which is embedded within Java Runtime Environment (JRE). On the other hand, each agent -at the time of incoming new data and need to optimize new problem- can connect to CP solver and scheduling its environment dynamically.

In this section two types of QCSP is implemented. For the first type overlapping area has not considered so $R = 0$, and for the second QCSP $R = 1/4$ is considered. We implement the single-agent and multi-agent scheme for both types of QCSP and D-QCSP. We show that how increasing the number of tasks for both types of QCSP can affect on the overall processing time to reach solution. Fig. 35-Left shows a comparison between a single-agent and a multi-agent scheme for $R = 0$ and $R = 1/4$. Since in a centralized scheme the optimal solution for all tasks is considered, increasing number of tasks will highly affect on processing time. In contrast, in the MAS scheme all non-overlapping tasks are optimized first, then overlapping tasks are assigned to QCAs. This procedure will divide tasks into two subgroups and can effectively reduce the amount of processing in a reasonable time. In Fig. 35 - Right, the communicational cost for the multi-agent scheme is shown. For $R = 0$ the amount of messaging among agents is being constant in case of increasing number of tasks. On the other hand, for $R = 1/4$ the amount of messages between agents will be increased dramatically by growing the number of tasks.

The problem is implemented and results are shown in Fig. 35 for a number of tasks. A comparison between a traditional system (centralized) and MAS results is depicted. Also computational cost is mentioned in case of increasing number of tasks.

Commented [RN100]:
RN: What problem exactly? Give parameters (e.g. of load, number of agents, number of containers, cranes, etc.)

IZ: Added

On the other side, user can connect to CP Solver through its embedded libraries which enable us to call the OPL model directly, and solve the model remotely without using the stand-alone OPL IDE.

MODIFICATIONS

In this section, we extend our previous simulations. We create different tasks with different overlapping index $R_{i,j}$ for 4 quay cranes and simulated in both centralized QCSP and Distributed QCSP scheme and make a comparison of them. The selected values for all tasks are 10, 20, 50, 100, 200, 500, and 1000. For simplicity we consider all $R_{i,j}$ for diffenrent quay cranes are the same, i.e. $R_{1,2} = R_{2,3} = R_{3,4} = R$. In this case the simulation is performed for $R = 0, R = 1/4, R = 1/2,$ and $R = 1$. According to (31), the overlapping ratio R refers to the shared tasks between cranes that have to be performed.

QCSP:

In QCSP simulation, we select the OPL model same as 5th Scenario (Fig. 18) which is explained the Modified QCSP (16)-(30), with all the constraints (explained in Table 1). In this simulation we create a transition time matrix $T_{n\times n}$ (34) for every number of tasks (e.g. for 50 tasks, we have to create $T_{50\times 50}$ and for 1000 tasks we have to create $T_{1000\times 1000}$). Moreover, for each overlapping ratio (R), we have to define overlapping tasks for each QCA, i.e. for case of 1000 tasks for all QCAs with different R, first for simplicity, we divide all the tasks among all agents equally, so each agent will have 250 tasks. After that for $R = 1/4$, there is no tasks to share, and all the tasks should be handled by each agent individually (overlapping tasks is zero and non-overlapping tasks is 250), $R = 1/4$, there are almost 60 tasks is shared between each adjacent quay cranes and all other tasks are assumed as non-overlapping tasks, for $R = 1/2$ there are 125 tasks and for $R = 1$, all the tasks for each QCA will be overlapped with its neighbor agents and there is no non-overlapping tasks. All the above tasks divisions can be performed by tuple "`Modes`" in Fig. 18. This tuple defines which tasks "`Modes.taskid`" should be performed by which crane(s) "`Modes.machine`" and also, we can define the task ship bay number by this tuple "`Modes.modeid`" which is an arbitrary number defined by supervisory layer of port container terminal.

D-QCSP:

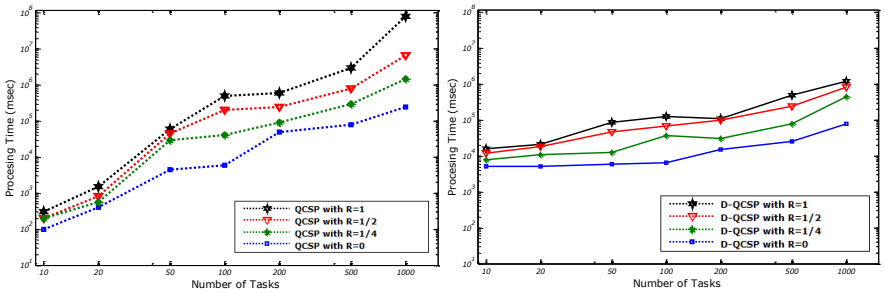


Fig. 37 - Left: Showing the execution performance for modified QCSP for different *R*, Right: Showing the execution performance for Distributed QCSP for different *R*

In D-QCSP simulation, the QCAs are developed under JADE using Java programming. Each agent can construct an OPL model (same as Fig. 8 with the mathematical form of (36)-(42)) by allocating tasks and scheduling the assigned tasks with connecting to CPLEX from Java and find the optimal sequence of handling tasks with the CP Optimizer library. QCAs are using the Extended-ABT (Fig. 25, Fig. 26, and Fig. 27) for coordinating and assigning the tasks. They initialized with the overlapping and non-overlapping tasks, ship bay number costs for each tasks, threshold time, transitional time matrix, and so on.

In the following figures, we bring the simulation results. In Fig. 36, two main approaches i.e. centralized QCSP approach and the distributed version are depicted for different number of tasks and different overlapping ratios. In the left hand side, we can comparison the processing time for the centralized version of quay crane scheduling to reach to a near optimal solution. On the other hand, in the right hand side, since the calculations are divided over four quay cranes, the processing time generally is decreased more, specially when the number of tasks are become high. We can see the comparisons for every *R*, between the both approaches in Fig. 37. Finally, the communication among agent for each scenario has been depicted in Fig. 38.

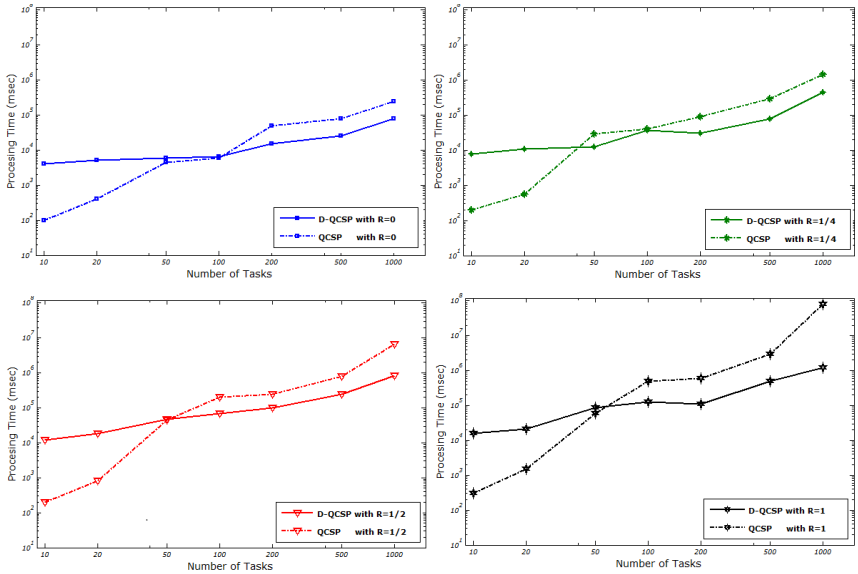


Fig. 36 - The execution performance for each two methods regarding to different overlapping ration (*R*)

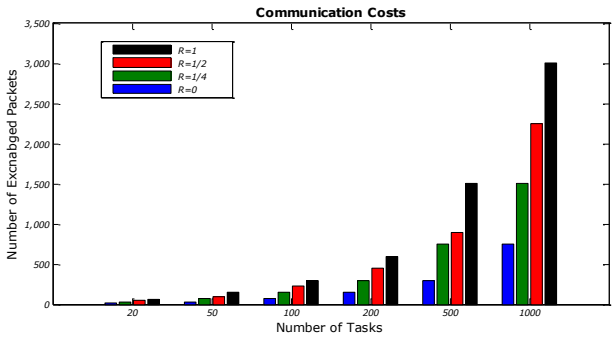


Fig. 38 – Showing the number of exchanged packets among QCAs regarding to various number of tasks and different overlapping ration (*R*)

VII. Conclusion and Discussion

In this paper, we have presented a distributed dynamic scheduling approach for quay cranes at a container terminal based on multi-agent technology. Each agent makes its local decisions and controls its own allocated area, under the control of the port automation supervisory system.

For various kind of scheduling problems (as a subset of combinatorial optimization problems), when the size of problem grows up, conventional centralized programming i.e. traditional dynamic programming, linear programming, branch and bound search and heuristic methods are very hard to find an optimal or near-optimal solution in a reasonable time. For this sake, a model based on parallel distributed processing and working through the cooperative and collective computation upon a MAS structure is tried to get more opportunity to find a near-optimal solution within a reasonable time.

We need to note that, our work has a primary focus on distributed and social aspects of the scheduling tasks cooperatively among agents with a solving procedure to solve the QCSP constantly over time as a dynamic approach. We are specially interested in aspects such as negotiate between peers with an applicable, simple, and consistent algorithm such as ABT algorithm. However this does not mean that the inherent scheduling problem, known as an NP-complete problem [35], does not have enough significance to be considered.

In general, our expectations were confirmed by the experimental results shown in implementation section. However, we have discovered some interesting results which we would like to discuss here.

As we can expect, the efficiency of finding optimal solution decreases when the overlapped area between agents increases particularly. However, when they exchange their partial information, each agent solves its sub-problem in a very low amount of time in comparison with centralized solver. Also, increasing overlapped area between adjacent QCAs will increase communicational cost (total message delivered by agents) dramatically within a certain number of tasks. In this case, we have a trade-off between computational and communicational cost.

We intend to continue our research toward more realistic scenarios. Those scenarios could be in different aspects from the scheduling and control layer to operational layer. We can extend our work with considering real crane constraint such as spatial constraints [19] like non-crossing [7] and non-interference [20] constraints in operational layer. These considerations will affect to problem model and implementations in scheduling layer consequently. Other scenarios can be using different communication algorithms aspect among agents and see results and comparisons their performance and costs with each other.

This preliminary version of the system is currently being implemented, consists of exploiting several fully commercial tools and software such as JAVA, CPLEX, and JADE which fully compliant with FIPA standards. They bring more potential for the MAS prototype to be integrated soon with a port automation system as an integral management of containers which is associated with QCSP.

Commented [Z101]:
RN: In Figure, what if high values of R? (½ and ¾)
Do you see some trends?
What is the difference in performance?
Cost function?
Are the solution obtained the same?

IZ: Added

Commented [Z102]:
RN: Do you find a near-optimal solution? Show this in problem

IZ: Please be note that almost all solution both in QCSP and D-QCSP are "sub-optimal".
In simulation the for more than 50 tasks, the optimal solution will be obtained in a very large time it can take one or two days to reach an optimal solution.
But, I attached two samples of my simulations in the appendix section and you can see after a distinct time, next solutions have not very differences. I assume that time for my simulation results.
Please see the appendix section. You can find the solution (not optimal) time and the elapsed time bellow of each figure

Commented [Z103]:
RN: Show this also for ½ and ¾ and 1

IZ: Done

Commented [Z104]:
RN: give details on how much time individual agents need?

IZ: added

Commented [Z105]:
RN: ?

VIII. ACKNOWLEDGMENT

[…]

IX. REFERENCES

[1] Alphaliner Weekly Newsletter Vol. 2011 Issue 10, website: http://www.alphaliner.com/.

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

[3] I.F.A. Vis and R. de Koster, "Transshipment of containers at a container terminal: an overview",. European Journal of Operational Research 147, 1016 (2003)..

[4] C. F. Daganzo. "The crane scheduling problem". Transportation Research - B, 23:159–175, 1989.

[5] R. Peterkofsky, and C. Daganzo, "A branch and bound soluiton method for the crane scheduling problem". Transportation Research, Part B, vol. 24B, pp. 159-172, 1990..

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

[7] Y. Zhu, and A. Lim, "Crane scheduling with non-crossing constraint," Journal of the Operational Research Society, vol. 57, pp. 1464-1471, 2006..

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

[9] A. Pokahr, et al., "Simulation and Implementation of Logistics Systems based on Agent Technology", in Hamburg International Conference on Logistics 2008: Logistics Networks and Nodes. 2008, Erich Schmidt Verlag. p. 291-308..

[10] N. Neagu, et al., "LS/ATN: Reporting on a Successful Agent-Based Solution for Transport Logistics Optimization", Prague: IEEE 2006 Workshop on Distributed Intelligent Systems (WDIS'06), 2006.

[11] M. Rebollo, et al. "A MAS Approach for Port Container Terminal Management". in Proceedings of the 3rd Iberoamerican workshop on DAI-MAS. Atiaia, Sao Paulo, Brasil, 2001..

[12] P. Davidsson, et al. "Agent-Based Approaches to Transport Logistics". in AAMAS Workshop on Agents in Traffic and Transportation. 2004. New York City..

[13] FIPA: The Foundation for Intelligent Physical Agents, [Online].Available:http://www.fipa.org.,FIPA standards..

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

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

[16] FIPA Contract Net Interaction Protocol Specification. [Online] Available: http://www.fipa.org/specs/fipa00029/.

[17] A. Meisels, "Distributed Search by Constrained Agents (Algorithms, Performance, Communication) - Advanced Information and Knowledge Processing". Springer, 1st Edition., 2008.

[18] H.-O. Gnther, K. H. Kim, "Container Terminals and Automated Transport Systems", Springer, 2004.

[19] A. Lim, B. Rodrigues, F. Xiao, and Y. Zhu, "Crane scheduling with spatial constraints". Naval Research Logistics, vol. 51, pp. 386-406, 2004..

[20] D.-H Lee, H. Wang, and L. Miao, "Quay crane scheduling with non-interference constraints in port container terminals". Transportation Research E, doi: 10.1016/j.tre. 2006.08.001, 2006..

[21] M. Sammarra, J.-F Cordeau, G. Laporte, M. F. Monaco, "A Tabu Search Heuristic for the Quay Crane Scheduling Problem", Journal of Scheduling 10, 327-336, 2006.

[22] L. Moccia, J.-F. Cordeau, M. Gaudioso, and G. Laporte. "A branch-and-cut algorithm for the quay crane scheduling problem in a container terminal". Naval Research Logistics, 53:45–59, 2006..

[23] "Modeling with IBM ILOG CPLEX CP Optimizer – Practical Scheduling Examples", White paper, www.ibm.com.

[24] http://www.ibm.com.

[25] 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.

[26] K. Hirayama and M. Yokoo. Distributed partial constraint satisfaction problem. In G. Smolka, editor, Principles and Practice of Constraint Programming (CP-97), volume 1330 of Lecture Notes in Computer Science, pages 222-236. Springer-Verlag, 1997..

[27] M. Lemaitre and G. Verfaillie, "An incomplete method for solving distributed valued constraint satisfaction problems." In Proceedings of the AAAI Workshop on Constraints and Agents, 1997..

[28] M. Yokoo and E. H. Durfee. Distributed constraint optimization as a formal model of partially adversarial cooperation. Technical Report CSE-TR-101-91, University of Michigan, Ann Arbor, MI 48109, 1991..

[29] P.J. Modi, W.-M. Shen, M. Tambe, and M.Yokoo, "An asynchronous complete method for distributed constraint optimization", In Proceedings of the Second International Joint Conference on Autonomous Agent and Multiagent Systems(AAMAS-03), Australia 2003..

[30] "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".

[31] D. Waltz, Understanding line drawings of scenes with shadows. In Winston, P., editor, The Psychology of Computer Vision, pages 19-91. McGraw-Hill, 1975..

[32] J.M.Vidal, "Fundamentals of Multiagent Systems with NetLogo Examples", Unpublished, 2009 (http://www.multiagent.com).

[33] G. Weiss, "Multiagent systems. A modern approach to Distributed Artificial Intelligence.", Cambridge University, MIT Press, 1999..

[34] "Foundation for Intelligent Physical Agents: FIPA 97 Specfication. Part 2, Agent Communication Language", (1997)..

[35] M. R. Garey, and D. S. Johnson, "Computers and Interactability: A Guide to the Theory of NP-Completeness" Freeman and Co. 1979.

[36] M. Lemaitre and G. Verfaillie. An incomplete method for slving distributed valued constraint satisfaction problems. In Proceedings of the AAAI Workshop on Constraints and Agents, 1997..

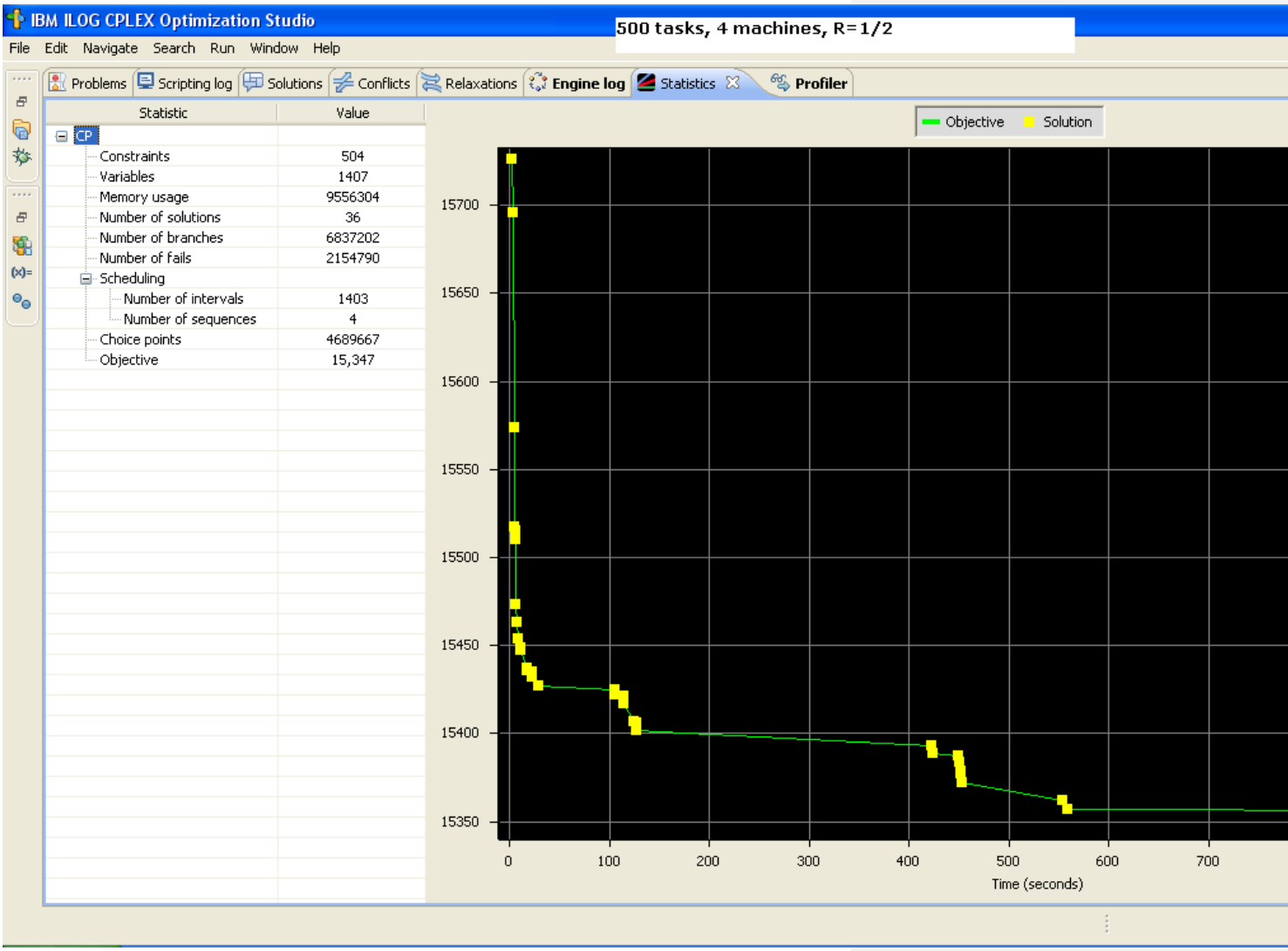
[37] P. J. Modi, W.-M. Shen, M. Tambe, and M. Yokoo. An asynchronous complete method for distributed constraint optimization. In Proceedings of the Second International Joint Conference on Autonomous Agent and Multiagent Systems, Melbourne, Austraalia, 2003.

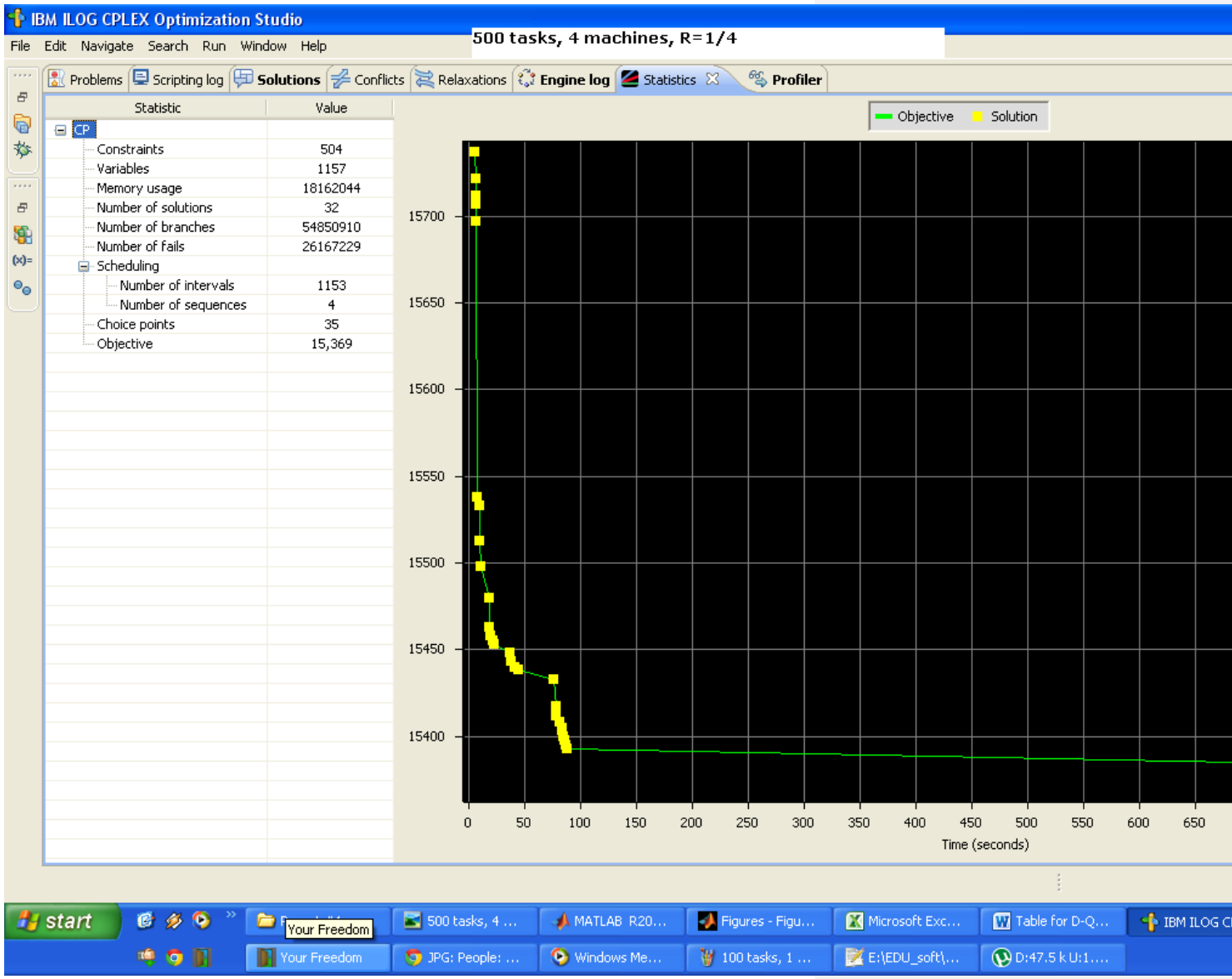
[38] B. Horling, R. Mailler, and V. Lesser. Farm: A Scalable Environment for Multi-Agent Development and Evaluation. Proceedings of the 2nd International Workshop on Software Engineering for Large-Scale Multi-Agent Systems-SELMAS 2003, pages 171-177, May 2003.

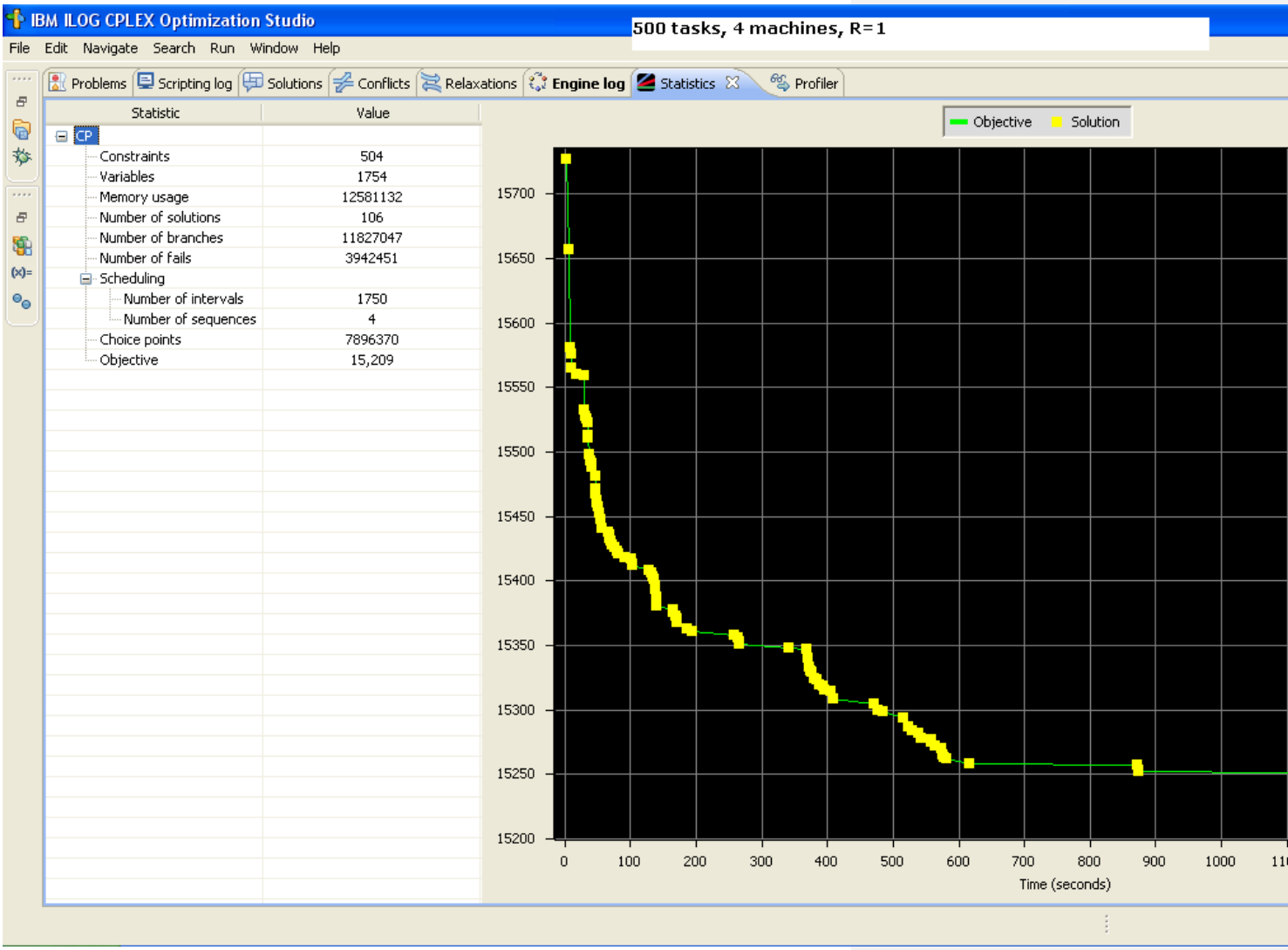
[39] K.Hirayama and M.Yokoo, "Distributed partial constraint satisfaction problem." In G. Smolka, editor, Principles and Practice of Constraint Programming(CP-97), volume 1330 of Lecture Notesin Computer Science, pages 222-236. Springer-Verlag, 1997..

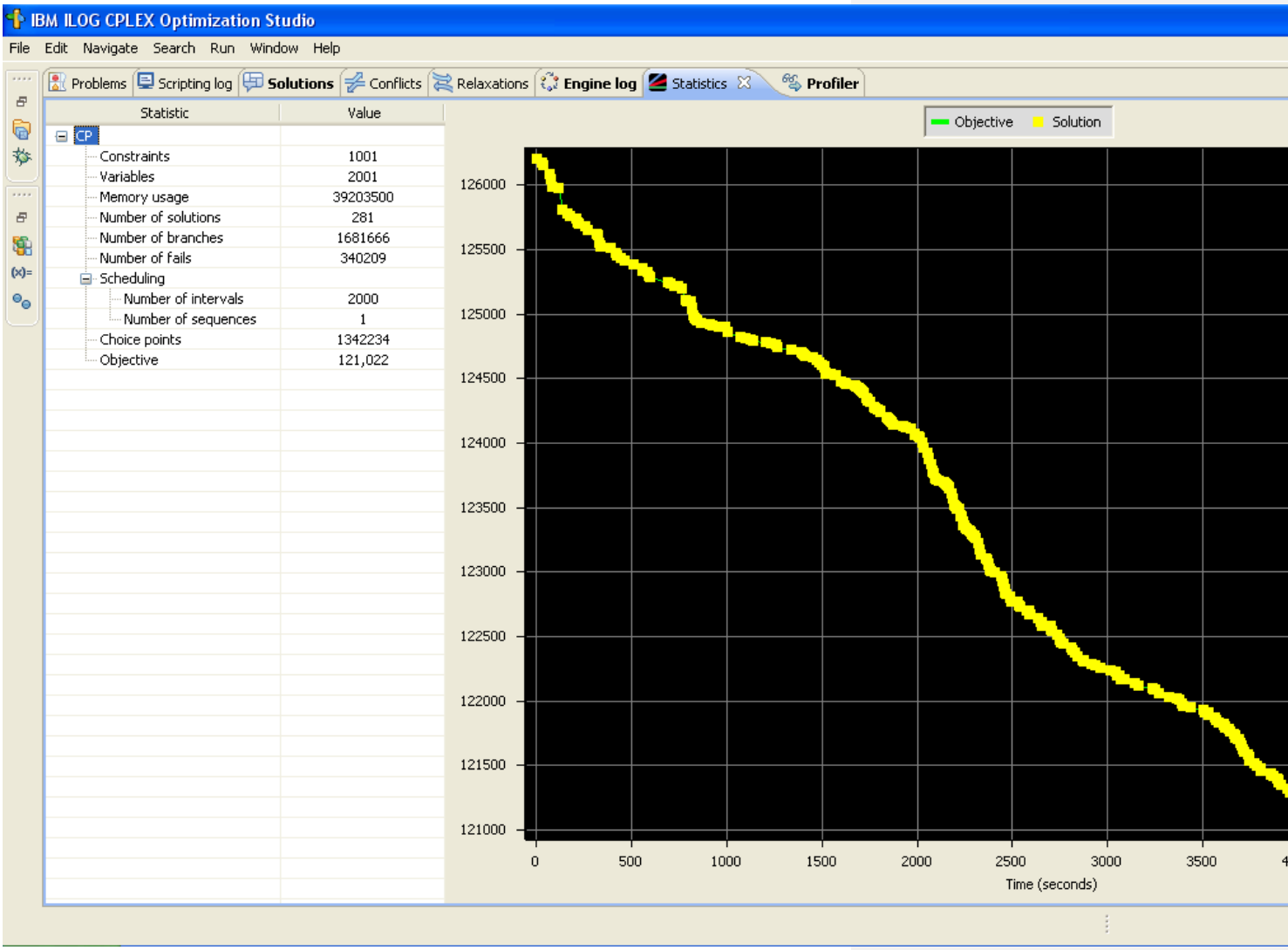
[40] M. Yokoo and E. H. Durfee. Distributed constraint optimiza-tion as a formal model of partially adversarial cooperation. Technical Report CSE-TR-101-91, University of Michigan, Ann Arbor, MI 48109, 1991..

X. APPENDIX


IBM ILOG CPLEX Optimization Studio
500 tasks, 4 machines, R=1/2
File Edit Navigate Search Run Window Help
Problems Scripting log Solutions Conflicts Relaxations Engine log Statistics Profiler
Statistic Value
CP
Constraints 504
Variables 1407
Memory usage 9556304
Number of solutions 36
Number of branches 6837202
Number of fails 2154790
Scheduling
Number of intervals 1403
Number of sequences 4
Choice points 4689667
Objective 15,347
Objective Solution
15700
15650
15600
15550
15500
15450
15400
15350
0 100 200 300 400 500 600 700
Time (seconds)


IBM ILOG CPLEX Optimization Studio
File Edit Navigate Search Run Window Help
500 tasks, 4 machines, R=1/4
Problems Scripting log Solutions Conflicts Relaxations Engine log Statistics Profiler
Statistic
Value
CP
Constraints 504
Variables 1157
Memory usage 18162044
Number of solutions 32
Number of branches 54850910
Number of fails 26167229
Scheduling
Number of intervals 1153
Number of sequences 4
Choice points 35
Objective 15,369
Objective
Solution
15700
15650
15600
15550
15500
15450
15400
0 50 100 150 200 250 300 350 400 450 500 550 600 650
Time (seconds)
start
Your Freedom
500 tasks, 4 ...
MATLAB R20...
Figures - Figu...
Microsoft Exc...
Table for D-Q...
IBM ILOG C
Your Freedom
JPG: People: ...
Windows Me...
100 tasks, 1 ...
E:\EDU_soft\...
D:47.5 k U:1....


IBM ILOG CPLEX Optimization Studio
500 tasks, 4 machines, R=1
File Edit Navigate Search Run Window Help
Problems Scripting log Solutions Conflicts Relaxations Engine log Statistics Profiler
Statistic Value
CP
Constraints 504
Variables 1754
Memory usage 12581132
Number of solutions 106
Number of branches 11827047
Number of fails 3942451
Scheduling
Number of intervals 1750
Number of sequences 4
Choice points 7896370
Objective 15,209
Objective Solution
15700 15650 15600 15550 15500 15450 15400 15350 15300 15250 15200
0 100 200 300 400 500 600 700 800 900 1000
Time (seconds)


IBM ILOG CPLEX Optimization Studio
File Edit Navigate Search Run Window Help
Problems Scripting log Solutions Conflicts Relaxations Engine log Statistics Profiler
Statistic Value
CP
Constraints 1001
Variables 2001
Memory usage 39203500
Number of solutions 281
Number of branches 1681666
Number of fails 340209
Scheduling
Number of intervals 2000
Number of sequences 1
Choice points 1342234
Objective 121,022
Objective Solution
126000
125500
125000
124500
124000
123500
123000
122500
122000
121500
121000
0 500 1000 1500 2000 2500 3000 3500
Time (seconds)