Paper Code: **Indicate your paper code**

A EXACT COMBINATORIAL ALGORITHM FOR BERTH MINIMUM COST ALLOCATION PROBLEM IN CONTAINER PORT

Stevan Kordić[a1], Branislav Dragović[b], Tatjana Davidović[c] and Nataša Kovač[d]

[a] Maritime Faculty, University of Montenegro, Kotor, Montenegro. Email: stevan.kordic@gmail.com
[b] Maritime Faculty, University of Montenegro, Kotor, Montenegro. Email: branod@ac.me
[c] Mathematical Institute, Serbian Academy of Sciences and Arts, Belgrade, Serbia. Email: tanjad@turing.mi.sanu.ac.rs
[b] Maritime Faculty, University of Montenegro, Kotor, Montenegro. Email: knatasa@ac.me

ABSTRACT
This paper presents two exact combinatorial algorithms for solving the *Discrete Berth Allocation Problem (DBAP)* with fixed handling times of vessels classified by *disc* | *stat* | *fix* | $\Sigma(w_1\ wait + w_2\ speed + w_3\ tard + w_4\ pos)$. We address the issues of DBAP according to the well known Park and Kim model. To the best of our knowledge these are the first combinatorial exact algorithms for solving DBAP. Computer results prove the dominance of the proposed algorithms over the classical MIP based exact solvers.

Key Words: Container port, BAP, Combinatorial algorithm, Optimal solution

1. INTROUCTION

Berth Allocation Problem (BAP) is the problem of allocating berths for a set of vessels in that have to be served within time horizon in a container port. Vessels are represented by the set of data regarding expected time of arrival, size, projected time of handling, preferred berth in the port, penalties, etc. Port is presented by the structure of the berths. Beside that there is also a given objective function. The problem is to allocated berth and time for each vessel in the set, such that given objective function is minimized. BAP was proven to be hard NP problem by Lim (1998).

Berth Allocation Problem, according to Meisel (2009) can by classified by four attributes:

1. *Spatial* – concerns the berth layout;
2. *Temporal* – describing temporal constrains on the vessels services;
3. *Handling time* – determining how vessels handling time are considered and
4. *Performance measure* – determines elements of objective function.

The above mentioned attributes determine the type of BAP. In this work we will consider

[1] Corresponding Author's Address: Stevan Kordić, University of Montenegro. Address: Dobrota 36, 85330 Kotor, Montenegro. Email: stevan.kordic@gmail.com

Discrete Berth Allocation Problem (DBAP), which assumes that quay is partitioned into discrete berths. DBAP has a *disc* value of Spatial attribute. Temporal attribute can have values: *stat, dyn* and *due,* which further diversify DBAP. Static case, denoted *stat,* assumes all vessels are in the port when the berth assignment is done or vessels have expected arrival time, but the vessels can be speeded up at a certain cost. Dynamic case, denoted by *dyn,* allows vessel to arrive during the container operations at the port. Finally *due* dates restrict the least allowed departure times of the vessels.

Because of the various values of the attributes there are a lot of different formulations of DBAP and therefore a lot of ways of solving DBAP. Smaller part of them are dedicated to the exact solving of the problem, bigger part are using some heuristic or meta heuristic method to get solutions of DBAP.

Imai *et al.* (2001) first solved static variant of DBAP having vessel handling time dependent on the assigned berth. A Lagrangean relaxation based heuristic is used to solve the problem. Similar approach with stronger Lagrangean relaxation, due to different formulation was applied by Monaco and Samara (2007) for dynamic version of DBAP.

Cordeau *et al.* (2005) model DBAP as Multi-Depot Vehicle Routing Problem with Time Windows and applied Tabu Search meta heuristic for finding solution of the problem. Similar approach was adopted in solving DBAP by Mauri *et al.* (2008). Set partition approach was used in solving DBAP by Cristensen and Holst (2008). Hansen *et al.* (2008) used Variable Neighborhood Search in solving DBAP. Genetic Algorithms where applied in solving different by several variants of DBAP by: Imai *et al.* (2008), Han *et al.* (2006) end Zhou et al. (2006).

In the literature exact methods for solving BAP are rare. Vacca *et al*. (2011) proposed exact algorithm for solving the Tactical Berth Allocation Problem (TBAP) defined by Giallombardo et al. (2010). Exact branch-and-price algorithm produce optimal integer solutions of TBAP.

We propose to approach this problem from combinatorial point of view. Two algorithms for exact solving DBAP are developed. The first one named *Core Sedimentation Algorithm* (CSA) and the second one named *Sedimentation Algorithm with Estimation & Rearrange Heuristic* (SA+ERH). Second algorithm in the preprocessing phase use *Estimation & Rearrange Heuristic* to reduce the search space for CSA. To the best of our knowledge these are the first combinatorial exact algorithms for solving DBAP. Computer results prove the dominance of the proposed algorithms over the classical MIP based exact solvers.

The rest of this paper is organized as follows: Section 2. contains brief description of the considered problem. The proposed algorithms are presented in Section 3. Preliminary computational results are given in Section 4., while Section 5. concludes the paper.

2. BERTH ALLOCATION PROBLEM DESCRIPTION

Proposed algorithm solves desecrate case of BAP (DBAP). Formulation of the problem is a sub model of the Park and Kim (2003) model. We use only part of that model relevant for the berth allocation. It is generally assumed that vessel can occupy only one berth, since in this paper we will consider discrete case of BAP.

2.1 INPUT VARIABLES

Our model, as well as algorithm, is using input data listed bellow:

T: Total number of time periods in the planning horizon.
m : The number of berths in the port.
l : The number of vessels in the planning horizon.
vessel: Sequence of data relevant for vessels with following structure:

$$vessel = \{ (ETA_k, a_k, b_k, d_k, s_k, c_{1k}, c_{2k}, c_{3k}, c_{4k}) \mid k = 1,...,l \}.$$

Elements of vessel 9-tuple represents following data for each vessel:

ETA_k: Expected time of arrival of $vessel_k$;
a_k: The processing time of $vessel_k$;
b_k: The length of $vessel_k$;
d_k: The due time for the departure of $vessel_k$;
s_k: The least-cost berthing location of the reference point of $vessel_k$;
C_{1k}: The penalty cost of $vessel_k$ if the vessel could not dock at its preferred berth;
C_{2k}: The penalty cost of $vessel_k$ per unit time of earlier arrival before ETA_k;
C_{3k}: The penalty cost of $vessel_k$ per unit time of late arrival after ETA_k;
C_{4k}: The penalty cost of $vessel_k$ per unit time delay behind the due time d_k.

As previously mentioned we will consider only desecrate BAP, so the value for the variable b_k will be 1 for all vessels.

2.2 DECISION VARIABLES AND DOMAINS

Park and Kim formulation of BAP uses decision variables. Although combinatorial algorithm do not use them, we will list them:

At_k: The arrival time of $vessel_k$ to the berth, $At_k \in \{1,...,T\}$;
Dt_k: The departing time of $vessel_k$ to the berth, $Dt_k \in \{1,...,T\}$;
X_{itk}: If the berth i at the time t is allocated to $vessel_k$ value is 1, otherwise 0;
$X_{itk} \in \{0,1\}$.

2.3 CONSTRAINS

Every solution of BAP must obey two constrains.

Constrain 1. Each berth at time t can be assigned to only one vessel:

$$(\forall i \in \{1,...,m\})(\forall t \in \{1,...,T\}) \sum_{k=1}^{l} X_{itk} \leq 1.$$

Constrain 2. Berth is allocated for the vessel only between its arrival and departure:

$$(\forall t \in \{1,...,T\})(\forall i \in \{1,...,m\})(\forall k \in \{1,...,l\}) \; (At_k \leq t \leq Dt_k \Longrightarrow X_{itk} = 1) \vee (t < At_k \vee Dt_k < t \Longrightarrow X_{itk} = 0).$$

2.4 OBJECTIVE FUNCTION

Let us first introduce auxiliary variable Z_k:

$$Z_k = \sum_{t=1,\ldots,T} \sum_{i=1,\ldots,m} \text{If}[X_{itk} = 1, |i\text{-}s_k|, 0].$$

Objective function for the minimisation of the port penalty cost can be formulated as follows:

$$VesselsCost = \sum_{k=1,\ldots,l} \{C_{1k}Z_k + C_{2k}(ETA_k - At_k)^+ + C_{2k}(At_k - ETA_k)^+ + C_{4k}(Dt_k - d_k)^+\}$$

From the previous description of the model of BAP we are considering it is clear that it can be classified as:

$$disc \mid stat \mid fix \mid \Sigma(w_1\, wait + w_2\, speed + w_3\, tard + w_4\, pos).$$

The described formulation of the problem is used with CPLEX 11.2 commercial MIP solver for comparison with CSA and SA+ERH proposed here.

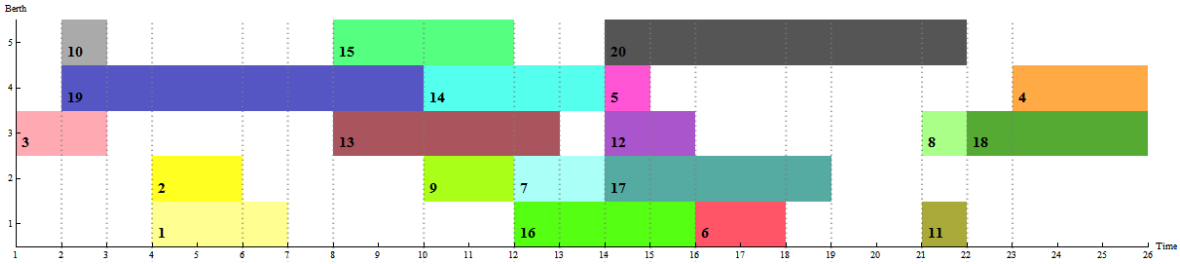

Figure 1. Solved BAP for the 20 vessels in the port with 5 berths and 26 units time horizon

3. SEDIMENTATION ALGORITHM

We introduce the Sedimentation Algorithm for exact solving of discrete BAP which belongs in the class of combinatorial algorithms. It use backtracking mechanism along with a couple of look-a-head techniques for solving discrete BAP developed by authors.

3.1 CORE SEDIMENTATION ALGORITHM

Container port we can be viewed as two dimensional space where horizontal axis represents time and vertical axis enumerate the berths. If we imagine position of the vessel in that berth and time plane as the rectangle, we define two positions are conflicting if they have non empty interior intersection.

CSA is designed to find single optimal solution of DBAP. It can be easily modified to find all of them if more than one exists. As previously mention it search sequentially from vessel to vessel best position for them in order to fine optimal solution. Search through the space of solution is done using backtracking mechanism.

Backtracking mechanism in the algorithm can make ether step forward ($\lambda = 1$) or step backward ($\lambda = -1$). Step forward it makes when it successfully find position for a vessel.

Step backward is made when some of the look-a-head techniques finds that position of a vessel is not good, or when algorithm determine the position of the last vessel and then has to go back to check if there are some more solutions of the problem.

During the work of the algorithm solutions for BAP are generated and algorithm records minimal penalty cost for current best solution in *minCost* variable and the position of the vessels in the α sequence. Whenever new solution is generated its penalty cost is compared with *minCost* and if it is lower we set new *minCost* and α sequence according to the new current best solution. When search through the space of solution is over in α sequence we will have positions of the vessels and in *minCost* penalty cost of the one of optimal solutions. Current best solutions play important part in the work of algorithm, because using them we can apply some of the look-a-head techniques.

By *position* of the vessel we will assume a pair (b,t). The first element of the pair b denotes the lowermost berth assigned to a vessel. The second element of the pair t denotes the time unit in which berth b was assigned to a vessel. We denote by ξ_k function of penalty cost of assigning a $vessel_k$ at the berth b in time unit t.

Let us now explain in detail how Core Sedimentation Algorithm works. Variables α_k, $k=1,...,l$ hold position of the $vessel_k$ in current best solution of the problem. Initially $\alpha_k=(0,0)$ for $k=1,...,l$. Variables P_k, $k=1,...,l$ denotes feasible sub domain of functions ξ_k, $k=1,...,l$. Initially P_k =Dom[ξ_k], for $k=1,...,l$. The value of P_k, $k=1,...,l$ changes during the execution of backtracking mechanisms. Variable *minCost* records of the smallest penalty cost or the best current solution. Initially *minCost* is set to $+\infty$. Variable λ determine the type of current step of the backtracking mechanism. If it is set to 1 a step forward is to be made and if it is set to -1 step backward. Initial value for λ is 1. Counter k denotes the vessel that we are considering and initially its value is set to 1. Beside mentioned variables algorithm uses auxiliary variables: i (as a counter), ε and Q_i, $i=1,...,l$.

CSA also uses procedures and functions. Function Dom[f] computes domain of the function f, function Min[f] returns minimal value of the function f on its domain. Function MinD[f] determines the argument x corresponding to the minimal value of the function f. Procedure ReportSolution[α] examines whether the sequence in the argument represents new current best solution for BAP. In that case the value of variable *minCost* is updated. Function MinVesselsCost[k] is calculated in the following way:

$$\text{MinVesselsCost}[k] = \sum_{i=1,...,k} \xi_i[\alpha_i] + \sum_{i=k+1,...,l} \text{If}[P_i \neq \emptyset,\ \text{Min}[\xi_i|P_i], +\infty].$$

Argument of the function is k and represents the number of vessels for which algorithm has determined positions. Vessels with indices larger than k are still without the position. First sum calculates penalty cost for the vessels which have positions. Second sum is calculated for the vessels still without position. For each $vessel_i$, $i=k+1,...,l$ we sum minimal possible cost of the $vessel_i$, on sub domain P_k i.e. Min[$\xi_k|P_k$] if $P_i \neq \emptyset$, otherwise we sum $+\infty$. Therefore MinVesselsCost[k], will have a finite value if there are available positions for all vessels and infinite if at least one $vessel_i$, $k+1 \leq i \leq l$ have no available positions.

Function ConflictingPositions[P_i, k, p] returns all the positions in the P_i conflicting with the position p of the $vessel_k$. The inspiration for the name of the algorithm comes from the way we select position for a vessel. Selection of the least-cost position, which is the minimum of the function ξ_k on sub domain P_k, resembles the natural phenomena of sedimentation of particles in fluids.

The pseudo code of Core Sedimentation Algorithm is this:

```
[ 1]  CoreSedimentAlgorithm[m,T,l,vessel]
[ 2]      for k=1 to l do α_k = (0,0);
```

```
[ 3]    for k=1 to l do P_k = Dom[ξ_k];
[ 4]    minCost = +∞;
[ 5]    λ = 1;
[ 6]    k = 1;
[ 7]    while 0 < k do
[ 8]        if λ = –1 then
[ 9]            P_k = Dom[ξ_k];
[10]            P_k = P_k \ ∪_{i=1,...,k-1} ConflictingPositions[P_k, i, α_i];
[11]            k--;
[12]            λ = 1;
[13]        else
[14]            if P_k = ∅ then
[15]                λ = –1;
[16]            else
[17]                α_k = MinP[ξ_k|P_k];
[18]                P_k = P_k \ {α_k};
[19]                if k = l then
[20]                    ReportSolution[α];
[21]                    λ = –1;
[22]                else
[23]                    if MinVesselsCost[k] < minCost then
[24]                        for i=k+1 to l do Q_i = P_i ;
[25]                        for i=k+1 to l do P_i = P_i \ ConflictingPositions[P_i, k, α_k];
[26]                        ε = minCost – MinVesselsCost[k];
[27]                        if 0 < ε then
[28]                            if minCost < +∞ then
[29]                                for i=k+1 to l do P_i = { p ∈ P_i | ξ_i[p] < Min[ξ_k|P_k] + ε};
[30]                            endif;
[31]                            k++;
[32]                        else
[33]                            for i=k+1 to l do P_i = Q_i ;
[34]                        endif;
[35]                    else
[36]                        λ = –1;
[37]                    endif;
[38]                endif;
[39]            endif;
[40]        endif;
[41]    endwhile;
[42]    return[α];
[43]  end.
```

Lines: [1]–[6] initialise variables of the algorithm. Backtracking mechanism is coded in lines: [7]–[41]. Counter k, representing vessel index, is at the beginning set to 1, during the work of the algorithm it will rise up to the maximum number of vessels i.e. l. On the end, after all possible positions for all vessels have been considered, the value of k will be 0, defining condition to exit backtracking.

Backtracking can take ether step forward or step backward. Step backward is performed if $\lambda = -1$, lines: [9]–[12], in which algorithm first sets P_k to be the Dom[ξ_k] and

then removes all positions conflicting with the previous vessels. Counter k is decreased by 1 and finally λ set for the backtracking step forward i.e. $\lambda = -1$.

Step forward, lines [14]–[40], begins with examining if $P_k = \emptyset$. If it is fulfilled then algorithm has examined all possible positions for the vessel k, therefore backtracking step backward is needed i.e. $\lambda = -1$, line [15]. If the condition is not fulfilled, then there are still some positions for vessel k to be examined, lines [17]–[39]. Then, minimal penalty cost i.e. $\alpha_k = \text{MinP}[\xi_k|P_k]$ is considered as the next position for the vessel k. If the last vessel has been reached i.e. $k = l$, then ReportSolution[α] is called and backtracking mechanism will be set for the next backward step, lines [20]–[21]. There is no need to examine other positions of the last vessel, because all of them lead to the higher value of objective function.

If $k \neq l$ then we can apply first look-a-head technique. Function MinVesselsCost[k] will return value of penalty cost for the vessels with indices less than or equal to k plus minimal possible penalty cost for the vessels from $k+1$ to l. If this value is equal to or higher then *minCost,* than position α_k can not improve current best solution. Moreover, any position $p \in P_k \setminus \{\alpha_k\}$, which is still to be considered, will have a cost equal to or higher than α_k. Conseqently following ineqality holds:

$$\begin{aligned}
&\text{MinVesselsCost}[k] = \\
&\quad = \textstyle\sum_{i=1,\ldots,k} \xi_i[\alpha_i] + \sum_{i=k+1,\ldots,l} \text{If}[P_i \neq \emptyset,\ \text{Min}[\xi_i|P_i], +\infty] \\
&\quad = \textstyle\sum_{i=1,\ldots,k-1} \xi_i[\alpha_i] + \xi_k[\alpha_k] + \sum_{i=k+1,\ldots,l} \text{If}[P_i \neq \emptyset,\ \text{Min}[\xi_i|P_i], +\infty] \\
&\quad \leq \textstyle\sum_{i=1,\ldots,k-1} \xi_i[\alpha_i] + \xi_k[p] + \sum_{i=k+1,\ldots,l} \text{If}[P_i \neq \emptyset,\ \text{Min}[\xi_i|P_i], +\infty], \text{ for } p \in P_k \setminus \{\alpha_k\}.
\end{aligned}$$

Previous inequality means that none of the remaining positions for the vessel k can lead to the improvement of the current best solution. Therefore if the test in line [23] fails, backtracking step backward, line [36] is performed. Otherwise, CSA can proceed finding better solution.

In that case algorithm puts the P_i, $i=k+1,\ldots,l$ sub domains in auxiliary variables Q_i, $i=k+1,\ldots,l$, line [24]. Then all the conflicting positions in P_i, $i=k+1,\ldots,l$ with position α_k of the $vessel_k$ are removed, line [26]. Some of the P_i, $i=k+1,\ldots,l$ sub domains may be changed by the deletion of the conflicting positions, so we can apply second look-a-head technique by testing again MinVesselsCost[k] < *minCost.* If the test fails then algorithm can move only to the another position of vessel k, but can not perform backtracking step backward as it was the case with the first look-a-head technique. In the first look-a-head no deletion was done in the P_i, $i=k+1,\ldots,l$ sub domains, so all the conclusions about relationship of MinVesselsCost[k] and *minCost* where general. That is not the case with the second look-a-head, because some positions in P_i, $i=k+1,\ldots,l$ sub domains may be deleted, which means that in this case conclusions about relationship of MinVesselsCost[k] and *minCost* are valid only for the vessel k and position α_k. The above mention second look-a-head technique is coded in lines [26] and [27]. In [27] ε is calculated as $\varepsilon = minCost - \text{MinVesselsCost}[k]$. Note that there is possibility to get indeterminable expression $\infty - \infty$. In that case we assume that $\varepsilon = 0$. The comparison MinVesselsCost[k] < *minCost* is equivalent to $0 < \varepsilon$. If $0 < \varepsilon$ fails algorithm restores P_i, $i=k+1,\ldots,l$ sub domains sequences from auxiliary variables Q_i, $i=k+1,\ldots,l$, line [33], and keep the value of the vessel index k unchanged.

If $0 < \varepsilon$ is true, then CSA can proceed finding better solution by applying third look-a-head technique. This technique can be applied only if there is at least one solution of DBAP i.e. $\varepsilon < +\infty$, If that is the case then also $0 < \varepsilon < +\infty$ holds.

Proposition 1. If $0 < \varepsilon < +\infty$ and positions for the first k vessels are the same, then following holds:

$$\text{MinVesselsCost}[l] < \text{MinVesselsCost}[k] \iff (\forall i \in \{k+1,...,l\})\, \xi_i|P'_i[\alpha_i] < \text{Min}[\xi_k|P_k] + \varepsilon,$$

where P_i', $i=k+1,...,l$ represent sub domains in the algorithm at the moment when the position of the last vessel $vessel_l$ has been found.

Proof. The ($\Longleftarrow$) direction of the equivalence is trivial, the ($\Longrightarrow$) other is not. Let us prove it by *reductio ad absurdum*. Suppose that $\text{MinVesselsCost}[l] < \text{MinVesselsCost}[k]$ holds and that $(\exists i \in \{k+1,...,l\})\, \xi_i|P'_i[\alpha_i] \geq \text{Min}[\xi_k|P_k] + \varepsilon$. For convenience let us suppose that there is a $s \in \{k+1,...,l\}$, such that $\xi_s|P'_s[\alpha_s] \geq \text{Min}[\xi_s|P_s] + \varepsilon$. As we have previously seen $0 < \varepsilon$, which means that $\text{MinVesselsCost}[k] < minCost$. Also from the construction of the algorithm we can easily conclude that $(\forall i \in \{k+1,...,l\})\ P'_i \subseteq P_i$, from which we easily derive that $(\forall i \in \{k+1,...,l\})\ \text{Min}[\xi_i|P_i] \leq \text{Min}[\xi_i|P'_i]$. Then the following sequence of equalities and inequalities holds:

$$\begin{aligned}
&\text{MinVesselsCost}[k] > \\
&\quad > \text{MinVesselsCost}[l] \\
&\quad = \textstyle\sum_{i=1,...,k} \xi_i[\alpha_i] + \sum_{i=k+1,...,l} \xi_i|P'_i[\alpha_i] \\
&\quad = \textstyle\sum_{i=1,...,k} \xi_i[\alpha_i] + \sum_{i=k+1,...,s-1} \xi_i|P'_i[\alpha_i] + \xi_s|P'_s[\alpha_s] + \sum_{i=s+1,...,l} \xi_i|P'_i[\alpha_i] \\
&\quad \geq \textstyle\sum_{i=1,...,k} \xi_i[\alpha_i] + \sum_{i=k+1,...,s-1} \text{Min}[\xi_i|P'_i] + \text{Min}[\xi_s|P_s] + \varepsilon + \sum_{i=s+1,...,l} \text{Min}[\xi_i|P'_i] \\
&\quad \geq \textstyle\sum_{i=1,...,k} \xi_i[\alpha_i] + \sum_{i=k+1,...,s-1} \text{Min}[\xi_i|P_i] + \text{Min}[\xi_s|P_s] + \varepsilon + \sum_{i=s+1,...,l} \text{Min}[\xi_i|P_i] \\
&\quad = \textstyle\sum_{i=1,...,k} \xi_i[\alpha_i] + \sum_{i=k+1,...,l} \text{Min}[\xi_i|P_i] + \varepsilon \\
&\quad = \text{MinVesselsCost}[k] + \varepsilon \\
&\quad = \text{MinVesselsCost}[k] + minCost - \text{MinVesselsCost}[k] = minCost.
\end{aligned}$$

If we connect the beginning and the end of the above formula we get $\text{MinVesselsCost}[k] > minCost$. And that is the contradiction with the assumption that $0 < \varepsilon$ i.e. $\text{MinVesselsCost}[k] < minCost$. Which concludes the proof for the equivalence. QED.

Consequence 1. If we have $vessel_k$ such that $\text{MinVesselsCost}[k] < minCost$, then better solution with lover penalty cost than *minCost* exist if and only if penalty costs for the remaining vessels: $vessel_j$, $j=k+1,...,l$ are lower than $\text{Min}[\xi_i|P_i] + \varepsilon$, $i=k+1,...,l$, respectively.

The **Consequence 1.** justify the third look-a-head technique in lines [28]–[30].

Regardless the third look-a-head technique was applied or not CSA conclude backtracking step forward by increasing counter k by one, line [31], and keeping the value $\lambda = 1$. Thus a backtracking step forward for the next vessel is to be performed.

3.3 SEDIMENTATION ALGORITHM WITH ESTIMATION & REARRANGE HEURISTIC

CSA efficiency heavily depends on the order of the vessels. The best ordering for the algorithm is when we sort vessels in the descending order of their penalty costs in optimal solution. The problem is that we can not know this order in advance. Instead, we can allow CSA to run on the couple of different orderings for a limited time. Then we order vessels according to their penalty cost in the best obtain solution. Finally, we start CSA from the new ordering until it stops. This simple heuristic, leads to the significant improvement of the efficiency of CSA. We name this heuristic *Estimation & Rearrange Heuristic* (ERH). Algorithm we get when ERH is applied, and than CSA we name *Sedimentation Algorithm*

with Estimation & Rearrange Heuristic (SA+ERH).

Let us now describe SA+ERH in more detail. By straight forward modification of the described CSA we can make it start from any predefined ordering of the vessels ω. Variable ω is a permutation of set of vessels indices $\{1, 2,..., l\}$. Instead of accessing k-th vessel in the algorithm we access vessel $\omega(k)$. CSA modified this way we call as a CoreSedimentAlgorithm$\Omega[m,T,l,vessel_k,\omega]$ function. Same as the described algorithm function CoreSedimentAlgorithm$[m,T,l,vessel_k]$, this new function also return sequence α containing the positions of the vessels in optimal solution of DBAP. Because of the several executions of the function, variables *minCost* and α are treated as the global ones.

We assume to have a function RunForLimitedTime[*function,time*] which causes *function* stopping after *time* seconds of execution. It returns a pair (*result, finished*), where *result* represents the result of the *function* and *finished* is true if the function was completed during the *time* seconds of execution, otherwise it is false. Moreover we assume the availability of the function RandomPermutation[ω] which returns a random permutation of the sequence ω. Variable *CSArestarts* determines number of estimations and variable *EstT* determines time duration in seconds of the one estimation. Both of them are treated as the global ones.

SA+ERH can be represented by the following pseudocode:

```
[ 1]  EstimationRearrangeHeuristic[m,T,l,vessel]
[ 2]     minCost = +∞;
[ 3]     for k=1 to l do α_k = (0,0);
[ 4]     ω = {1, 2, … , l};
[ 5]     for i=1 to CSArestarts do
[ 6]          (α,completed) =
[ 7]               RunForLimitedTime[CoreSedimentAlgorithmΩ[m,T,l,vessel_k,ω],EstT];
[ 8]          if completed then return[α];
[ 9]          ω = RandomPermutation[ω];
[10]     endfor;
[11]     if minCost < +∞ then
[12]        for i=1 to l–1 do
[13]             for j=i+1 to l do
[14]                if  ξ_ω[i][α_ω[i]] < ξ_ω[j][α_ω[j]] then
[15]                   k = ω[i];
[16]                   ω[i] = ω[j];
[17]                   ω[j] = k;
[18]                endif;
[19]             endfor;
[20]        endfor;
[21]     else
[22]        ω = {1, 2, … , l};
[23]     endif;
[24]     α = CoreSedimentAlgorithmΩ[m,T,l,vessel_k,ω];
[25]     return[α];
[26]  end.
```

Lines: [1]–[4] initialize variables in the algorithm. In the lines [5]–[10] algorithm makes estimations. If CSA reaches for some feasible solution it will be saved in global variable α. Values of the variable *minCost* will also be changed accordingly. If estimations found at least one feasible solution the value of *minCost* will be less than $+\infty$, line [11].

Then we can sort vessels, by rearranging ω, according to their penalty costs in the minimal solution α, lines [12]–[20]. Otherwise we use the most simple ω, line [22]. Finally, the algorithm runs CSA from ω ordering of the vessels and, after the end of execution, returns optimal solution of DBAP.

4. COMPUTATIONAL RESULTS

In this section we give the computational results of the CSA and SA+ERH. Moreover, we select one example and compare running times of both Sedimentation algorithms with CPLEX commercial solver.

4.1 COMPARISON BETWEEN CORE SEDIMENTATION AND SEDIMENTATION WITH ESTIMATION & REARRANGE HEURISTIC ALGORITHMS

CSA and SA+ERH have been coded in *Wolfram Mathematica v8.0* programing language. Wolfram Mathematica v8.0 interprets instructions and, although it is very convenient for algorithm design, it can not be considerd as a fast in the terms of execution. The test were conducted on computer with *Intel Core i7 CPU Q720 @ 1.60GHz* processor with 6 GB of RAM, running on *Microsoft Windows 7 64-bit* operating system type.

We made two classes of test instances. In the first one time horizon is one week and in the second one time horizon is two weeks. Time horizon is divided by 3 hour time unit. Thus, one week has 56 time units and two weeks are divided into 112 time units. Classes of test instances are the following:

Class I: 3 berths, 1 week;
Class II: 5 berths, 2 week;

Similar classes of instances you can find in Giallombardo *et al.* (2010). The tests were conducted on the range of vessels, starting from 5 up to 40 with an increment of 5. Size, handling time distribution in time units and penalties in 1000 USD units of the vessels are given in Table 1. Specifications resembles on those in Meisel (2009), adjusted here to DBAP.

Table 1. Test vessels specifications

Size, handling times and penalties for test vessels						
Vessel type	*Percent in test population*	*Handling time range*	C_1	C_2	C_3	C_4
Feeder	*60%*	*1 – 3*	*2*	*3*	*3*	*9*
Medium	*30%*	*4 – 5*	*3*	*6*	*6*	*18*
Mega	*10%*	*6 – 8*	*4*	*9*	*9*	*27*

Distribution of the least-cost berthing location for vessels is homogeneous. For each instance and number of vessels, a set of 300 test were generated randomly. We where recording percentage of the tests solved in a half a hour period i.e. 1800 seconds. For the tests solved we recorded the minimal, average and maximal problem solving time. All the times are expressed in seconds.

Table 2. Computational results for the Class I test instances

l Number of vessels	CLASS I: 3×56 | 300 samples									
	Core Sedimentation Algorithm					Sedimentation Algorithm with Estimation & Rearrange Heuristic				
	$t \le ½h$	min	aver	max	$½h < t$	$t \le ½h$	min	aver	max	$½h < t$
5	*100.0%*	0.11	**0.15**	0.25	*0.00%*	*100.0%*	0.11	**0.15**	0.28	*0.0%*
10	*100.0%*	0.22	**6.15**	571.48	*0.0%*	*100.0%*	0.23	**0.44**	3.68	*0.0%*
15	*83.3%*	0.38	**83.99**	1584.32	*16.7%*	*100.0%*	0.37	**2.07**	80.93	*0.0%*
20	–	–	**–**	–	–	*97.7%*	0.58	**43.11**	1791.13	*2.3%*

Table 2. shows computational result for the Class I test instances. It is evident from this table that in the trivial case of 5 vessels both algorithms show equal performance. In the case of 10 and more vessels SA+ERH outperforms CSA. In the case of 10 vessels it is almost 13 times faster and for 15 vessels it is 37.51 times faster.

Table 3. Computational results for the Class II test instances

l Number of vessels	CLASS II: 5×112 | 300 samples									
	Core Sedimentation Algorithm					Sedimentation Algorithm with Estimation & Rearrange Heuristic				
	$t \le ½h$	min	aver	max	$½h < t$	$t \le ½h$	min	aver	max	$½h < t$
5	*100.0%*	0.39	**0.47**	0.66	*0.0%*	*100.0%*	0.39	**0.50**	0.72	*0.0%*
10	*100.0%*	0.83	**0.95**	1.86	*0.0%*	*100.0%*	0.83	**1.03**	1.48	*0.0%*
15	*99.3%*	1.30	**7.35**	620.07	*0.7%*	*100.0%*	1.33	**1.69**	5.84	*0.0%*
20	*93.7%*	63.23	**63.23**	1731.7	*633.3%*	*100.0%*	1.83	**2.82**	7.22	*0.0%*
25						*100.0%*	2.40	**5.10**	21.95	*0.0%*
30						*99.3%*	3.00	**8.40**	29.03	*0.7%*
35						*97.0%*	3.57	**33.15**	1699.13	*3.0%*
40						*95.7%*	5.26	**40.17**	1735.26	*4.3%*

Table 3. shows computational result for the Class II test instances. It is evident from this table that in the trivial cases of 5 and 10 vessels both algorithms show equal performance. In the case of 15 and more vessels SA+ERH outperforms CSA. In the case of 15 vessels it is 4.35 times faster, for 20 vessels it is 22.42 times faster. Times of CSA execution for cases of vessels number bigger than 20 are not presented in the Table 2. because of a long time needed for solving 300 examples. However, note than on the average SA+ERH time for solving the examples with 40 vessels is lower than the average time of the CSA for 20 vessels.

4.2 COMPARISON BETWEEN SEDIMENTATION ALGORITHM AND CPLEX

For this comparison we select one representative example to compare with CPLEX commercial solver. The version of CPLEX 11.2 running on Intel Core 2 DUO CPU E6750

on 2.66GHz, operated by Linux Slackware 12, Kernel: 2.6.21.5 on the computer with 8 GB of RAM was used for comparison. The first example is from Class I and the second is from Class III. Input data for the Example are given in Table 4.

Table 4. Input data for Example

k Vessel number	EXAMPLE: 5×112 | CLASS II								
	ETA_k	a_k	b_k	d_k	s_k	C_{1k}	C_{2k}	C_{3k}	C_{4k}
1	102	2	1	105	1	2	3	3	9
2	14	2	1	17	2	2	3	3	9
3	72	3	1	76	3	2	3	3	9
4	76	3	1	80	4	2	3	3	9
5	61	1	1	63	5	2	3	3	9
6	7	3	1	10	1	2	3	3	9
7	29	2	1	31	2	2	3	3	9
8	81	1	1	83	3	2	3	3	9
9	30	1	1	32	4	2	3	3	9
10	21	1	1	22	5	2	3	3	9
11	76	1	1	78	1	2	3	3	9
12	61	1	1	62	2	2	3	3	9
13	30	3	1	33	3	2	3	3	9
14	65	2	1	67	4	2	3	3	9
15	84	1	1	85	5	2	3	3	9
16	79	2	1	82	1	2	3	3	9
17	45	3	1	48	2	2	3	3	9
18	35	2	1	37	3	2	3	3	9
19	16	5	1	22	4	3	6	6	18
20	30	4	1	34	5	3	6	6	18
21	15	4	1	19	1	3	6	6	18
22	47	4	1	52	2	3	6	6	18
23	34	5	1	39	3	3	6	6	18
24	28	4	1	33	4	3	6	6	18
25	2	5	1	8	5	3	6	6	18
26	14	4	1	19	1	3	6	6	18
27	32	4	1	37	2	3	6	6	18
28	103	7	1	110	3	4	9	9	27
29	67	6	1	73	4	4	9	9	27
30	11	6	1	18	5	4	9	9	27

Out of above Example we solved cases with 20, 25 and 30 vessels by CPLEX, CSA and SA+ERH. In the last two columns we give ratio between *CSA* and *CS+ERH* solving times to CPLEX and solving time.

Table 7. Results for Example I

k Vessel number	EXAMPLE solving times: 3×56 | CLASS III					
	Optimum	CPLEX	CSA	SA+ERH	$\times_{CSA}$	$\times_{SA+ERH}$

20	*0*	200.552	2.215	2.200	**90.54**	**91.16**
25	*12*	10840.700	27.300	5.148	**397.09**	**2105.81**
30	*27*	21803.600		9.391		**2321.75**

Table 7. shows computational results for Example. Both CSA and SA+ERH outperforms CPLEX in a very persuasive way. For the case of l=25 vessels CSA is 397 times faster and SA+ERH is 2105 times faster. This proves straight of both dedicated combinatorial algorithm over a general MIP solver such as CPLEX. Relatively slow solving time of CSA in the case of l=30 is caused by the fact that vessels with indices from 26 to 30 are very big (mega type) with high penalty costs. Putting this type of vessel at the end of processed vessels significantly slows down CSA performance. Because of ERH this situation is avoided in SA+ERH.

5. CONCLUSION

We considered the discrete minimal cost Berth Allocation Problem (DBAP) with the static arrival of vessels and fixed vessels time handling. Performed computational experiments fully justify the design and further development of the *Sedimentation Algorithm* for exact solving of DBAP. When it is combined with even simple heuristics like *Estimation & Rearrange Heuristic* it can be used for instances with large number of vessels, as stand alone method for solving of DBAP, or as a part some more complex heuristic or meta-heuristic approach for solving DBAP. Results also indicate that this method can certainly be used for solving real-life medium sized problems of BAP.

Difference between minimal and maximal solving times for the large number of vessels in test instances indicates that *Sedimentation Algorithm* is worth of further development. We find specially worth investigating possibility of partial solving of DBAP for the conflicting vessels and combining it with other vessels to get optimal solution of the entire problem. Moreover, the expansion of the algorithm on hybrid and continuous BAP, as well as adding crane assignment to *Sedimentation Algorithm* are natural extensions for the further development of the described algorithms.

REFERENCES

Christensen, C.G, Holst, C.T., 2008. Berth allocation in container terminals. Master's thesis, Department of Informatics and Mathematical Modelling, Technical University of Denmark.

Cordeau, J.F., Laporte, G., Legato, P., Moccia, L, 2005. Models and tabu search heuristics for the bearth-allocation problem. Transportation Science, 39(4), 526-538.

Giallombardo, G., Moccia L., Salani, M., Vacca, I., 2010. Modeling and solving the tactical berth allocation problem. Transportation Research Part B, 44(3), 400-415.

Han, M., Li, P., Sun, J., 2006. The algorithm for berth scheduling problem by the hybrid optimization strategy GASA. Proceedings of the 9th International Conference of Control, Automation, Robotics and Vision (ICARCV '06), IEEE Computer Society, Washington, 1-4.

Hansen, P., Oğuz, C., Mladenović, N., 2008. Variable neighborhood search for minimum cost berth allocation. European Journal of Operational Research, 191(3), 636-649.

Vacca, I., Salani, M., Bierlaire M., 2011. An exact algorithm for integrated planning of

berth allocation and quay crane assignment, Report TRANSP-OR 110323.
Imai, A., Nishimura, E., Papadimitriou, S., 2001. The dynamic berth allocation problem for container port. Transportation Research Part B, 35(4), 401-417.
Imai, A., Nishimura, E., Papadimitriou, S., 2008. Berthing ships at a multi-user container terminalwith a limited quay capacity. Transportation Research Part E, 44(1), 136-151.
Lim, A., 1998. The berthing planning problem. Operational Research Letters, 22(2), 105-110.
Maisel, F., Seaside Operations Planning in Container Terminals, Physica Verlag, Berlin et al., 2009.
Mauri, G.R., Oliveira A.C.M., Lorena, A.N., 2008. A hybrid column generation approach for the berth allocation problem. EvoCOP 2008, Lecture Notes in Computer Science, volume 4972, 110-122.
Monaco, F.M., Samara, M., 2007. The berth allocation problem: a strong formulation solved by a lagrangian approach. Transportation Science, 41(2), 256-280.
Park, Y.M., Kim, K.H., 2003. A scheduling method for the berth and quay cranes. OR Spectrum, 25(1), 1-23.
Zhou, P., Kang, H., Lin, L., 2006. A dynamic berth allocation model based on stochastic consideration. Proceedings of the 6th World Congress on Intelligent Control and Automation (WCICA), IEEE Computer Society, Washington DC, vol 2, 7297-7301.