UNLOADING AND PREMARSHALLING ALGORITHMS WITH JAVA COMPUTER ANIMATION FOR TERMINAL YARD OPERATIONS

Ivan Makohon
M.S. Student*
Modeling, Simulation & Visualization Engineering (MSVE) Department
Phone: 757-481-0832
imako001@odu.edu

Mecit Cetin (corresponding author)
Associate Professor*
Civil & Environmental Engineering (CEE) Department
Phone: 757-683-6700
mcetin@odu.edu

Manwo Ng
Assistant Professor*
Department of Information Technology and Decision Sciences
Phone: 757-683-6665
mng@odu.edu

Duc T. Nguyen
Professor*
CEE and MSVE Departments
Phone: 757-683-3761
dnguyen@odu.edu

*Old Dominion University
Norfolk, Virginia 23529

Submitted for **Presentation** at the 95th Annual Meeting (January 10-14, 2016; Washington, D.C.) of the Transportation Research Board (TRB)
Primary Committee: Education and Training (9)
Secondary Committee: Terminals and Facilities (32) and Marine Transportation (18)
Submitted: July 31, 2015
Total words = 5,469 + 250*8 (7 Figures and 1 Table) = 7,469

ABSTRACT

Unloading and pre-marshalling (or reshuffling) operations have been considered as basic/important tasks in port terminals' activities, since the economic impacts can be significant. The primary goal for this paper is to develop simple heuristic unloading and pre-marshalling algorithms. Detailed explanations for the proposed algorithms are given in the form of step-by-step numerical procedures, so that the reported results can be reproduced by the readers. Several numerical examples (including the ones considered by other researchers) are used to validate the proposed unloading and pre-marshalling algorithms.

A secondary goal for this paper is to develop a user-friendly and easy-to-use Java Graphical User Interface (GUI) software application that provides precise and clear step-by-step instructions with visual animation and voice for the unloading and pre-marshalling algorithms for Terminal Container Yard operations. The developed Java software can be used as an additional/educational tool that would serve both academia students and transportation professionals in solving any complex dynamic Terminal Container Yard layout visually with provided steps to show the "final/optimal" results. **A demo video of the both the unloading and pre-marshalling algorithm's animation and result can be viewed online from any web browser using the website provided in reference [*4*].**

Keywords: Terminal, Yard, Cargo, Container, Unloading, Pre-marshalling, Reshuffling, Algorithm, Java, Animation

1. INTRODUCTION

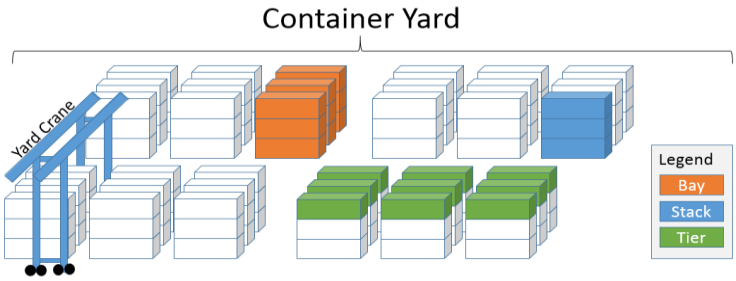


Figure 1 Container Yard (Storage Yard)

Container Terminal Yard operations [see Figures 1-2] reveal an increasing traffic of cargo container shipment around world ports [*1*], which makes planning decision critical with great importance to optimizing the terminal container yard's unloading and pre-marshalling (or reshuffling) of cargo containers within the yard. Container Terminal Yards consists of container bays, where each bay contains a set of container stacks, and each container stack contains stackable cargo containers. It's an important storage point because it links the seaside with the landside. More precisely, it's a temporary storage point for loading and unloading cargo containers between vessels and vehicles (trucks or trains) for further distribution. The yard operations department is responsible for allocating space and equipment needed to maintain terminal efficiency.

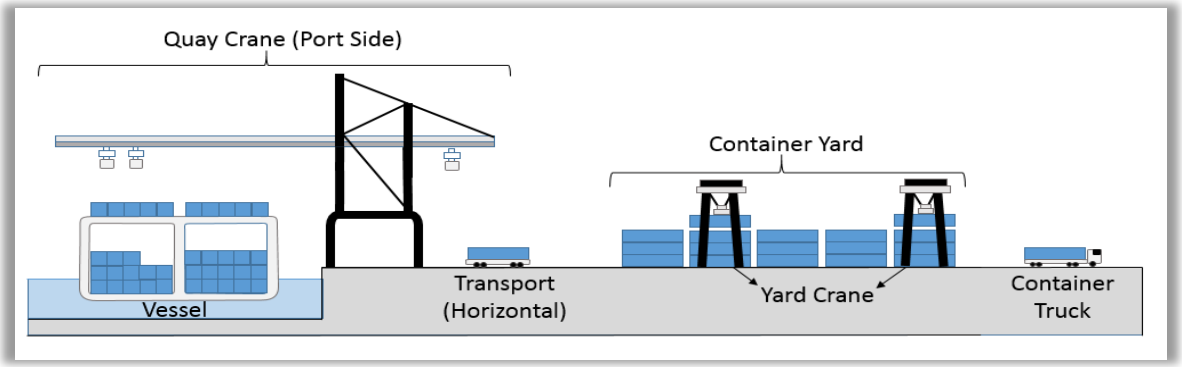


Figure 2 Container Terminal

In this paper, the unloading and pre-marshalling algorithms for container terminal yard operations is revisited with the primary goals of developing simple heuristic "unloading" and "pre-marshalling" algorithms, for port operations. For safety reasons, moving containers from one bay to another bay by crane is usually prohibited. The proper movement involves putting the container on a truck first, moving the truck from one bay to another, and then moving the container into the desired bay [*2*]. Therefore, these algorithms perform efficient unloading and pre-marshalling tasks in container terminals.

This container yard stacking problem is very similar to the Blocks World planning domain problem which consists of a finite number of blocks stacked into columns. The goal is to turn the initial state into a goal state by moving one block at a time from the top of one column onto another column. The Block World planning problem is to get to the goal state in the minimal number of moves [*3*]. Since efficient operations during the "unloading" and "pre-marshalling" phases will save time in port operations which will lead to significant economic gains, large amount of existing literatures have been devoted in these needed algorithms [*1-4, 5, 6-11*]. Researchers have also surveyed and synthesized various available methods and algorithms for unloading and pre-marshaling [*9*].

Various models have been proposed in the literature including a simulation model for stacking containers in a container terminal through developing and applying a genetic algorithm (GA) for containers location assignment which minimizes total lifting time and therefore, increases service efficiency of the container terminals [*6*].

The authors have presented an algorithm selection benchmark based on optimal search algorithms for solving the Container Pre-Marshalling Problem (CPMP), an NP-hard problem from the field of container terminal optimization [*7-8*]. The CPMP deals with the sorting of containers in a set of stacks (called a bay) of intermodal containers based on their exit times from the stacks, such that containers that must leave the stacks first are placed on top of containers that must leave later. A recent approach for solving the CPMP to optimality [*8*] presents two state-of-the-art approaches, based on A* and IDA*. The authors then use parameterized versions of these approaches to form a benchmark for algorithm selection. An example (with 3 stacks, maximum height per stack is 3, and having a total of 6 containers) is high-lighted by the authors' CPMP for algorithm selection.
The authors have presented an exact algorithm based on branch and bound algorithms, and is shown to be NP-hard [*10*].

Two heuristic algorithms have been proposed to solve for the pre-marshalling problems [*11*]. While the existing literatures have had extensive discussions (including some numerical examples) on unloading and pre-marshalling (or re-shuffling) operations at the port terminals, step-by-step numerical algorithms have either not been given, or have been described without sufficient details. Thus, it is not an easy task to implement and compare the performance of various proposed algorithms. In this work, some simple heuristic algorithms for both "unloading" and "pre-marshalling" operations are proposed by the authors. Detailed step-by-step algorithms are explained and presented, so that the readers can reproduce the presented results. Several numerical examples are used to validate the proposed heuristic algorithms.

A secondary goal for this paper is to develop useful, user friendly, attractive Java computer animation for "teaching" these basic/important algorithms for optimizing the Terminal Container Yards unloading and pre-marshalling operations.

Recognizing the steadily decline in US Science Technology Engineering Mathematics (STEM) interests and enrollments, the National Science Foundation (NSF) and the White House have developed national strategies and provided the significant budget to STEM education research [*12-13*] in the past years, with the ultimate goals to improve both the quality and number of highly trained US educators, students workforce in STEM topics, in today highly competitive global markets. With the explosions of internet's capability and availability, it is even more critical to effectively train this future USA-STEM work-force and/or to develop effective STEM related teaching tools to reach a maximum possible number of "distance learners/audiences".

Various teaching philosophies have been proposed, tested and documented by the educational research communities, such as video lectures (YouTube), "flipped" class lectures (where students are encouraged to read the lecture materials at their own time at homes, and problem solving and/or

questions/answers sessions are conducted in the usual classroom environments), STEM summer camps, game-based--learning (GBL) [*14-16*], virtual laboratories [*17*] and concept inventory [*18*]. The final product from this work will help both the students and their instructor to not only master this technical subject, but also provide valuable tool for obtaining the solutions for homework assignments, class examinations, self-assessment tools, etc. The developed "educational version" of Java-based application should also help the "transportation professionals" since it has several desirable features, such as:

- A detailed, precise and clear step-by-step algorithm will be displayed in text and human voice during the animation of the algorithm (unloading or pre-marshalling).
- Options to hear animated voice in 2 major languages (English and Spanish).
- Options to input/output terminal container yard layouts (CVS file), or manually edit the layouts using an editor, or "randomly generating" layouts.
- Output of the "final/optimal" results can be exported to text, so that the users/learners can check/verify their "hand-calculated" results, which is an important part of the learning process.

2. UNLOADING AND PREMARSHALLING (RESHUFFLING) ALGORITHMS FOR TERMINAL STORAGE YARDS

External vehicles (trucks or trains) are responsible for transporting cargo containers in and out of the container yard; whereas the internal vehicles are responsible for transporting containers within the terminal from the storage yard to the quayside. The storage yard is where containers are stored temporarily until a vehicle arrives to further distribute it to its next location [9].

Two types of algorithms (Unloading and Pre-marshalling) are discussed and proposed along with a visualization and animation software tool for teaching, learning, and improving the algorithms. The unloading algorithm approach is based upon having the stack reshuffled to get to the priority container when a vehicle arrival. The pre-marshalling algorithm approach is based on having the stack ordered before the first vehicle arrives so that the priority container is on top.

GIVEN:

NS:= 4

MNCPS:= 3

Layout:= 8 containers

[*]	[*]	[4]	[*]
[8]	[*]	[2]	[6]
[1]	[7]	[5]	[3]
S1	S2	S3	S4

[*] means empty space

Figure 3 Initial Input Parameters and Layout (Given)

To facilitate the discussion in this section, the given input variables and layout are defined [see Figure 3].

- N = NS = the number of container stacks (columns).
- M = MNCPS = the maximum number of containers per stack (rows).
- NAES = the number of available empty spaces (total number of empty spaces in each stack).
- NESR = the number of empty spaces required.
- NCC = the number of cargo containers.
- TNS = the total number of spaces in the container yard

We also initialize and define some variables for collecting results:

- NM = the number of moves (container relocations).
- NR = the number of removes (containers unloaded).

First, we begin with a given initial storage layout [see Figure 3], the given N = NS = (4 stacks), M = MNCPS = (3 Tiers). From this we can compute the TNS (12 total spaces), NAES (4 available spaces based on the given initial layout) and NESR (3 Tiers) based on the equations:

$$TNS = (MxN)$$
$$NAES = TNS - NCC$$
$$NESR = MNCPS$$

Figure 3 provides a 2D initial layout that will be discussed throughout this paper. The number in each container represents the priority order; the order sequence in which the containers need to be unloaded on to a vehicle.
Once the initial parameters and layout are given, we begin each algorithm performing Step 0. Step 0 purpose is to validate if we have enough spaces to perform the algorithms. The verifications for both algorithms are define:

1. Unloading Algorithm
   - If the NAES is greater-than or equal to NESR-1. The NESR-1 spaces should allow us to get to the priority cargo container for removal.
   - If the NS is greater-than 1 container stacks. Need at least 2 or more container stacks to be able to do unloading. NAES should provide enough empty spaces initially to get to the priority cargo container if it's on the bottom.

2. Pre-Marshalling (or Reshuffling) Algorithm
   - If the NAES is greater-than or equal to NESR. Need NAES spaces in order to do reshuffling without removal.
   - If the NS is greater-than 2 container stacks. Need at least 3 or more container stacks to be able to do reshuffling. Note: some layouts can be solved without these validations for reshuffling, though these verifications avoid deadlock cases.

The remainder of the section provides an overview of each algorithm in pseudo code and walks through the proposed step-by-step numerical algorithms.

2.1 Unloading Algorithm

In the Unloading Algorithm, cargo containers remain where they reside and are unloaded when a vehicle arrives. If the prioritized cargo container is not on top of the Stack, then those cargo container above it must be relocated. This approach free up a space before the next vehicle arrives; therefore, the NAES is increased by 1 after every unloaded container.

Before the algorithm begins, the given initial input data is required (Figure 3). From this initial data, the algorithm determines if there's enough available empty spaces in the container stacks to solve this problem (Step 0). For this example, there's enough available empty space (NAES-1 >= NESR). Next, the algorithm determines if Cargo Containers can be unloaded from the Container Stacks (Step 1). This step loops until the Container Stacks are empty (all Cargo Containers unloaded). Before unloading Cargo Containers, this step determines the Cargo Container with the highest priority and then performs the Cargo Container move (Step 2).

Step 2 checks to determine if the given next highest priority Cargo Container is on top of the Container Stack. If it is, then it is removed (unloaded onto a truck); otherwise, the algorithm finds the Container Stack with a highest priority (sorted) but lesser priority than the one that needs to be moved. Once the move occurs, it then loops back and repeats until all the Cargo Containers are unloaded.

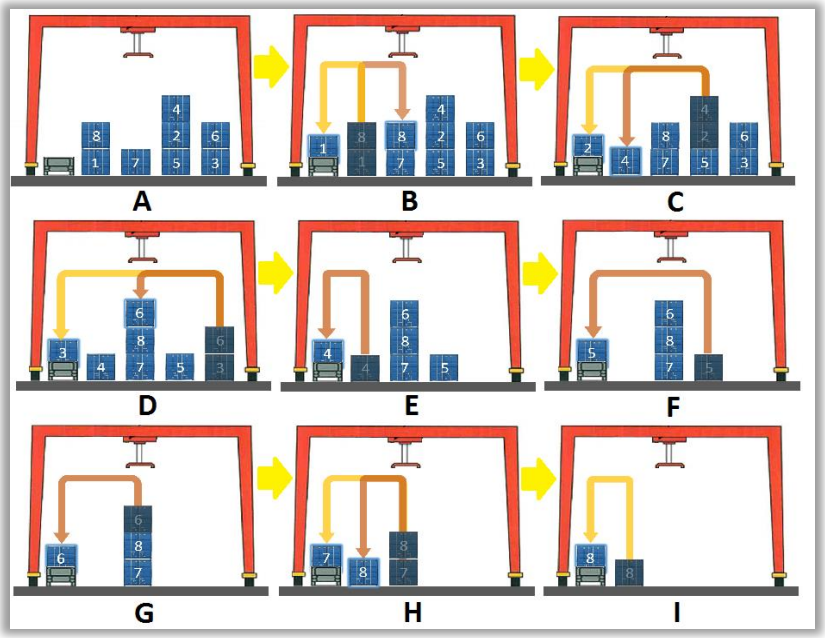


Figure 4 Unloading Algorithm (Movement Iterations)

For example, Figure 4 shows the iterations of the movements and unloads using the given initial data against the unloading algorithm. Figure 4B, shows that Cargo Container 1 is the next highest priority to unload. Though, in order to unload it those Cargo Container(s) above it must be relocated; therefore, Cargo Container 8 is relocated to Container Stack 2. Note that there's no Cargo Container is the highest priority container lesser than the one that needs to be relocated, so we randomly pick the closest Container Stack with an available space. We can improve this to pick the closest stack, pick the least number of containers in the stack, or pick the least priority container to research what kind of impact it would have. Once Cargo Container 8 relocated (number of moves = 1), Cargo Container 1 can be unloaded onto a vehicle (truck in this example).

Figure 4C shows that Cargo Container 2 is the next highest priority to unload. Though, in order to unload it those Cargo Container(s) above it must be relocated; therefore, Cargo Container 4 is relocated to Container Stack 1. Once Cargo Container 4 relocated (number of moves = 2), Cargo Container 2 can be unloaded onto a vehicle.

Figure 4D shows that Cargo Container 3 is the next highest priority to unload. Though, in order to unload it those Cargo Container(s) above it must be relocated; therefore, Cargo Container 6 is relocated to Container Stack 2. Once Cargo Container 6 relocated (number of moves = 3), Cargo Container 3 can be unloaded onto a vehicle.

Figure 4E shows that Cargo Container 4 is the next highest priority to unload. Cargo Container 4 is on top of Container Stack 1; therefore, no movement is required so Cargo Container 2 can be unloaded onto a vehicle.

Figure 4F shows that Cargo Container 5 is the next highest priority to unload. Cargo Container 5 is on top of Container Stack 3; therefore, no movement is required so Cargo Container 5 can be unloaded onto a vehicle.

Figure 4G shows that Cargo Container 6 is the next highest priority to unload. Cargo Container 6 is on top of Container Stack 2; therefore, no movement is required so Cargo Container 6 can be unloaded onto a vehicle.

Figure 4H shows that Cargo Container 7 is the next highest priority to unload. Though, in order to unload it those Cargo Container(s) above it must be relocated; therefore, Cargo Container 8 is relocated to Container Stack 1. Once Cargo Container 8 relocated (number of moves = 4), Cargo Container 7 can be unloaded onto a vehicle.

Figure 4I shows that Cargo Container 8 is the next highest priority to unload. Cargo Container 8 is on top of Container Stack 1; therefore, no movement is required so Cargo Container 8 can be unloaded onto a vehicle.

For the initial example given above, the final results show that there was a total of 4 container relocations (number of moves = 4) and all 8 Cargo Containers were unloaded onto a vehicle.

2.2 Pre-marshalling (Reshuffling) Algorithm

In the Pre-marshalling Algorithm, the container yard is reshuffled and prioritized before the first vehicle arrives. The goal is to have the container stacks in order with the highest priority containers on top and the lower priority on the bottom. This algorithm is identical to the Block World planning domain problem and is more complex than the unloading algorithm since it's limited to the initial number of spaces available whereas the unloading algorithm frees up a space when a priority container is unloaded onto a vehicle.

The algorithm begins by initializing the given input data required (Figure 3) along with additional variables for book-keeping purposes. From this initial data, the algorithm determines if there's enough available empty spaces and number of stacks available in order to perform reshuffling of cargo containers with the number of container stacks (Step 0). For this example, there's enough available empty space (NAES >= NESR and NS > 2).

After the initial given data is verified for reshuffling, Step 1 is performed. This step begins the main loop which loops through all the Container Stacks and checks if all the Container Stacks are in order. If they're not then the goal is to reshuffle all the highest priority Cargo Containers to the top and the lowest to the bottom while each Container Stack is sorted from highest to lowest in priority. In doing so the algorithm must determine the source and destination Container Stacks.

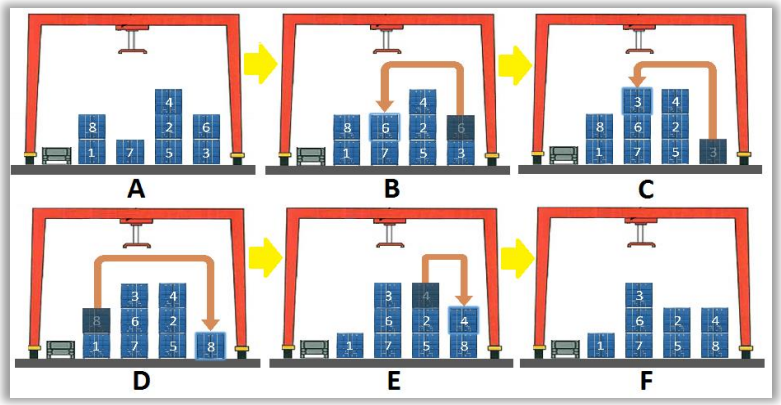


Figure 5 Pre-Marshalling (Reshuffling) Algorithm (Movement Iterations)

Step 1 first begins to look at any container stack with a single cargo container. If there is on that exists, the algorithm determines if either it's the source or destination Container Stack. In Figure 5A, the initial container layout example shows that Cargo Container 7 is the only Cargo Container in Container Stack 2 so it meets this condition. The same step determines if this Cargo Container can be placed on top of a sorted Container Stack with that selected Container Stack still prioritized or if another higher priority (lower number) Cargo Container can be placed on top of Cargo Container 7 in Container Stack 2. Step 1 determines that Container Stack 2 is selected as the destination Container Stack since there's not another sorted Container Stack where Cargo Container 7 can be placed on. Figure 5B shows that Container Stack 4 is determined to be the source Container Stack; therefore, Cargo Container 6 is placed on top of Cargo Container 7 in Container Stack 2. After this move (relocation), the algorithm is then restarted back at the beginning of the loop.

Step 1 is again repeated, once again there's a condition where one Cargo Container exists within a Container Stack shown in Figure 7B. The algorithm again needs to determine if this is the source or destination Container Stack. In this case, Container Stack 2 is a sorted stack that will allow Cargo Container 3 to move on top of it; therefore, Container Stack becomes the source Container Stack while Container Stack 2 becomes the destination. Figure 5C shows that Cargo Container 3 is moved (relocated) from Container Stack 4 onto Container Stack 2. After this move (relocation, the algorithm is then restarted back at the beginning of the loop.

Note that the first two moves are not placed in book-keeping for containers to move back towards their original Container Stack. The original stack being looked at is Container Stack 2, there have been no Cargo Containers moved from this Container Stack.

Once again Step 1 is repeated, during this iteration there are no Container Stacks with one Cargo Container so that check doesn't meet that condition. From this point, the algorithm looks for an unsorted Container Stack with the least amount of Cargo Containers that is not empty. In this case,

Container Stack 1 is selected, refer to Figure 5D. From this point, the algorithm enters Step 2 which is to reshuffle (relocated) the top Cargo Container and temp spot it onto a different Container Stack. Step 2 is now entered with Container Stack 1 as the selected source Container Stack. The goal of this step is to reshuffle this selected Container Stack and temp spot the Cargo Containers into a different Container Stack. The step keeps track of the least priority Cargo Container being moved (relocated) so that it can always be placed on the bottom of the Container Stack when moved back. In Figure 5D, the algorithm takes the top Cargo Container in Container Stack 1 which is Cargo Container 8 and picks the Container Stack with the least amount of Cargo Containers. Container Stack 4 is empty; therefore, it is selected as the destination Container Stack. Cargo Container 8 is then moved from Container Stack 1 to Container Stack 4. In this case, since Container Stack 4 is empty and Cargo Container 8 is the Cargo Container with the least priority in Container Stack 1, the algorithm does not place Cargo Container 8 in book-keeping for containers to move back towards their original Container Stack. After this move, the algorithm loops back and looks at the next Cargo Container on Container Stack 1 so Step 2 is repeated.

In this next iteration in Step 2, the algorithm continues to try to temp spot the next top Cargo Container in the selected Container Stack 1. Before the algorithm checks for a destination Container Stack for Cargo Container 1, it looks for the highest priority Cargo Containers in the other unsorted Container Stacks. Container Stack 3 in Figure 5E shows that Cargo Container 4 is the top Cargo Container for all unsorted Container Stacks. Since this Cargo Container isn't in the Cargo Containers to move back book-keeping, it's moved (relocated) to the Container Stack 4 since it's a higher priority than Cargo Container 8. Since Cargo Container 6 isn't from the original source Container Stack it isn't placed within the Cargo Containers to move back book-keeping. After this move, the algorithm loops back and looks at the next Cargo Container on Container Stack 1 so Step 2 is repeated. At the very beginning of this loop, there's a check to see if all the Container Stacks are in order. Figure 5F shows that all Container Stacks are sorted from highest to lowest priority after Cargo Container 4 was moved to Container Stack 4; therefore, the algorithm breaks out of its loop and returns back to Step 1 and continues after it called Step 2's procedures. After this procedure there's a check to see if all the Container Stacks are in order so that it too can trigger the restart of the Step 2's main loop. At the very top of the main loop, if all the Container Stacks are in order the algorithm can proceed to Step 3. Figure 5F shows that final state of the reshuffled sorted Container Stacks.

In Step 3, all the Container Stacks are sorted from highest priority Cargo Containers on top with the lowest priority on the bottom. With all the highest priority Cargo Containers on top, unloading Cargo Containers on to vehicles doesn't require any more reshuffling.

For the initial example given above, the final results show that there was a total of 4 container relocations (number of moves = 4) and all 8 Cargo Containers were unloaded onto a vehicle.

3. JAVA COMPUTER ANIMATED SOFTWARE TOOL FOR TEACHING TERMINAL CONTAINER YARD (STORAGE YARD) UNLOADING AND PREMARSHALLING (RESHUFFLING) ALGORITHMS

The software is custom written in Java [*19-22*] and is meant to create 2D animations and provide voice explanations for both the unloading and pre-marshalling algorithms for the Terminal Container Yard operations. The Java software is 100% written in Java and developed from scratch using several open-source software. The software uses various third-party libraries, such as the 2D graphics library (Geosoft's G 2D) for animations, the text-to-speech library (Google API Translate) for voice,

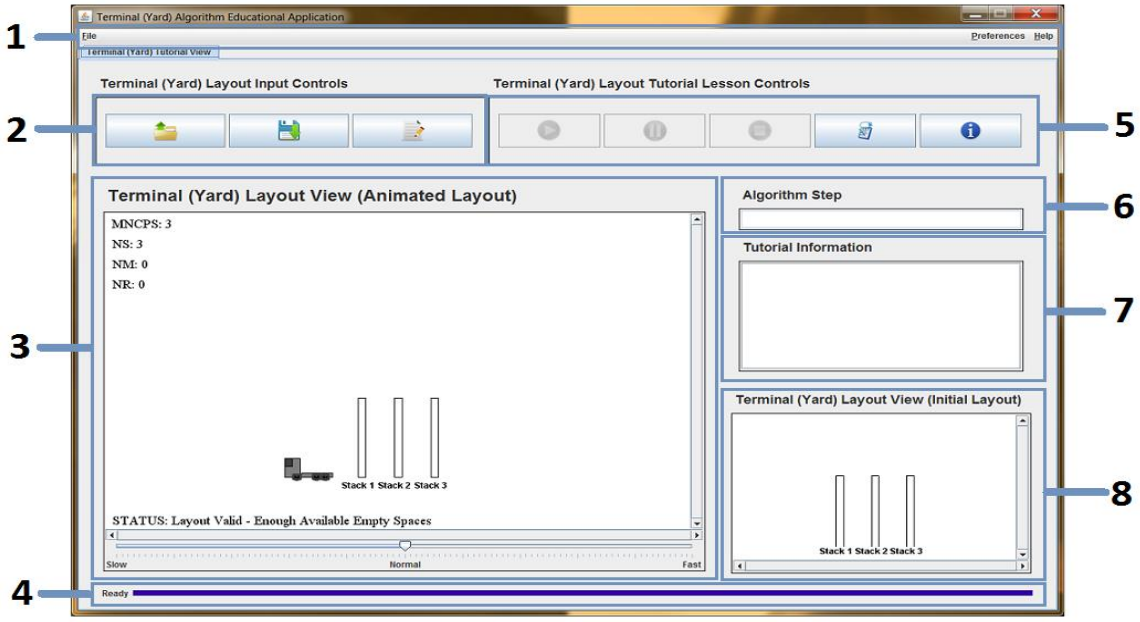


Figure 6 Terminal (Yard) Algorithm Educational Application

and the matrix library (Efficient Java Matrix) for data storage. Applying open-source libraries to this software allowed lots of Graphical User Interface (GUI) functionality and features to be developed. The software GUI is developed with JFC/Swing which is included within the Java Development Kit (JDK). The Main GUI consists of eight main components as shown in Figure 6.

1. Menu Buttons (File, Preferences, and Help)
   Provides basic options, such as an option to open/save Container Layouts, an option to provide terse, verbose, or no voice step-by-step instructions, an option to change the voice language to English, Spanish, and Chinese, and an option to display a user manual.
2. Input Control Buttons
   Buttons on the main window to open, save or edit a container yard layout.
3. Terminal (Yard) Layout Animation View
   Provides an animated view for Cargo Container movement and attributes (NS, MNCPS, NM, and NR fields) updates.
4. Status View

Provides a status view which displays meaningful status: ready, play, or paused.

5. Lesson Control Buttons
   Buttons on the main window to play, pause, or stop a lesson (an algorithm). The buttons are enabled when an initial layout is given. The information button provides a user guide for the Main GUI.
6. Algorithm Step
   Textbox view of the current algorithm step (during play session).
7. Tutorial Information View
   Textbox view of instructions or steps being done within the algorithm (during play session).
8. Terminal (Yard) Layout Initial View
   Provides a given initial layout view of the terminal yard.

The Terminal Yard Editor GUI in Figure 7 provides a means to edit terminal layouts and consists of 5 main components as shown in Figure.

1. Display and Information View
   This view displays the NS and MNCPS for this layout. It also provides a random button that will randomly generate a layout for the given NCC and the information button provides a user guide for this GUI.
2. Editor Buttons
   The editor buttons allows the user to add/remove a container from the container stack and to increase/decrease the NS and MNCPS.
3. Zoom Buttons
   The zoom buttons zooms in/out or reset the view back to the original state.
4. Visualization Editor View
   The visualization editor allows the user to drag-and-drop Cargo Containers from one Container Stack to another (left-click hold to drag) and to select a Cargo Container to remove (right-click).
5. Bottom Buttons
   These buttons either accepts the user's edited layout or cancels.

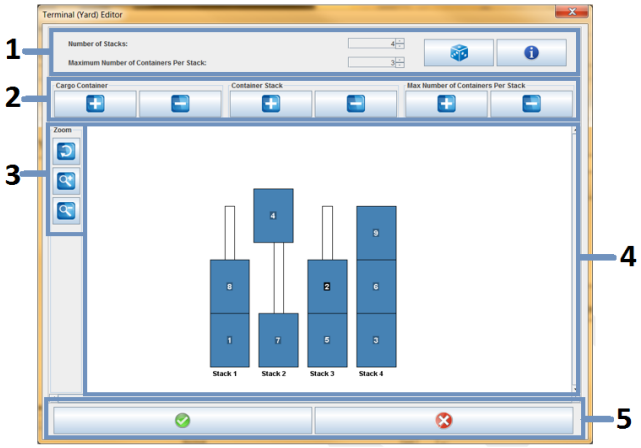
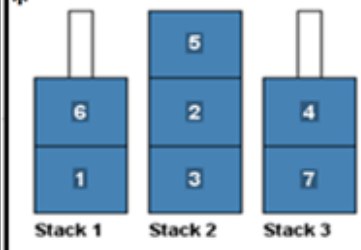
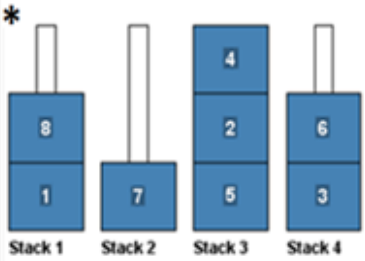
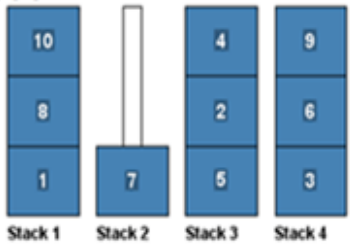
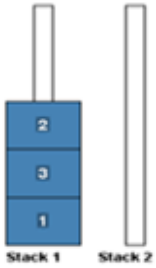


Figure 7 Terminal Editor GUI

As mentioned above, the software has several special functionality and features, such as animating the Cargo Container movements based on a selected algorithm applied towards a given Terminal Yard Layout. It supports a step-by-step animation with voice for a given terminal layout and a selected algorithm. This mode aims to help students understand the working mechanism of the algorithm and to visually fine tune it (developers). To achieve this goal, the software provides steps within the algorithm with voice and animates the movement of containers from one stack to another. In addition to the visible perception, a computer generated (text-to-speech) voice is accompanied during the step and while the animation is playing.

4. NUMERICAL EXAMPLES/RESULTS

Several terminal container yard layout where referenced from the literatures and some referenced from the classroom lecture [*4*]. These layouts were used within the Java computer animation software and executed for both the unloading and pre-marshalling (or reshuffling) algorithms. **A demo video of the both the unloading and pre-marshalling algorithm's animation and result can be viewed online from any web browser using the website provided in reference [*4*].** The algorithm results are recorded below within the table. Results in Table 1 that show "N/A" for the pre-marshalling algorithm did not provide any results since the layout did not meet the validation checks described earlier in Section 2 in this paper. The main reason for no results for the pre-marshalling algorithm was either that the NAES was not greater-than or equal to NESR and/or the NS was not greater-than 2 container stacks.

Container Yard Layout	Algorithm	NS	MNCPS	TNS	NCC	NAES	NESR	NM	NR
* 	Unloading	3	3	9	7	2	3	3	7
	Reshuffling	3	3	9	7	2	3	N/A	N/A
* 	Unloading	4	3	12	8	4	3	4	8
	Reshuffling	4	3	12	8	4	3	4	8
*** 	Unloading	4	3	12	10	2	3	9	10
	Reshuffling	4	3	12	10	2	3	N/A	N/A
*** 	Unloading	2	5	10	3	7	5	3	3
	Reshuffling	2	5	10	3	7	5	N/A	N/A

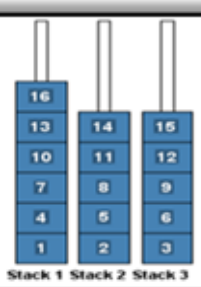
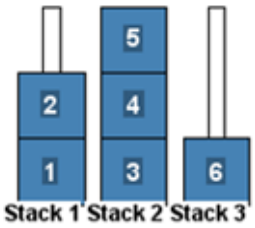
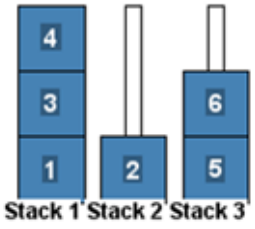
*** 	Unloading	3	8	24	16	8	8	39	16
	Reshuffling	3	8	24	16	8	8	51	16
** 	Unloading	3	3	9	6	3	3	3	6
	Reshuffling	3	3	9	6	3	3	3	6
* 	Unloading	3	3	9	6	3	3	4	6
	Reshuffling	3	3	9	6	3	3	6	6

Table 1 Unloading and Pre-Marshalling (Reshuffling) Algorithm Results
(denotes from reference [9], ** denotes from reference [7], and *** denotes from reference [4])*

5. CONCLUSION AND FURTHER WORKS

In this paper, the terminal container yard unloading and pre-marshalling (or reshuffling) algorithms have been proposed, explained and verified through several numerical examples. Detailed descriptions of the proposed step-by-step algorithms are also provided so that the readers can reproduce the presented results. A secondary goal for this paper is to develop the Java computer animated software (associated with the proposed unloading and pre-marshalling algorithms) which can be used as an additional tool for teaching and learning these unloading and pre-marshalling algorithms. **A demo video of the both the unloading and pre-marshalling algorithm's animation and result can be viewed online from any web browser using the website provided in reference [*4*].** The software tool is custom developed in Java and integrated with open-source libraries. It provides desirable teaching/learning features for these complex algorithms for unloading/pre-marshalling, such as:

- Software tool with a user friendly and easy to use GUI.
- 2D Graphical visualization and animation for displaying container movement (relocation).
- 2D Graphical visualization to edit a new or existing terminal container yard layout.
- Ability to allow the user/learner to load/store a terminal container yard layout.
- Ability to allow the user/learner to output/save the step-by-step results.
- Provide a clearly and attractive computer animated voice that provides step-by-step instructions of the algorithms.
- Animated voice can be configurable to translate text-to-speech into another language, such as English, Spanish and Chinese.
- Provides results of the algorithms, such as NM, NR, NS, and MNCPS.

The software tool itself can be used as a visualization and animated framework for including other implementations of the unloading/pre-marshalling algorithms. Applying the JBoss Rules (Drools) engine which uses the Rete Algorithm could potentially be considered as further work. The Rete Algorithm is a pattern matching algorithm for implementing production rule systems. It's known to be helpful in planning rule-based systems [*5*]. By using this approach we can experiment with JBoss rules and using the Java animation teaching tool to refine and optimize both the unloading and pre-marshalling algorithms for Terminal Container Yard operations.

6. ACKNOWLEDGMENT

This paper was in part funded by Mid-Atlantic Transportation Sustainability University Transportation Center (MATS UTC).

REFERENCES

[1] Kap Hwan Kim, Hans-Otto Gunther, “Container Terminals and Cargo Systems: Design, Operations Management, and Logistics Control Issues”.
[2] Yusin Lee and Nai-Yun Hsu, ‘An optimization model for the container pre-marshalling problem’, Computers & Operations Research.
[3] John Slaney, Sylvie Thiebaux, “Blocks World revisited”. Artificial Intelligence (14 June 2000).
[4] Nguyen, D.T., Vi Nguyen, Nga Pham, Gelareh Bakhtyar, “Yard Crane Scheduling (YCS), Unloading and Reshuffling Formulations/Algorithms”, CEE-775/875: Computational Transportation Course, Spring’2015 semester (January 14 2015). Demo: http://www.lions.odu.edu/~imako001/
[5] JBoss Rules (Drools) and the Rete Algorithm. https://docs.jboss.org/drools/release/5.2.0.Final/drools-expert-docs/html/ch03.html
[6] P. Sriphrabu, K. Sethanan, and B. Amonkijpanich, “A Solution of the Container Stacking Problem by Genetic Algorithm”, IACSIT International Journal of Engineering and Technology, Vol. 5, No. 1 (Feb. 2013).
[7] K. Tierney, and Y. Malitsky, “An Algorithm Selection Benchmark of the Container Pre-marshalling Problem”.
[8] Tierney, K., Pacino, D., Voß, S.: Solving the pre-marshalling problem to optimality with A* and IDA*. Technical report WP#1401, DS&OR Lab, University of Paderborn (2014).
[9] Lehnfeld, J., Knust, S. “Loading, Unloading and Pre-marshalling of Stacks in Storage Areas: Survey and Classification”, Eur. J. Oper. Res. 239(2), 297–312 (2014).
[10] M. van Brink, and R. van der Zwaan, “A Branch and Price Procedure for the Container Pre-marshalling Problem”.
[11] S. Huang, and T. Lin, “Heuristic Algorithms for Container Pre-Marshalling Problems”, Computers & Industrial Engineering 62, pages 13-20 (2012).
[12] National Science Foundation (NSF), National Science Board (NSB), “A National Action Plan for Addressing the Critical Needs of the U.S. Science, Technology, Engineering, and Mathematics (STEM) Education System” (October 30, 2007).
[13] Executive Office of the President of the United States, “Federal Science, Technology, Engineering, and Mathematics (STEM) Education 5-Year Strategic Plan”, a Report from the Committee on STEM Education, National Science and Technology Council (May 2013).
[14] Nguyen, D.T., A.A. Mohammed, S. Kadiam, and Y. Shen “Internet Chess-Like Game and Simultaneous Linear Equations”, Global Conference on Learning and Technology Penang (Island), Malaysia; http://www.aace.org/conf/cities/penang; (May 17-20’2010).
[15] Nguyen, D.T., Y. Shen, A.A. Mohammed, and S. Kadiam, “Tossing Coin Game and Linear Programming Big M Simplex Algorithms”, Global Conference on Learning and Technology, Penang (Island), Malaysia; http://www.aace.org/conf/cities/penang; (May 17-20’2010).
[16] Nguyen, D.T. (PI), Autar K. Kaw (Co-PI), Ram Pendyala (Co-PI), and Gwen Lee-Thomas (Co-PI), “Collaborative Research: Development of New Prototype Tools, and Adaptation and Implementation of Current Resources for a Course in Numerical Methods”. NSF funded educational grant (Proposal # 0836916; DUE-CCLI Phase 1: Exploratory); (funding period: January 1’2009 – July 30’2011).
[17] Nguyen, D.T. (P.I. = Prof. S. Chaturvedi), “Implementation Grant: Simulation and Visualization Enhanced Engineering Education,” National Science Foundation (NSF), funding period: September 2005 – September 2009.

[18] Autar Kaw (P.I.) et al., “Improving and Assessing Student Learning in an Inverted STEM Classroom Setting”, NSF (Division of Undergraduate Education) awarded grant # 1322586 (September 2013-September 2016).
[19] Efficient Java Matrix Library (EJML). http://code.google.com/p/efficient-java-matrix-library/wiki/EjmlManual
[20] Google Translate Java. http://code.google.com/p/google-api-translate-java/
[21] G Java 2D Generic Graphics Library. http://geosoft.no/graphics/
[22] Java Platform Standard Edition. http://www.oracle.com/technetwork/java/javase/downloads/index.html