

Energy Efficient Container Yard Stacking

Part 2

Alexandre Goussiatiner, Container Terminal and Transportation Specialist, Modern Port Technologies Inc., E-mail: alexg@modernport.com, Web: www.modernport.com

Reinforcement Learning (RL) Stacking Strategy

Container Stacking component (CS) receives as input a time sequence of containers which are being received or reshuffled C_t where $t = 0, 1, 2, \dots$. CS determines the stacks, where empty slots are available, and selects an action $a_t \in \{0, 1, \dots, n\}$ which places C_t to one of the stacks in the section.

In order to be able to make the decision, following data is made available to CS:

- x_t - state vector which represents heights of the stacks
- $p(a_t)$ reshuffling probability indexes for all the potential stack
- characteristics of the container C_t

CS selects action a_t which provides minimum for the following policy function:

$$\pi(a_t) = y_{pl}(a_t) + p(a_t)y_{res}(gw) + y_{ret}(a_t) + we_t(a_t),$$

where first three elements in the formula gives us an estimation for the complete power input required for handling container C_t during its life time inside the section. This sum represents immediate factors that are used for the decision making:

$y_{pl}(a_t)$ kWh -power input required for the placement. The quantity is calculated by the mechanical software component which takes into account container weight, configuration of the stacks and trajectory of the laden spreader during the placement.

$p(a_t)y_{res}(gw)$, kWh - product of reshuffling index $p(a_t)$ and $y_{res}(gw)$ - average power input required for the container reshuffling with the gross weight gw [kg].

$y_{ret}(a_t)$ -kWh power input required for the retrieval from the stack. It is assumed that

$$y_{ret}(a_t) = y_{pl}(a_t).$$

Also,

$e_t(a_t)$ kWh - is an estimation of the energy input after action a_t is taken. $e_t(a_t)$ represents long term consequences of a_t .

w - coefficient, which represent importance of the $e_t(a_t)$ prediction.

Learning Process

To calculate $e_t(a_t)$, CS uses special *prediction function*. Assuming that action a_t is taken, the task for the *prediction function* is to produce an estimate, or prediction of the following quantity:

$$Y_t = y_{t+1} + \gamma y_{t+2} + \gamma^2 y_{t+3} + \dots = \sum_{i=1}^{\infty} \gamma^{i-1} y_{t+i} ,$$

where γ is a discount factor and y_{t+1} -is the power input required to handle

C_{t+1} container during its complete life-time. The discount factor determines how strongly future values of y influence current predictions. When $\gamma=0$, e_t predicts just the next value of y , that is, y_{t+1} . As γ increases toward one, values of y in the more distance future become more significant.

This *prediction function* is in a form of large lookup table $Q(x_t, a_t)$ which maps state-action pairs (x_t, a_t) with $e_t(a_t)$.

CS interacts with the environment and learns by updating the prediction function as follows:

For each $t = 0, 1, 2, \dots$, upon generating action a_t , CS remembers action a_{t+1} and waits for outcome after container c_{t+1} is retrieved from the section: (x_{t+1}, y_{t+1}) . CS uses the outcome and performs correction for the (x_t, y_t) entry in the lookup table as follows:

$$Q_{t+1}(x_t, a_t) = Q_t(x_t, a_t) + \alpha [y_{t+1} + \gamma Q_t(x_{t+1}, a_{t+1}) - Q_t(x_t, a_t)]$$

As you can see in the formula, the Q value for a state-action is updated by an error, adjusted by rate α . The rate is called the *learning rate*. Using $\alpha < 1$ causes the table entries to approach the *expected* prediction targets gradually.

Of course each estimate is a prediction because it involves future values of y . Is it more accurate than random guessing? Casual observations and the experiments, performed in

the simulation environment, indicate that there is regularity between state-action pair and future values of the y . Because of that answer to the question should be positive.

Comparison with the Baseline Strategy

We used following partial productivity measures to evaluate the strategies.

Average Power Consumption: $\alpha = \frac{Y_{tot}}{N}$ kWh/cont ,

where

Y_{tot} - total power input for the container handling ,

N - container throughput (total number of containers handled).

Note, that Y_{tot} does not include power input required for the crane idling, gantry traveling and reallocations. We are assuming that those power input elements will be the same for RL and baseline strategies.

Reshuffling Index: $\beta = \frac{R_{tot}}{N}$,

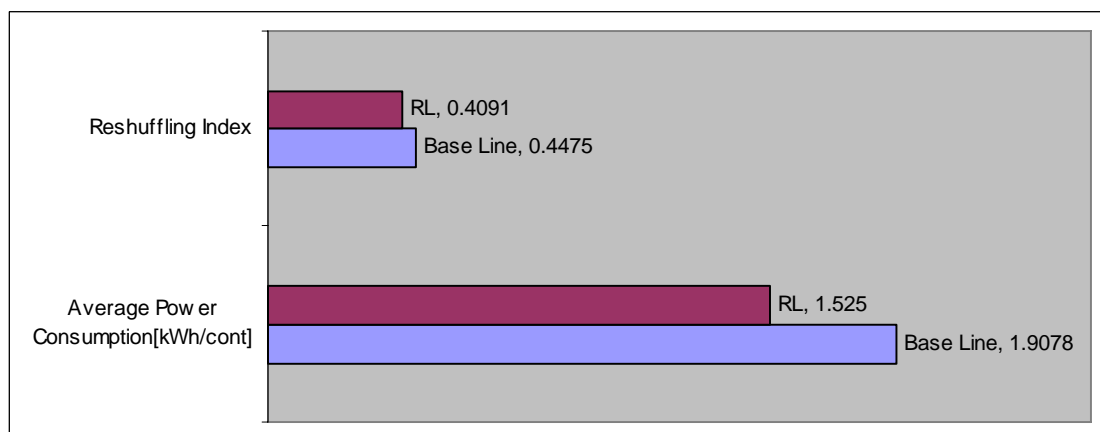
where

R_{tot} - total number of reshuffling moves performed by crane,

N - container throughput .

Typical stacking strategy for the import yard is to allocate incoming containers to the lowest stack excluding reserved slots at the first stack near the traffic lane. The simulation was used to produce results for the baseline and CS strategies. Comparison of the results shows that RL stacking strategy can reduce the power input from 1,907 kWh/cont to 1.525 kWh/cont or by 20%. At the same time reshuffling index gets reduced by 9%.

Figure 1. Partial productivity measures



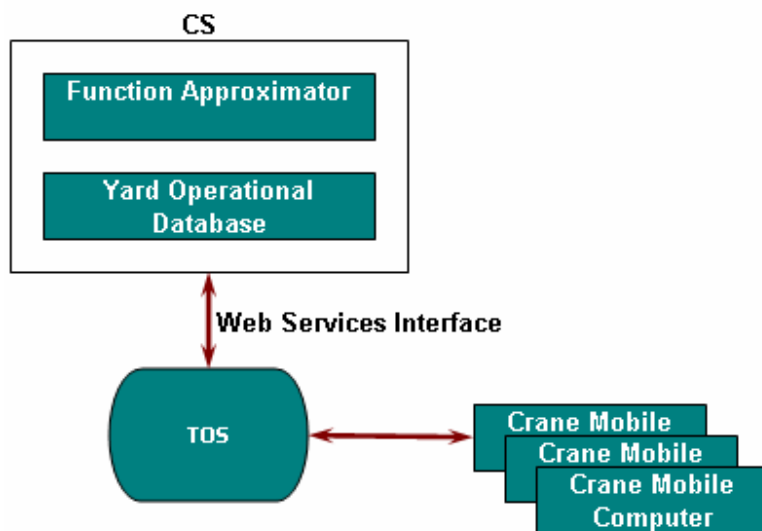
System Architecture

CS includes two main subsystems: Yard Operational Database and Function Approximator. The Yard Operational Database records all the yard container transactions and maintains temporary data on the stack configuration and containers. The Function Approximator produces stacking decisions and to 'learns' by updating the *prediction* function.

To facilitate integration with existing container terminals IT infrastructure, CS is made as independent software and hardware component (black box). Consequently CS can be implemented as part of any Terminal Operating System (TOS) or Yard Automation System. Implementation requires minimum or no alteration for the existing software and system architecture.

As it is presented in the Figure 2, TOS uses standard WEB Services interface to communicate with CS. TOS updates Yard Operational Database and requests stacking decisions when needed (Fig.2).

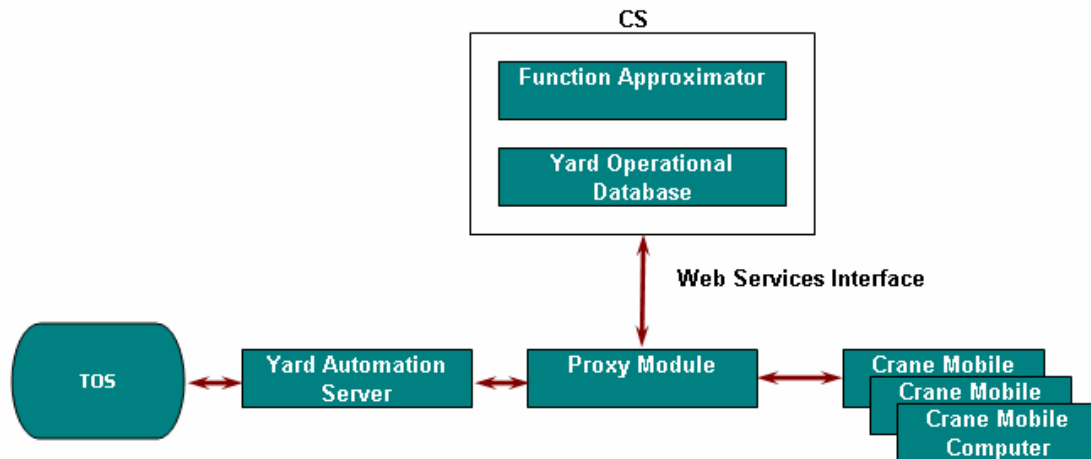
Figure 2. CS-Terminal Operating System Integration



To streamline integration with Yard Automation, we suggest developing proxy modules which will emulate the communication protocol between yard automation server and crane mobile computers. The proxy module will update CS using container transactions

received from the crane mobile computers. Although the variant requires development of the Proxy Module, it will not require any changes in the Yard Automation.

Figure 3. CS-Yard Automation System Integration



Final Comments

- Collected data suggests that crane operation, controlled by CS requires less power input and brings some financial savings. If we assume, that diesel cost remains at the current level (1.1 Euro per liter), reduction in the fuel cost will reach 7 568 Euros for 100 000 TEUs handled (see Table.1).

Table 1. Diesel Fuel Cost Reduction

Container Throughput [TEU]	100 000
Container to TEU ratio	1.4
Container Throughput [cont]	71 429
Power Input Reduction [kWh]	0.3830
Total Power Input Reduction [kWh]	27 357
Diesel fuel Power Efficiency [kg/kWh]	0.210
Diesel fuel density [kg/liter]	0.835
Reduction in consumption [liter]	6 880
Diesel fuel cost [Euros/liter]	€ 1.1
Diesel fuel cost reduction [Euros]	€ 7 568

- Under CS control, heavy lift containers will travel shorter horizontal and vertical distances. Consequently terminal operators can expect lower repair and maintenance cost for the cranes.

- It was confirmed, that power input as the main criterion for the optimization is not in a conflict with productivity indicators. The energy efficient solutions have reduced reshuffling and handling time.
- RL strategy can target other objectives: CS can be trained to improve various productivity measurers. For instance, using minimizing handling time as objective, will explicitly target net crane rate. The decision should be made by terminal operators based on certain operations conditions.
- The RL strategy described in the paper is not limited to the import stack. CS can use different policies depending on the type of stack including export stacks, reefer stacks, temporarily transshipment stacks, etc.