Stacking Under Uncertainty: We Know How To Predict, But How Should We Act?

Hado van Hasselt*, Han La Poutré*†
* Intelligent Systems, Centrum Wiskunde & Informatica, Amsterdam
†Decision Support Systems, Information and Computing Sciences, Utrecht University

***Abstract*—We consider the problem of stacking containers in a given set of stacks of fixed maximum capacity when the pick-up times are stochastic with unknown probability distributions. The goal is to minimize the expected number of times a container is picked up while it is not at the top of its stack. We formulate several algorithms under varying assumptions about the available knowledge about the pick-up-time distributions. We distinguish four qualitatively different settings: 1) we know nothing about the actual distributions, 2) we have point estimates of the means, 3) we have point estimates of means and variances, or 4) we have historical samples of actual pick-up times. For each assumption we propose one or more algorithms. We test the algorithms empirically in many different scenarios, considering both sequential and concurrent arrivals. Additionally, we consider the computational complexity and ease of use of each algorithm.**

I. Introduction

This paper is inspired by a real-world problem at a terminal at the Port of Rotterdam, Europe's largest port. On Sunday a large shipment of refrigerated containers (*reefers*) arrives at the terminal, to be picked up by trucks for further land-based transport. Reefers require external power to stay cool, and terminals typically only have a limited number of ground spaces that provide this power [1]. As a result, the reefers need to be stacked on top of each other.

Often, many trucks arrive within a fairly short time window on the following Monday morning to pick up the reefers, resulting in traffic jams at the terminal. An important reason for these undesired delays is that many reefers are not at the top of their stack when they are picked up. Unfortunately, the actual pick-up times are typically not known in advance, so we cannot use these to stack the containers. We investigate this problem in some generality, considering different assumptions about the available knowledge about the pick-up-time distribution of each reefer, to give recommendations on how to stack items with uncertain pick-up times in this and similar settings.

We consider an abstract version of the real-world problem, focusing solely on algorithms to reduce the number of reshuffles on pick up. In practice, other considerations play a role, such as the capacity and workload of automatic stacking cranes. The real-time implementation of any stacking algorithm contains many steps that can be further optimized. Our work is easily extended to include specific customizations, but we leave this for future work. These issues are largely orthogonal to the issues we investigate, and we ignore them to allow for a more focused discussion. An additional advantage of this abstraction is that is easier to apply the proposed algorithms to a larger range of problems that feature last-in first-out stacks and stochastic removal times.

This research was partially funded by the SUPPORT project in the "Pieken in de Delta" program of AgentschapNL.

Context and Related Work

To make predictions about stochastic processes, such as the aforementioned pick-up times, many computational-intelligence techniques exist [2], [3]. The reefers have features—such as their contents and final destination—that have a predictive value for the expected pick-up time. For instance, we might cluster the reefers into distinct classes with unsupervised learning methods [4], [5] and estimate the pick-up-time distribution for each class from historical data. Or we might approximate the mapping from features to pick-up times as a (semi-)continuous function, for instance using neural networks [2], [6]. If we desire more than a point estimate of the pick-up times a Bayesian approach such as Gaussian processes [7], [8] can be used to include estimates of the variance.

However, these techniques do not tell us how to use these predictions to stack the reefers efficiently. In this paper, we investigate how to create useful algorithms that use these predictions to determine how to *act*. We do not delve into the details of the prediction problem, in order to focus fully on the decision problem that succeeds it. We consider a wide range of different scenarios, which differ in the assumed knowledge about the distributions of the pick-up times. Either 1) nothing is know about the actual pick-up times, 2) we have point estimates for the mean pick-up times, 3) we have point estimates of the means and variances, or 4) we have a set of historical samples for each item (or class). These four cases cover a large portion of the possible choices and we are pragmatic about how this information is obtained or constructed. For each assumption, one or more algorithms are proposed. In all cases we consider both batch arrivals where all reefers are available concurrently, and sequential arrivals where the reefers need to be stacked in the order of arrival.

Much previous work exists on stacking containers [9], [10], [11]. However, most papers consider either the stacking *within a ship* or *within the yard for later ship-based transport*. In both cases, the destination and the pick-up order of the containers is known in advance. This contrasts with our setting, where this pick-up time is stochastic. Furthermore, most previous algorithms are either random or group the containers into categories, wherein the individual containers are interchangeable [12]. When considering truck-based import transport,

containers are rarely interchangeable, since each truck typically collects a specific container.

Although our inspiration comes from stacking containers, there are other domains for which this work can be useful. For instance when dispatching trams in a storage yard, the tracks in the yard are essentially last-in first-out buffers similar to the stacks we consider. The tram-dispatching problem has been investigated with stochastic arrival times [13] but, to our knowledge, not yet with stochastic departure times.

As an alternative to the methods we propose, one might consider using statistical decision algorithms such as considered in reinforcement learning [14]. Since these methods learn from interactions with an environment, this would require either a simulator or sufficient historical data to act as pseudo-interactions. Even then, although quite some literature exists to extend these algorithms to large and continuous state and action spaces [15], [16], [17], these algorithms are not easily applied to a problem with large nominal state and action spaces, such as considered in this paper. This does not mean reinforcement learning can not be usefully applied in such domains, but we leave a more thorough investigation of this to future work.

II. Problem Setting

There are n items that need to be placed into m stacks, where $n > m$. Each stack has a maximum capacity of $h_{\max}$. Let s_i denote the i^{th} item (counted from the bottom, starting at $i = 1$) in stack s. Let $|s|$ denote the current number of items in s. Let T_x denote the actual time item x is picked up. When the stack is clear from the context, we write T_i rather than T_{s_i}. The true pick-up time of an item is determined externally and can be viewed as a random variable with a fixed unknown distribution. Each stack is a last-in first-out buffer, such that only the top item is directly available. When an item is picked up while it is not at the top of its stack, a fixed cost of 1 is incurred. This cost is independent on the number of items on top of the removed item. The goal is to minimize the cumulative costs of all stacks. We assume the order of the remaining stack is unchanged after removing an item.

A. Expected Costs

Let $s_{(i)}$ denote the i^{th} item that is picked up from stack s. The actual cost $C(s)$ of s can recursively be computed with

$$C(s) = C(s \setminus \{s_{(1)}\}) + \mathcal{I}\left(T_{|s|} \neq \min_i T_i\right) \quad , \tag{1}$$

where $\mathcal{I}(\cdot)$ is the indicator function which is equal to one when its argument is true and equal to zero otherwise, and where $s \setminus \{s_{(i)}\}$ denotes the stack of height $|s| - 1$ after removing element $s_{(i)}$. Since the T_i are random, $C(s)$ is random as well. Let $F_x(t) \equiv P(T_x \leq t)$ denote the cumulative distribution function (CDF) of the pick-up time of any item x, and let $f_x(t) \equiv \frac{d}{dt}F_x(t)$ be the corresponding probability distribution function (PDF), which for simplicity we assume to exist everywhere. When the stack is clear from the context, we write $F_i(t)$ rather than $F_{s_i}(t)$.

Theorem 1. *The expected cost of a given stack s is equal to*

$$\mathbb{E}\{C(s)\} = |s| - 1 - \int_{-\infty}^{\infty} \sum_{i=1}^{|s|-1} f_i(t) \prod_{j=i+1}^{|s|} F_j(t)\, dt \quad . \tag{2}$$

All proofs are shown in the appendix at the end of this paper. The proof of Theorem 1 uses the following lemma, that gives an alternative formula for the cost of a stack.

Lemma 1. *For any stack s, the cost in* (1) *is equal to*

$$C(s) = \sum_{i=1}^{|s|-1} \mathcal{I}\left(T_i < \max_{i<j\leq|s|} T_j\right) \quad . \tag{3}$$

Although it might seem more intuitive to use (1), which considers the items in the order that they are picked up, the expectancy of (3) is much easier to compute. In definition (1), $s_{(1)}$ in the first term on the right-hand side is itself a random variable. Therefore, the implied recursion in (1) computes the expected cost for $|s|$ different possible resulting stacks. For each of these stacks, it then computes $|s| - 1$ recursions, and so on. Some of these recursions will be equivalent, but even if we take that into account this approach still computes $\sum_{i=1}^{|s|} \binom{|s|}{i} \in \Theta(2^{|s|})$ recursions in total. In contrast, to compute the expectancy of $C(s)$ in (3), we only need to compute the $|s| - 1$ probabilities $P\left(T_i < \max_{i<j\leq|s|} T_j\right)$, which can be done in $\Theta(|s|)$ computations in total. The result can be expressed with the single integral in (2). This is especially convenient when we cannot solve this integral analytically, and we have to resort to—typically more computationally intensive—numerical integration procedures.

B. Marginal Costs For Adding New Items

We consider the marginal cost of placing x on top of stack s, resulting in a new stack $(x : s)$.

Theorem 2. *The expected marginal cost*

$$M(x, s) \equiv \mathbb{E}\{C(x : s)\} - \mathbb{E}\{C(s)\} \quad , \tag{4}$$

of placing an item x on a stack s, is equal to

$$\int_{-\infty}^{\infty} f_x(t)F_{|s|}(t) + (1 - F_x(t)) \sum_{i=1}^{|s|-1} f_i(t) \prod_{j=i+1}^{|s|} F_j(t)\, dt\,. \tag{5}$$

Often, we have too little information to compute $M(x, s)$ exactly. We can then approximate (5) or minimize a related quantity. The following theorem gives useful general bounds.

Theorem 3. *For any s and x,*

$$P\left(T_{|s|} < T_x\right) \leq M(x, s) \leq \sum_{i=1}^{|s|} P\left(T_i < T_x\right) \quad . \tag{6}$$

The lower bound is cheap to compute since it only depends on the top item in s, but it can be loose for large stacks.

III. Algorithms

In many applications, items arrive one by one and need to be stacked immediately. It is then pointless to consider all possible ways to stack the items. Even when all items are available before stacking begins, it quickly becomes intractable to consider all possible solutions. Therefore, we only consider algorithms that stack the items sequentially. In the batch setting where all items are available before stacking, we sort them by

Algorithm 1 A generic algorithm to iteratively stack items.

Input: an ordered set of n items $\{x_i\} \subset \mathcal{X}$, a set of m stacks $\{s\} \subset 2^{\mathcal{X}}$ and a cost function $c : \mathcal{X} \times 2^{\mathcal{X}} \to \mathbb{R}$
for i from 1 to n **do**
  Find $s_* = \arg\min_s c(x_i, s)$.
  Place x_i on top of stack s_*.
end for

estimated pick-up times (assuming these are available), and proceed as if the items arrive sequentially in that order.

Our generic algorithm is shown in Algorithm 1. For each item, a cost is calculated for all available stacks. The item is placed on the stack with the lowest cost. Below, we propose different specific cost functions c that lead to concrete implementations of this algorithm. The cost functions vary over a few dimensions. The most important dimensions are the computational complexity and the required prior knowledge. We distinguish four knowledge levels: 1) we know nothing about the pick-up-time distributions, 2) we have point estimates of the mean pick-up time for each item, 3) we have point estimates of mean and variance, or 4) we have a set of samples from the actual distribution. The second case is common when the expected pick-up time is estimated with a scalar-valued function on some features of the items. The third case is similar, but assumes something like Gaussian processes [8] rather than (non-linear) regression.

We define $c(x, s) = 0$ when $|s| = 0$ and $c(x, s) = \infty$ when $|s| = h_{\max}$. That is, the cost of placing an item on an empty stack is zero, and it is impossible—hence the infinite cost—to place an item on a full stack. Below we can therefore assume $0 < |s| < h_{\max}$ without loss of generality. The following important theorem is used several times.

Theorem 4. *Let $f : \mathbb{R} \to \mathbb{R}$ be a strictly monotonically increasing function. Then, cost functions $c(x, s)$ and $f(c(x, s))$ result in equivalent stackings for any set of items.*

A. No Knowledge About Expected Pick-Up Times

Suppose no information about the pick-up times is available. We consider three algorithms: random ($c_{\text{R}}(x, s) \sim U(0, 1)$), breadth-first ($c_{\text{BF}}(x, s) = |s|$) and depth-first ($c_{\text{DF}}(x, s) = h_{\max} - |s|$). The breadth-first algorithm places each item on the lowest available stack. The depth-first algorithm places it on the highest stack that is not yet full. This is obviously a poor choice, typically resulting in a few high stacks that increase the chance of conflicts; the algorithm is merely included for completeness. All algorithms, including depth-first, first fill all empty stacks with at least one item, because we defined this cost to be zero. If we store the height of each stack, the complexity of all these algorithms is $O(m)$ per item.

B. Point Estimates of the Mean

Assume we have a point estimate $\hat{\mu}(x)$ for the mean pick-up time $\mathbb{E}\{T_x\}$ of each item x. For instance, a scalar-valued function of the features was regressed on historical pick-up times. Although having only $\hat{\mu}(x)$ restricts our ability to approximate $M(x, s)$, we can define two useful cost functions.

1) Linear Differences: We consider the lower bound in (6). We assume that $P(T_{|s|} < T_x)$, the probability the top item in s is picked up before x, is a decreasing function of $\hat{\mu}(x) - \hat{\mu}(s_{|s|})$, the difference between the point estimates of the respective pick-up times. This is often plausible. We define the *exponential linear difference* (ELD) cost function by

$$c_{\text{ELD}}(x, s) = e^{\hat{\mu}(x) - \hat{\mu}(s_{|s|})} \ . \tag{7}$$

It may seem arbitrary to use this specific cost, but Theorem 4 ensures that it is equivalent to any strictly monotonically increasing function of the difference $\hat{\mu}(x) - \hat{\mu}(s_{|s|})$.[1] The complexity is $O(m)$ per item.

2) Dirac Point Estimates: If we want to apply the same reasoning to whole stacks, the choice of (monotonically increasing) function is no longer clear: Theorem 4 applies only to cost functions as a whole and in general $f(\sum_i x_i) \neq \sum_i f(x_i)$. For instance, assuming $P(T_i < T_x) \propto e^{\hat{\mu}(x) - \hat{\mu}(s_i)}$ can lead to different solutions than assuming $P(T_i < T_x) \propto e^{(\hat{\mu}(x) - \hat{\mu}(s_i))/2}$.

The only way to avoid this issue is to make further assumptions. Assuming near-infinite variances yields the breadth-first algorithm.[2] Alternatively, we can assume the true distributions are Dirac distributions with full weight at the point estimate ($\mathbb{E}\{T_x\} = \hat{\mu}(x)$ and $\text{Var}(T_x) = 0$ for all x). Under this assumption, we can approximate $M(x, s)$ directly. The resulting *Dirac point estimates* (DPE) cost function is

$$c_{\text{DPE}}(x, s) \equiv \sum_{i=1}^{|s|-1} \mathcal{I}\left(\hat{\mu}(s_i) \leq \hat{\mu}(x) \text{ and } \hat{\mu}(s_i) \geq \max_{i<j\leq|s|} \hat{\mu}(s_j)\right).$$

We now show that we can compute c_{DPE} efficiently. Let

$$A(s) = \{s_i \,|\, \hat{\mu}(s_i) \geq \max_{j>i} \hat{\mu}(s_j)\}$$

be the set of *conflict-free* items, which contains all items with a later estimated pick-up time than all items higher in the same stack. The set $A(x : s)$ then consists of all items in $A(s)$ with later estimated pick-up times than $\hat{\mu}(x)$, and x itself. The estimated marginal cost under the Dirac assumption is then the difference between $|A(s)| + 1$ and $|A(x : s)|$:

$$c_{\text{DPE}}(x, s) = |A(s)| + 1 - |A(x : s)| \ ,$$

where the '$+1$' accounts for the addition of x to the set of conflict-free items. If we store $A(s)$ for each stack, we only need $|A(s)| < |s|$ comparisons to compute $A(x : s)$ and $c_{\text{DPE}}(x, s)$. Since $|s| \leq h_{\max}$, the complexity of this cost function is therefore $O(m h_{\max})$ per item.

C. Point Estimates of Mean and Variance

Next, we assume we have point estimates of the variances of the pick-up-times in addition to the mean estimates. For instance, this occurs when a Gaussian process [8], [18] is mapped on the relation of historical pick-up times and features of the items. We use $\hat{\sigma}^2(x)$ to denote the variance estimates.

[1] In fact, our first implementation used $c(x, s) = \hat{\mu}(x) - \hat{\mu}(s_{|s|})$. We switched to the current version to ensure all cost functions are non-negative.

[2] Note that $\mathbb{E}\{C(s) \,|\, \text{Var}(T) \to \infty\} = \sum_{i=1}^{|s|} \frac{|s|-i}{|s|+1-i}$ and therefore $\mathbb{E}\{M(x, s) \,|\, \text{Var}(T) \to \infty\} = |s|/(|s|+1)$. Theorem 4 implies the resulting cost function $c(x, s) = |s|/(|s|+1)$ is equivalent to $c_{\text{BF}}(x, s) = |s|$, as both increase monotonically in $|s|$.

1) Chebyshev Bounds: We first consider the lower bound in (6). Distribution-free bounds for the probability that a new item is picked up before the top item in a stack can be derived from Chebyshev's inequality, which states

$$P\left(X \geq \mathbb{E}\{X\} + k\sqrt{\mathrm{Var}(X)}\right) \leq \frac{1}{1+k^2} \ ,$$

for any random variable X and $k \geq 0$. For $k < 0$ we can use the trivial bound $P(X \geq \mathbb{E}\{X\} + k\sqrt{\mathrm{Var}(X)}) \leq 1$. Let us define arbitrary random variables $\hat{T}_x$ such that $\mathbb{E}\{\hat{T}_x\} = \hat{\mu}(x)$ and $\mathrm{Var}(\hat{T}_x) = \hat{\sigma}^2(x)$. Define

$$\hat{p}(x,y) \equiv \begin{cases} \frac{\hat{\sigma}^2(x)+\hat{\sigma}^2(y)}{\hat{\sigma}^2(x)+\hat{\sigma}^2(y)+(\hat{\mu}(x)-\hat{\mu}(y))^2} & \text{if } \hat{\mu}(x) \leq \hat{\mu}(y), \\ 1 & \text{otherwise.} \end{cases} \tag{8}$$

This implies an approximate upper bound

$$P\left(T_x > T_y\right) \approx P\left(\hat{T}_x > \hat{T}_y\right) \leq \hat{p}(x,y) \ ,$$

for the probability that some item y is picked up before item x. Note that generally $\hat{p}(x,y) \neq \hat{p}(y,x)$. We combine this approximate bound with the lower bound in (6) to construct the *upper Chebyshev* (UC) cost function

$$c_{\mathrm{UC}}(x,s) = \hat{p}(x, s_{|s|}) \ .$$

With only an upper bound on the probability $P(T_{|s|} < T_x)$ we cannot differentiate between stacks where $\hat{\mu}(x) > \hat{\mu}(s_{|s|})$; in those cases $\hat{p}(x, s_{|s|}) = 1$. However, we can approximately lower bound $P(T_x < T_{|s|})$ with $1 - \hat{p}(s_{|s|}, x)$. This gives us the *lower Chebyshev* (LC) cost

$$c_{\mathrm{LC}}(x,s) = 1 - \hat{p}(s_{|s|}, x) \ .$$

Note that $c_{\mathrm{LC}}(x,s) > 0$ if and only if $c_{\mathrm{UC}}(x,s) = 1$, since these cost functions cover non-overlapping cases. We can combine these to get the *combined Chebyshev* (CC) cost

$$c_{\mathrm{CC}}(x,s) = 1 + \hat{p}(x, s_{|s|}) - \hat{p}(s_{|s|}, x) \ . \tag{9}$$

For constant variances, $c_{\mathrm{CC}}(x,s)$ is a strictly increasing function of the difference $\hat{\mu}(x) - \hat{\mu}(s_{|s|})$, and therefore equivalent to $c_{\mathrm{ELD}}(x,s)$. In general, $c_{\mathrm{CC}}(x,s)$ uses more information.

It may seem arbitrary to combine these two bounds in a single cost function. However, the following theorem shows an important general relation to the assumption of normality.

Theorem 5. *When $\hat{\sigma}^2(x) + \hat{\sigma}^2(s) > 0$, using cost $c_{\mathrm{CC}}(x,s)$ is equivalent to using any cost that is a strictly monotonically increasing function of the t-statistic $t(x, s_{|s|})$ of the difference between the estimated pick-up times of the top item and the new item, where*

$$t(x,y) = \frac{\hat{\mu}(x) - \hat{\mu}(y)}{\sqrt{\hat{\sigma}^2(x) + \hat{\sigma}^2(y)}} \ . \tag{10}$$

This theorem implies that using c_{CC} is an efficient way to obtain solutions consistent with the assumption that the pick-up time of each x is sampled from a normal distribution with mean $\hat{\mu}(x)$ and variance $\hat{\sigma}^2(x)$. However, it still considers only the top item in each stack, resulting in a low $O(m)$ complexity per item, but potentially harming the performance.

Algorithm 2 A $O(h_{\max} d)$ algorithm to compute $c_{\mathrm{ED}}(x,s)$.

Require: Labeled sorted data sets D_x, D_i for $1 \leq i \leq |s|$.
  Set $c = 0$, $\hat{F}_x = 0$, $\hat{F}_i = 0$ for all $1 \leq i \leq |s|$
  Sort $D = D_x \cup \bigcup_{i=1}^{|s|} D_i$ increasingly on pick-up time
  Remove duplicates from D
  for $t \in D$ **do**
    $\hat{F}_x \leftarrow \hat{F}_x + \mathcal{I}(t \in D_x)/|D_x|$
    for $i \in \{|s|-1, \ldots, 1\}$ **do**
      $\hat{F}_{i+1} \leftarrow \hat{F}_{i+1} + \mathcal{I}(t \in D_{i+1})/|D_{i+1}|$
      $P \leftarrow \hat{F}_{i+1} \cdot P$
      $c \leftarrow c + (1 - \hat{F}_x) \cdot P \cdot \mathcal{I}(t \in D_i)/|D_i|$
    end for
    $c \leftarrow c + \hat{F}_{|s|} \cdot \mathcal{I}(t \in D_x)/|D_x|$
  end for
  return c

2) Summed Chebyshev Bounds for Whole Stacks: For computationally-efficient algorithms that consider the whole stack, we use the upper bound in (6). We can apply the same Chebyshev bounds as before for each item in the stack. The resulting *summed combined Chebyshev* (SCC) cost is

$$c_{\mathrm{SCC}}(x,s) = |s| + \sum_{i=1}^{|s|} \hat{p}(x, s_i) - \hat{p}(s_i, x) \ .$$

Similarly, we define the *summed upper Chebyshev* (SUC) cost $c_{\mathrm{SUC}}(x,s) = \sum_i \hat{p}(x, s_i)$ and the *summed lower Chebyshev* (SLC) cost $c_{\mathrm{SLC}}(x,s) = |s| - \sum_i \hat{p}(x, s_i)$. The complexity of all these algorithms is $O(m h_{\max})$.

3) Normal Approximations for Whole Stacks: We now assume the pick-up time of each item x follows a normal distribution with mean $\hat{\mu}(x)$ and variance $\hat{\sigma}^2(x)$. This assumption or normality gives us approximations of the PDFs and CDFs of the pick-up times, which we can input into equation (5) to approximate $M(x,s)$ directly. This yields the *normal approximating* (NA) cost function $c_{\mathrm{NA}}(x,s)$. Analytic solutions to this integral generally do not exist, so we need to use a numeric approximation. We use a Gaussian quadrature [19]. This cost function is the most computationally expensive of all the cost functions we consider. The complexity is $O(m h_{\max} z^2)$ per item, where z is the number of nodes used in the numerical integration, which affects the accuracy. Rather than setting a specific value for z, in the experiments we required the estimated absolute error of the integration to be below 0.01, which is a fairly high value. Still, computation time for this cost was at least two orders of magnitude larger than for any of the other cost functions, which may become prohibitive when computational resources are scarce or the number of stacks and items is high.

D. Empirical Distributions

We now assume that for each item x we have a multiset D_x of $d_x \equiv |D_x|$ samples from the real distribution. For instance, this applies when the items are classifiable into distinct classes with data sets of historical pick-up times for each class.

The empirical distribution $\hat{F}_x(t) = |\{j \in D_x \,|\, j \leq t\}|/d_x$ for an item x is the ratio of pick-up times in D_x that are earlier than t. The corresponding probability mass function is $\hat{f}_x(t) =$

TABLE I. PROPERTIES OF THE COST FUNCTIONS FROM SECTION III.

cost	req. inf.	approximation	complexity per item
c_{R}	–	–	$O(m)$
c_{BF}	–	direct, using (5)[a]	$O(m)$
c_{DF}	–	–	$O(m)$
c_{ELD}	$\hat{\mu}$	lower bound (6)	$O(m)$
c_{DPE}	$\hat{\mu}$	direct, using (5)	$O(mh_{\max})$
c_{LC}	$\hat{\mu}$,$\hat{\sigma}^2$	lower bound (6)	$O(m)$
c_{UC}	$\hat{\mu}$,$\hat{\sigma}^2$	lower bound (6)	$O(m)$
c_{CC}	$\hat{\mu}$,$\hat{\sigma}^2$	lower bound (6)	$O(m)$
c_{SLC}	$\hat{\mu}$,$\hat{\sigma}^2$	upper bound (6)	$O(mh_{\max})$
c_{SUC}	$\hat{\mu}$,$\hat{\sigma}^2$	upper bound (6)	$O(mh_{\max})$
c_{SCC}	$\hat{\mu}$,$\hat{\sigma}^2$	upper bound (6)	$O(mh_{\max})$
c_{NA}	$\hat{\mu}$,$\hat{\sigma}^2$	direct, using (5)	$O(mh_{\max}z^2)$[b]
c_{ED}	D	direct, using (5)	$O(mh_{\max}d)$

[a] See Section III-B2.

[b] Here z determines the accuracy of the numerical integration.

$|\{j \in D_x \,|\, j = t\}|/d_x$. We can plug $\hat{F}$ and $\hat{f}$ into equation (5). The function within the integral is zero almost everywhere, except on the times in the data set D. Some rewriting gives us the *empirical distribution* cost

$$c_{\text{ED}}(x,s) = \sum_{t \in D_x} \frac{\hat{F}_{|s|}(t)}{d_x} + \sum_{i=1}^{|s|-1} \sum_{t \in D_i} \frac{(1-\hat{F}_x(t)) \prod_{j=i+1}^{|s|} \hat{F}_j(t)}{d_i} \ .$$

When $d_i = d_x = d$ for all i, a naive implementation of this is in $\Theta(h_{\max}d^2)$, since computing each $\hat{F}_i(t)$ is in $\Theta(d)$. However, when the samples in each D_x are sorted, $c_{\text{ED}}(x,s)$ can be computed in $\Theta(h_{\max}d)$ computations per item, using Algorithm 2. When D_x must be sorted in real-time for each new x, the complexity is $O(h_{\max}d \log d)$.

When d is large, we can use subsets to reduce computational complexity at the cost of a less accurate approximation. This might not decrease performance when the costs of the stacks are sufficiently far apart. Furthermore, we can focus the computation on low-cost stacks, since we do not need accurate estimates for high-cost stacks. Detailed considerations of such extensions fall outside the scope of this paper.

E. In Summary

Table I shows all the cost functions defined above, with the required information and used approximation for each of them. Since one can construct $\hat{\mu}$ and $\hat{\sigma}$ from D, access to the latter is the strongest of the four assumptions.

IV. EXPERIMENTS

In this section, we compare the proposed algorithms on a total of 258 experiments.

A. Experimental setup

For each experiment we define m and $h_{\max}$, and we explain how the real pick-up-time distributions F are constructed. We then draw 101 samples from F. The first 100 samples form a sample set D. Its sample average and (unbiased) sample variance are used as the estimated pick-up time $\hat{\mu}$ and the estimated variance $\hat{\sigma}^2$. The last sample is used as the actual pick-up time t of the item. We test the online performance when all items arrive in random order, and the batch performance when the items arrive in order of decreasing estimated pick-up times.

In our first 128 experiments there are $m \in \{2,4,8,16\}$ stacks of equal maximum capacity $h_{\max} \in \{2,4,8,16\}$. For each of the 16 possible combinations of m and $h_{\max}$, we generate $\lceil omh_{\max} \rceil$ items where $o = 0.9$ is the occupancy ratio. The experiments therefore range from placing $\lceil 0.9 \cdot 2 \cdot 2 \rceil = 4$ items in 2 stacks of capacity 2, to placing $\lceil 0.9 \cdot 16 \cdot 16 \rceil = 231$ items in 16 stacks of capacity 16. To construct the actual pick-up time distribution F_x for each x, we sample μ_x from a standard normal distribution and τ_x from a Gamma distribution with shape parameter k and a scale parameter θ. Each F_x is defined to be a normal distribution with mean μ_x and variance $(\lambda\tau_x)^{-1}$, where λ is a third parameter. We used $\theta = 1$ and $\lambda, k \in \{1, 10\}$, yielding four different possible combinations, for a total of $2 \cdot 4^3 = 128$ experiments.

The second 128 experiments are similar to the first 128, but with a different way to construct F_x. For each x, we generate two samples $\mu_{x,1}$ and $\mu_{x,2}$ from a standard normal distribution and two samples $\tau_{x,1}$ and $\tau_{x,2}$ from Gamma distribution with parameters k and θ. Additionally, we sample $w_x \sim U(0,1)$ uniformly from $[0,1]$. Each F_x is a mixture of two normal distributions $\mathcal{N}_{x,1} = \mathcal{N}(\mu_{x,1}, (\lambda\tau_{x,1})^{-1})$ and $\mathcal{N}_{x,2} = \mathcal{N}(\mu_{x,2}, (\lambda\tau_{x,2})^{-1})$ with weights w_x and $1 - w_x$, respectively. To sample from F_x, we can sample from $\mathcal{N}_{x,1}$ with probability w_x and from $\mathcal{N}_{x,2}$ with probability $1 - w_x$. We use these mixtures of Gaussians to test how the results change when the distributions F are not normal. The overlap in possible pick-up times, and correspondingly the probability of conflicts, is higher than in the first set of experiments.

For both sets of experiments, the combination $\lambda = k = 1$ yields the most overlap in the pick-up times, and $\lambda = k = 10$ yields the least overlap. To quantify the 'hardness' of the different settings, we consider the probability of conflicts if we stack two random items based only on the estimated pick-up time. In other words, if we construct two distributions F_1 and F_2 as described, and $\{x_{i,1}, \ldots, x_{i,d}, t_i\} \sim F_i^{d+1}$ and $\hat{\mu}_i = \frac{1}{d}\sum_{j=1}^{d} x_{i,j}$, what is then $\mathbb{E}\{P(t_1 > t_2 \,|\, \hat{\mu}_1 \leq \hat{\mu}_2) \,|\, \lambda, k\}$, where the expectancy is over the randomness in constructing F_1 and F_2? Table II gives these probabilities for $d = 100$, showing that 1) these probabilities are diverse, covering $[0.03, 0.36]$ out of a possible range of $[0, 0.5]$, thus indicating a good range of different scenarios, and 2) the bi-modal process on average indeed produces more unavoidable conflicts.

The last 2 experiments more closely resemble the real-world setting at the terminal in the Port of Rotterdam that inspired this paper. There are a few hundred suitable ground spaces, not all of which will be available at any time. We therefore use $m = 100$. Each stack can contain at most $h_{\max} = 3$ reefers. To construct F_x we draw a mean μ_x uniformly randomly from $[6, 18]$, representing hours of the day. We draw a width w_x uniformly randomly from $[\frac{1}{2}, 12]$, representing a pick-up time window for a specific reefer. Each F_x is defined by the uniform distribution $U(\mu_x - \frac{w_x}{2}, \mu_x + \frac{w_x}{2})$. For instance, if $\mu_x = 10$ and $w_x = 2$, then x is picked up between 9am and 11am. We generate $\lceil 0.9 \cdot 100 \cdot 3 \rceil = 270$

TABLE II. THE EXPECTED PROBABILITY OF INCORRECTLY ORDERED ESTIMATES $P(\lambda, k) \equiv \mathbb{E}\{P(t_1 > t_2 \,|\, \hat{\mu}_1 \leq \hat{\mu}_2) \,|\, \lambda, k\}$.

uni-modal distribution			bi-modal distribution		
k	λ	$P(\lambda, k)$	k	λ	$P(\lambda, k)$
1	1	0.31	1	1	0.36
1	10	0.16	1	10	0.26
10	1	0.10	10	1	0.22
10	10	0.03	10	10	0.20

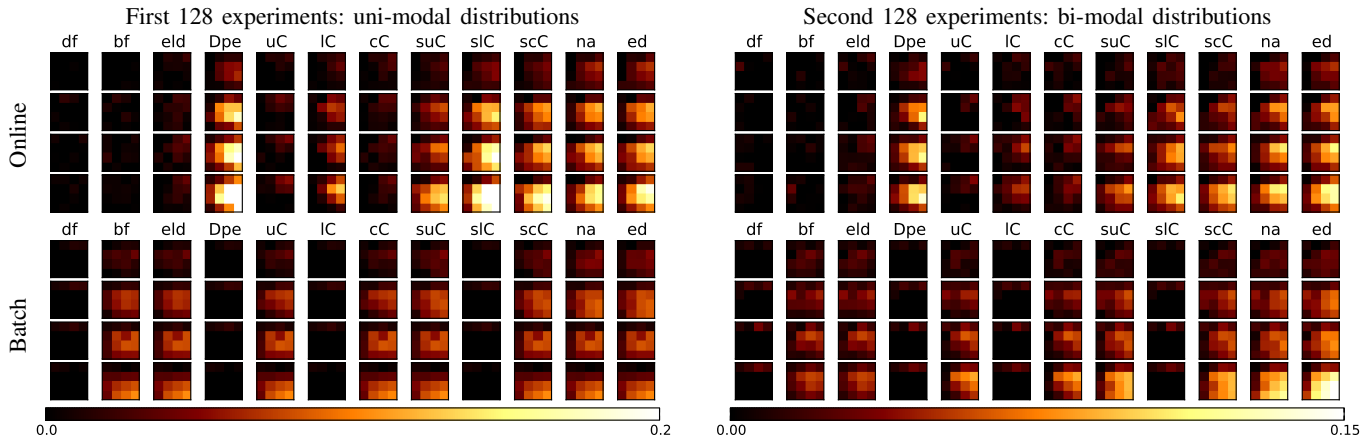

Fig. 1. The shown values are the reduced chance of a conflict per item, compared to a random stacking. The left plots are for the uni-modal generating process, the right plots for the bi-modal generating process. The top plots are for the online experiments; the bottom plots for the batch experiments. For each combination of a generating process and a online/batch setting, the results for each algorithm are shown in four 4x4 heat maps. Each 4x4 heat map corresponds to specific parameters for the Gamma-distribution that regulate the expected overlap in the distributions. Lower heat maps (within each row of plots) means *less* overlap, which implies conflicts are more easily avoided. Within each individual 4x4 heat map, each row corresponds to a specific capacity $h_{\max}$ and each column corresponds to a specific m; further down means *heighers* stacks and further right means *more* stacks. The values are averaged over 100 repetitions of the experiments.

items to be stacked, and we conduct an online and a batch experiment.

B. Results

Figure 1 shows the results for the first 256 experiments. The values correspond to the reduction in probability of a conflict per item for each cost as compared to the random cost. For instance, if stacking 60 items randomly results in 20 conflicts while breadth first results in 14 conflicts, the value for BF for that experiment is $(20-14)/60 = 0.1$. In Figure 1, brighter colors correspond to larger reductions; brighter is better. In some black sections—such as for much of the depth-first algorithm—performance was worse than random. The standard errors are all smaller than 0.02, and close to 0.005 for larger numbers of items ($m \geq 8$, $h_{\max} \geq 8$). Concretely, this implies that all easily perceivable differences in Figure 1 are statistically significant.

The left group of plots corresponds to the uni-modal generating process, the right group corresponds to the bi-modal process. Within each group, the top plots shows the online results and the bottom plots show the batch results. For each combination of generating process, type of ordering of the items (online or batch) and algorithm, four 4×4 heat maps are shown. The four heat maps correspond to the different combinations of λ and k. The rows within each 4×4 heat map correspond to $h_{\max}$, the columns to m. For instance, for online uni-modal c_{DPE} (top left, with label DPE) the brightest colors are found in the fourth 4×4 heat map, corresponding to $\lambda = k = 10$. Within this heat map, rows 2 and 3 are brightest, corresponding to $h_{\max} = 4$ and $h_{\max} = 8$. Finally, within these rows, the brightest values are located at the right, corresponding to $m = 16$. In fact, the highest observed difference to random is for $h_{\max} = 8$, $m = 16$, $\lambda = k = 10$, where using c_{DPE} decreases the average number of conflicts per item from 0.58 (for random) to 0.24. This means that when stacking these $\lceil 0.9 \cdot 16 \cdot 8 \rceil = 116$ items, on average random creates 67 conflicts, while c_{DPE} only creates 28 conflicts.

1) Online Experiments: In online version of both the uni-modal setting (top left) and bi-modal setting (top right), c_{DPE} is clearly the best choice when only mean estimates are available. Even when more information is available it remains a good choice, performing similar to c_{ED}. There is a clear difference between methods that consider only the top item in each stack and methods that consider the whole stack; the summed Chebyshev costs clearly outperform the Chebyshev costs that only consider the top item in each stack. The summed lower Chebyshev bounds work best, outperforming even the combined bound although the difference is not substantial. It does not seem worth while to spend the extra computational resources to compute c_{NA}. Although the empirical-distribution cost function performs well for both generating processes and outperforms the Chebyshev costs in the bi-modal setting, the difference to c_{DPE} is, surprisingly, not substantial.

2) Batch Experiments: In the batch experiments, there is a clear difference between the uni-modal (bottom left) and the bi-modal (bottom right) distributions. With uni-modal distributions, it is hard to beat breadth first. In the bi-modal setting it does pay to spend the extra resources to compute c_{SCC} or c_{ED}. The reason that some of the best online algorithms (c_{DPE}, c_{LC} and c_{SLC}) perform poorly in the batch setting is that these costs can not differentiate between stacks when all stacked items have later estimated pick-up times than the new item to be placed, which is always true in the batch setting. In our implementation, these algorithms then simply choose the first non-full stack, therefore defaulting to a policy equivalent to depth first. The combined Chebyshev costs avoid this problem and perform well in both online and batch settings, although if samples are available c_{ED} is slighter better.

3) Port-of-Rotterdam Experiments: The results for the last two experiments are shown in Table III. The table shows the average number of conflicts; lower numbers are better.

The capacity was $h_{\max} = 3$, so each stack that is considered when an item is to be placed can already contain at most two items. Still, in the online setting, costs that consider both these items outperform costs that only look at the top item; for

TABLE III. AVERAGE NUMBER OF CONFLICTS PER ALGORITHM. LOWER IS BETTER. VALUES ARE AVERAGED OVER 100 REPETITIONS OF THE EXPERIMENT.

	R	DF	BF	ELD	DPE	UC	LC	CC	SUC	SLC	SCC	NA	ED
	online												
average cost	89.0	90.1	86.4	81.0	66.7	74.8	72.7	74.5	63.6	70.3	57.8	60.3	58.3
std err	0.5	0.6	0.6	0.6	0.6	0.5	0.7	0.6	0.5	0.7	0.4	0.5	0.5
	batch												
average cost	27.3	40.1	10.4	10.4	40.1	7.4	40.1	7.4	7.5	40.1	7.5	7.4	7.5
std err	0.5	0.5	0.3	0.3	0.5	0.3	0.5	0.3	0.2	0.5	0.2	0.3	0.3

instance compare c_{CC} to c_{SCC}. Online, c_{SCC}, c_{ED} and c_{NA} (in that order) perform best, although c_{DPE} is not bad when only mean estimates are available. The best algorithms are substantially better than random, reducing the number of conflicts from about 90 to about 60. The difference between having some information (c_{DPE}) and no information (c_{BF}) is also substantial.

In the batch setting, the differences in performance are again substantial. Even the simple breadth-first algorithm already reduces the number of conflicts from over about 27 to about 10. A further reduction to somewhere around 7 conflicts—only a quarter of the number of conflicts for the random algorithm—is possible when we have point estimates of the variance. Again, it does not seem worth while to compute c_{NA}, and even the computationally cheap costs c_{UC} and c_{CC} perform very well, likely because of the relatively low stacks. As explained above, in the batch setting c_{DPE}, c_{LC} and c_{SLC} are equivalent to the poorly performing depth-first algorithm.

Even though the distributions are not normal, we can perform well with just mean and variance rather than a whole empirical distribution. If possible, using c_{ED} is a relatively safe bet. However, if data sets are not available or inefficient to store, the Chebyshev costs are a good alternative, especially since Gaussian processes can generalize more easily over continuous features. In such cases it may be hard to separate classes for which samples can be collected. In any case, the possible reductions in conflicts are substantial, and could result in much more efficient handling of peak traffic at the terminal.

V. CONCLUSION

It is possible to avoid a large number of conflicts by using intelligent stacking algorithms, rather than a simple breadth-first or random approach. Substantial improvements are possible in both online and batch settings, as demonstrated over a width range of cases. The difference between the best algorithms and simple random and breadth-first algorithms is often substantial. Such reductions can potentially make the difference between costly traffic jams and much more manageable conditions at the Port of Rotterdam.

Which algorithms are applicable depends on the available information. The Dirac-point-estimates (DPE) algorithm, which uses only estimates of the mean pick-up time, performs surprisingly well for items that arrive sequentially in random order. When the items are processed in batch and we can sort them before stacking, Chebyshev bounds that use variance estimates result in good stacking algorithms. These algorithms often even outperform a (computationally much costlier) algorithm based on normal approximations. The proposed empirical-distributions algorithm performs well overall, but requires sets of samples that may not always be available. Most proposed algorithms are efficient enough to run real-time on (very) large problems. Even with just noisy estimates of expected pick-up times it is possible to do much better than a random or breadth-first approach with a minimum of computation. With uncertainty (variance) estimates we can perform even better.

In future work, we intend to further analyze the proposed methods. An interesting question is how the algorithms hold up when their information is biased. In that light, the current results are promising: the DPE algorithm effectively assumes the variances are zero, but it still performs well.

APPENDIX

For conciseness we introduce the definitions

$$A_i \equiv (T_i < \max_{i<j\leq|s|} T_j)) \quad \text{and} \quad B_i \equiv (T_i < T_x) \ . \tag{11}$$

Proof of Lemma 1: Let $C_1(s)$ be the cost defined in (1) and let $C_2(s)$ be the cost in (3), such that

$$C_1(s) = C_1(s \backslash s_{(1)}) + \mathcal{I}\left(A_{(1)}\right), \text{ and } C_2(s) = \sum_{i=1}^{|s|-1} \mathcal{I}\left(A_i\right) \ .$$

For any k

$$\begin{aligned} C_2(s) &\equiv \sum_{i=1}^{|s|-1} \mathcal{I}\left(A_i\right) = \sum_{i=1}^{|s|-1} \mathcal{I}\left(A_i\right) - \mathcal{I}\left(A_k\right) + \mathcal{I}\left(A_k\right) \\ &= C_2(s \setminus s_k) + \mathcal{I}\left(A_k\right) \ . \end{aligned}$$

This includes $k = (1)$, so (1) and (3) are equivalent. ■

Proof of Theorem 1: The probability that s_i does not cause a conflict when $T_i = t$ is equal to the probability that all higher items in the stack are picked up later than t: $P\left(A_i \,|\, t_i = t\right) = \prod_{j=i+1}^{|s|} 1 - F_j(t)$. We integrate out t to get

$$P\left(A_i\right) = \int_{-\infty}^{\infty} f_i(t) \left(1 - \prod_{j=i+1}^{|s|} F_j(t)\right) \, dt \ .$$

Taking the expectation of (3) we then get

$$\begin{aligned} \mathbb{E}\left\{C(s)\right\} &= \sum_{i=1}^{|s|-1} P(A_i) \\ &= \sum_{i=1}^{|s|-1} \left(1 - \int_{-\infty}^{\infty} f_i(t) \prod_{j=i+1}^{|s|} F_j(t) \, dt\right) \\ &= |s| - 1 - \int_{-\infty}^{\infty} \sum_{i=1}^{|s|-1} f_i(t) \prod_{j=i+1}^{|s|} F_j(t) \, dt \ . \end{aligned}$$

■

Proof of Theorem 2: By Theorem 1,

$$\begin{aligned}
&\mathbb{E}\left\{C(x:s)\right\} - \mathbb{E}\left\{C(s)\right\} \\
&= |s| - \int_{-\infty}^{\infty} F(t|x) \sum_{i=1}^{|s|} f_i(t) \prod_{j=i+1}^{|s|} F_j(t)\, dt \\
&\quad - |s| + 1 + \int_{-\infty}^{\infty} \sum_{i=1}^{|s|-1} f_i(t) \prod_{j=i+1}^{|s|} F_j(t)\, dt \\
&= 1 + \int_{-\infty}^{\infty} (1 - F_x(t)) \sum_{i=1}^{|s|-1} f_i(t) \prod_{j=i+1}^{|s|} F_j(t) - F_x(t) f_{|s|}(t)\, dt\,.
\end{aligned}$$

By integration by parts,

$$1 - \int_{-\infty}^{\infty} F(t|x) f(t|s_{|s|})\, dt \;=\; \int_{-\infty}^{\infty} f(t|x) F(t|s_{|s|})\, dt \quad .$$

Plugging this back in yields the desired result. ∎

Proof of Theorem 3: By definitions (3) and (11)

$$\mathbb{E}\left\{C(x:s)\right\} - \mathbb{E}\left\{C(s)\right\} = \sum_{i=1}^{|s|} P\left(A_i \text{ or } B_i\right) - P\left(A_i\right) \quad ,$$

where $P\left(A_{|s|}\right) = 0$ and $P\left(A_{|s|} \text{ or } B_{|s|}\right) = P\left(B_{|s|}\right)$.

For any A and B,

$$\begin{aligned}
&P\left(A \text{ or } B\right) - P\left(A\right) = P\left(B\right) - P\left(A \text{ and } B\right) \\
&= P\left(B\right)\left(1 - P\left(A \,|\, B\right)\right) = P\left(B\right) P\left(\neg A \,|\, B\right) \quad .
\end{aligned}$$

Therefore,

$$M(x,s) = P\left(B_{|s|}\right) + \sum_{i=1}^{|s|-1} P\left(B_i\right) P\left(\neg A_i \,|\, B_i\right) \quad .$$

The claimed inequalities then follow immediately by noting $0 \leq P\left(\neg A_i \,|\, B_i\right) \leq 1$. ∎

Proof of Theorem 4: Our generic algorithm places x on s_*, where $c(x, s_*) = \min_s c(x, s)$. For any strictly monotonic increasing function f

$$s_* = \arg\min_s c(x,s) = \arg\min_s f(c(x,s)) \quad ,$$

which directly implies the claimed result. ∎

Proof of Theorem 5: Note that $\hat{p}(x,y)$ increases strictly with $t(x,y)$ for $t(x,y) \leq 0$ and is constant for $t(x,y) > 0$. Similarly, $1 - \hat{p}(y,x)$ increases strictly with $t(x,y)$ for $t(x,y) > 0$ and is constant for $t(x,y) \leq 0$. Therefore, $c_{cC}(x,s)$ increases strictly with $t(x, s_{|s|})$ over the whole number line. Now apply Theorem 4, and we are done. ∎

REFERENCES

[1] H. Günther and K. Kim, "Container terminals and terminal operations," *OR Spectrum*, vol. 28, no. 4, pp. 437–445, 2006.

[2] C. M. Bishop, *Pattern recognition and machine learning*. Springer New York:, 2006.

[3] T. Hastie, R. Tibshirani, and J. Friedman, *The elements of statistical learning: data mining, inference and prediction*, 2nd ed. Springer, 2009.

[4] Z. Ghahramani, "Unsupervised learning," *Advanced Lectures on Machine Learning*, pp. 72–112, 2004.

[5] T. Kohonen, *Self-organizing maps*. Springer Verlag, 2001, vol. 30.

[6] C. M. Bishop, *Neural networks for pattern recognition*. Oxford University Press, USA, 1995.

[7] M. Gibbs, "Bayesian Gaussian processes for regression and classification," Ph.D. dissertation, University of Cambridge, 1997.

[8] C. Rasmussen and C. Williams, *Gaussian processes for machine learning*. MIT press Cambridge, MA, 2006, vol. 1.

[9] M. Avriel and M. Penn, "Exact and approximate solutions of the container ship stowage problem," *Computers & industrial engineering*, vol. 25, no. 1, pp. 271–274, 1993.

[10] P. Giemsch and A. Jellinghaus, "Optimization models for the container-ship stowage problem," in *Operations Research Proceedings 2003: Selected Papers of the International Conference on Operations Research, Heidleberg, September 3-5, 2003*. Springer Verlag, 2004, p. 347.

[11] I. Wilson and P. Roach, "Principles of combinatorial optimization applied to container-ship stowage planning," *Journal of Heuristics*, vol. 5, no. 4, pp. 403–418, 1999.

[12] R. Dekker, P. Voogd, and E. Asperen, "Advanced methods for container stacking," *Container terminals and cargo systems*, pp. 131–154, 2007.

[13] T. Winter and U. Zimmermann, "Real-time dispatch of trams in storage yards," *Annals of Operations Research*, vol. 96, no. 1, pp. 287–315, 2000.

[14] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. The MIT press, Cambridge MA, 1998.

[15] H. P. van Hasselt and M. A. Wiering, "Reinforcement learning in continuous action spaces," in *Proceedings of the IEEE International Symposium on Adaptive Dynamic Programming and Reinforcement Learning*, 2007, pp. 272–279.

[16] ——, "Using continuous action spaces to solve discrete problems," in *Proceedings of the International Joint Conference on Neural Networks (IJCNN 2009)*, 2009, pp. 1149–1156.

[17] H. P. van Hasselt, "Reinforcement learning in continuous state and action spaces," in *Reinforcement Learning: State of the Art*, ser. Adaptation, Learning, and Optimization, M. A. Wiering and M. van Otterlo, Eds. Springer, 2012, vol. 12, pp. 207–251.

[18] C. Williams, "Prediction with Gaussian processes: from linear regression to linear prediction and beyond," in *Learning in graphical models*. MIT Press, 1999, pp. 599–621.

[19] M. Abramowitz and I. Stegun, *Handbook of mathematical functions: with formulas, graphs, and mathematical tables*, ser. National Bureau of Standards Applied Mathematics Series. Dover publications, 1972, vol. 55.